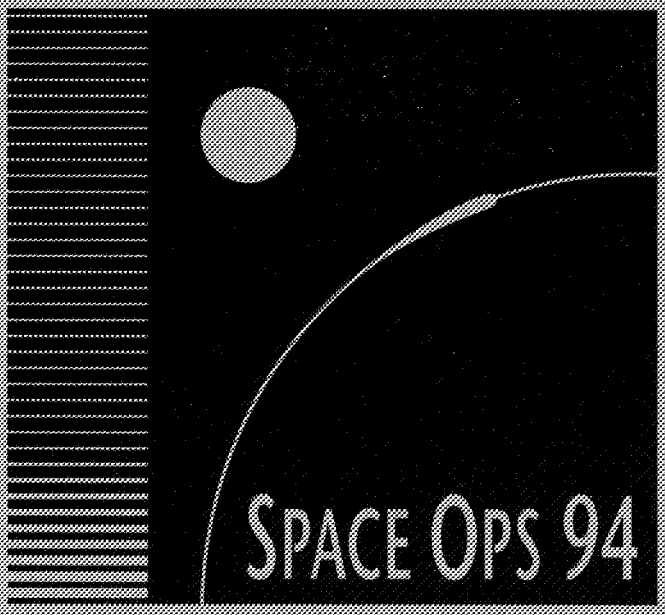
NASA Conference Publication 3281

Third International Symposium on Space Mission Operations and Ground Data Systems Part 2

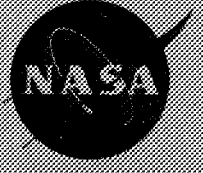


Proceedings of a conference held at
Greenbelt Marriott Hotel
Greenbelt, Maryland, USA
November 15-18, 1994

(NASA-CP-3281-Pt-2) THIRD INTERNATIONAL SYMPOSIUM ON SPACE MISSION OPERATIONS AND GROUND DATA SYSTEMS, PART 2 (NASA. Goddard Space Flight Center) 700 p

N95-17531
--THRU--
N95-17614
Unclas

H1/17 0031936



11116

353934

NASA Conference Publication 3281

P.21

Third International Symposium on Space Mission Operations and Ground Data Systems Part 2

Edited by
James L. Rash
NASA Goddard Space Flight Center
Greenbelt, Maryland

Proceedings of a conference held at
Greenbelt Marriott Hotel
Greenbelt, Maryland, USA
November 15-18, 1994

National Aeronautics
and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

1994

This publication is available from the NASA Center for AeroSpace Information,
800 Elkridge Landing Road, Linthicum Heights, MD 21090-2934, (301) 621-0390.

EXECUTIVE SUMMARY

The Third International Symposium on Space Mission Operations and Ground Data Systems (SpaceOps 94) is being held November 14-18, 1994, in Greenbelt Maryland, USA, and is hosted by the NASA Goddard Space Flight Center. More than 400 people from nine countries are attending. This symposium follows the Second International Symposium that was hosted by the Jet Propulsion Laboratory in Pasadena, California, during November 1992. The First International Symposium on Ground Data Systems for Spacecraft Control, conducted in June 1990, was sponsored by the European Space Agency and the European Space Operations Centre.

The theme of this Third International Symposium is "Opportunities in ground data systems for high efficiency operations of space missions". Accordingly, the Symposium features more than 150 oral presentations in five technical tracks:

- Mission Management
- Operations
- Data Management
- Systems Engineering
- Systems Development

These five tracks are subdivided into over 50 sessions, each containing three presentations. The presentations focus on improvements in the efficiency, effectiveness, productivity, and quality of data acquisition, ground systems, and mission operations. New technology, techniques, methods, and human systems are discussed. Accomplishments are also reported in the application of information systems to improve data retrieval, reporting, and archiving; the management of human factors; the use of telescience and teleoperations; and the design and implementation of logistics support for mission operations.

FOREWORD

We welcome you to SpaceOps 94! The Goddard Space Flight Center is pleased to host and sponsor our biennial symposium this year. We intend to maintain the same high standards set by our predecessors--the Jet Propulsion Laboratory in 1992, and the European Space Agency with the European Space Operations Centre in 1990.

Like other participating organizations, we benefit from the shared knowledge and combined experiences that are topics of discussion at the SpaceOps 94 symposium. Best of all, we benefit from seeing each other face-to-face and having the opportunity to discuss in person technical issues of mutual, often compelling interest.

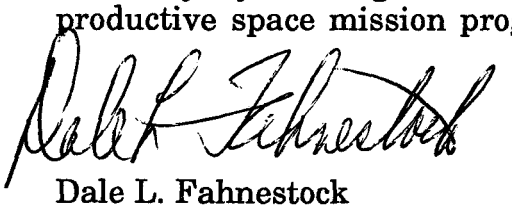
The large number of papers submitted to the SpaceOps 94 committee for acceptance and the projected attendance of over 400 of our colleagues should mean we are in for another splendid symposium this year. We believe these numbers mean that biennial meetings of our international space mission operations community are needed and are viewed as productive.

During the four days of our Symposium, more than 400 people from nine countries will hear more than 150 papers presented, as well as keynote, plenary, and panel talks by individuals from throughout the world. The papers in this proceedings document describe a wide range of ideas and experiences in our field that are developed from the perspectives of international space programs and their supporting industries.

Our review of the papers indicates that future space mission operations will be strongly influenced by the following kinds of challenges and objectives:

- Empowering operators to perform at higher intellectual levels by the increased use of artificial intelligence
- Standardizing protocols, formats, databases, and operations to enable simultaneous and economical support of multiple missions
- Dealing with the science data avalanche
- Converting yesterday's and today's mission experiences into the "corporate knowledge" databases of tomorrow
- Sharing national resources in cooperative space ventures.

We wish you a rewarding week. We also wish for, and look forward to, greater interaction between our people and our countries--not just at our symposia, but in our everyday working world as we learn to achieve increasingly successful and productive space mission programs.



Dale L. Fahnestock
General Chair

Donald D. Wilson
Executive Committee Chair

PRECEDING PAGE BLANK NOT FILMED

PAGE iv INTENTIONALLY BLANK

PREFACE

I would like to acknowledge the fine support of Laura Capella, Todd Del Priore, and April Johnson in the preparation of the manuscript for this document, which included entering data and creating FileMaker Pro scripts on the Macintosh computer to produce the the table of contents and author index.

If you have Internet access, I invite you to navigate to the NASA "Hot Topics" page using URL address http://hypatia.gsfc.nasa.gov/NASA_homepage.html. Possibly, using this path, you already may have accessed the World Wide Web information pages on SpaceOps 94, and we solicit your comments on what you find there. It is reasonable to assume that the call for papers and other information on the next SpaceOps (in 1996) will be similarly accessible a few months in advance. Please inform potentially interested colleagues regarding this information resource.

James L. Rash/NASA/GSFC
Editor
Publications Committee Chair

PRECEDING PAGE BLANK NOT FILMED

PAGE VI INTENTIONALLY BLANK

1994 SYMPOSIUM EXECUTIVE COMMITTEE

Dale L. Fahnestock
General Chair

Donald D. Wilson
Executive Committee Chair

Peter M. Hughes
Technical Program

J. Michael Moore
Paper Selection

James L. Rash
Publications

Jessica N. Katz
Logistics and Registration

Roberta A. Valonis
Publicity and Technical Tours

Michael R. Bracken
Exhibitions and Industry Liaison

Lori M. Keehan
Demonstrations and Industry Liaison

Sajjad H. Durrani
Technical Program Support

Richard V. Nguyen
Technical Program Support

Roger E. Wye
Technical Program Support

SPACEOPS EXECUTIVE COMMITTEE

Charles F. Fuechsel
Co-Chair
NASA /HQ

Herwig Laue
Co-Chair
ESA/ESOC

PRECEDING PAGE BLANK NOT FILMED

TABLE OF CONTENTS

Data Management

Data Management

** Presented in Poster Session*

PRECEDING PAGE BLANK NOT FILMED

PAGE X INTENTIONALLY BLANK

Data Management

** Presented in Poster Session*

Mission Management

** Presented in Poster Session*

Mission Management

** Presented in Poster Session*

ORIGINAL PAGE IS OF POOR QUALITY

Mission Management

Operations

* *Presented in Poster Session*

Operations

** Presented in Poster Session*

Operations

Operations

** Presented in Poster Session*

** Presented in Poster Session*

Operations

Systems Development

Systems Development

** Presented in Poster Session*

Systems Development

** Presented in Poster Session*

Systems Development

Systems Development

** Presented in Poster Session*

Systems Engineering

1. Re-engineering — Page 1121

Systems Engineering

2. Reusable Systems — Page 1159

** Presented in Poster Session*

Systems Engineering

Systems Engineering

* *Presented in Poster Session*

** Presented in Poster Session*

* *Presented in Poster Session*

Operations

4. Expert Systems — Page 679

* *Presented in Poster Session*

Part 2.

PAGE 678 INTENTIONALLY BLANK
PRECEDING PAGE BLANK NOT FILMED

N95-17532

11117

353961

Delivering Spacecraft Control Centers with embedded Knowledge-Based Systems: the Methodology issue

S. Ayache , M. Haziza , D.Cayrac

Matra Marconi Space, 31, rue des Cosmonautes, 31077 Toulouse Cedex – FRANCE.
Tel: (33) 62 24 77 60, Fax: (33) 62 24 77 80
e-mail : ayache@soleil.matra-espace.fr

Abstract

Matra Marconi Space (MMS) occupies a leading place in Europe in the domain of satellite and space data processing systems. The maturity of the Knowledge-Based Systems (KBS) technology, the theoretical and practical experience acquired in the development of prototype, pre-operational and operational applications, make it possible today to consider the wide operational deployment of KBS's in space applications. In this perspective, MMS has to prepare the introduction of the new methods and support tools that will form the basis of the development of such systems. This paper introduces elements of the MMS methodology initiatives in the domain and the main rationale that motivated the approach. These initiatives develop along two main axes: knowledge engineering methods & tools, and a hybrid method approach for coexisting knowledge-based and conventional developments.

I. Introduction

Matra Marconi Space (MMS) occupies a leading place in Europe in the domain of satellite and space data processing systems. It has a long experience, as architect of both types of systems, in the integration of hardware and software components, man-machine interfaces, knowledge and data management systems, etc.

The development of methods and supporting environments is a part of MMS missions. MMS has a confirmed expertise in the domain of system engineering methods and tools. For instance, MMS has co-authored the HOOD design method (dedicated to the architectural and detailed design of large real-time and embedded Software applications) and is involved in the working group in charge of proposing evolutions of the method.

MMS has also acquired a theoretical and practical experience in the development of Knowledge-Based Systems (KBS) through numerous R&D, pre-operational and operational projects generally sponsored by CNES (the French space agency), ESA or other customers such as ARIANESPACE. The development activities conducted at MMS in the eighties have allowed to demonstrate the benefits of KBS to assist users in operation environments. That experience has also led to a robust in-house KBS development methodology.

It is now possible to consider the wide operational deployment of KBS's in space applications. In this perspective, MMS has to prepare the introduction of new methods and support tools that will form the basis of such systems development as well as their cooperation with more conventional methods [10]. After a brief description of the MMS approach in the field of space diagnostic support systems development, this paper develops the methodology issue that MMS is currently tackling and presents an experimentation of a hybrid method approach in the diagnostic systems field.

II. Space diagnostic support systems: the DIAMS programme

MMS has been investigating and experimenting spacecraft diagnostic support systems for eight years. The DIAMS concept, initiated in 1985, led to the development of a prototype expert system dedicated to the Telecom 1 Attitude and Orbit Control System [7] DIAMS-1, and to the present Telecom 2 Expert System [8], DIAMS-2, covering a whole satellite (platform and interfaces with the payload), which was installed in the Satellite Control Center at the beginning of 1993 [3].

One of the main advances realized through DIAMS-1 was the decomposition of the Knowledge base (KB) into different types of Knowledge Islands (KI) representing different domains of expertise.
DIAMS-1was implemented in Emicat (an object dialect on top of Prolog).

The next generation called DIAMS-2 was a near operational system developed on top of a KEE/ CommonLISP platform. It is a hybrid system

PRECEDING PAGE BLANK NOT FILMED

combining decision-tree based symptom-hypothesis associational reasoning to initiate and to focus the diagnosis, and the DIAMS-1 model-based techniques to complete the diagnostic reasoning on particular functions and to provide the final isolation of the fault.

In DIAMS-2,comprehensiveness and efficiency was priviledged against fineness of representation and reasoning. Simplified representations well suited to the practical problems faced in space industry were introduced as a first approximation. A progressive refinement of the models and of the reasoning paradigms selected (for instance to include the handling of incompleteness, uncertainty and time) is now being considered in the definition of a new generation of knowledge based systems, DIAMS-3 [4],[5].

DIAMS-3 is being implemented in C++ and uses the ONTOS Object Oriented Database Management System for knowledge storage and retrieval. Beyond the porting into C++ of the DIAMS-2 machinery, DIAMS-3 will provide generic model edition services and C++ libraries of operational standard for handling time, incompleteness and uncertainty. These libraries could also be reused in other KBS development projects.

Other important objectives of DIAMS-3 concern tigher integration with other knowledge-based systems like data analysis or procedure management tools and more generally the complete integration of that kind of tools in the operational loop [11].

III. Methodology issues

Spacecraft Control Centers (SCC's) have to process large amounts of data from which the relevant information is generally difficult to extract and may require the use of KBS for instance for data analysis and diagnosis (such as those belonging to the DIAMS family). Knowledge-based planning and scheduling or procedures management tools can also be useful to master the management and execution of complex operational tasks. These different categories of KBS's generally need to communicate with the operational environment, i.e to exchange information with conventional software or databases. In addition, the embedding of the various kind of software components (including KBS's) into hardware and at a higher level into a system with its organizational logic has to be taken into consideration.

An example of typical Satellite Control Center functional architecture is provided in table1

Table 1. Typical SCC functional architecture

Core system and Common services	Databases, data storage and retrieval Time synchronization and distribution Local Area Network(s), communications Distributed environment monitoring and control Operation documentation management.
Procedural applications	Data reconstruction and distribution Flight dynamics monitoring and control Operation procedures construction and execution...
Knowledge-based applications	Data analysis Diagnosis Planning/Scheduling...

Various methods, tools, languages, models, or architectures are used to develop these different kindsof components. To give an example, in many SCC's development projects currently conducted at MMS, SADT and HOOD are used for the analysis and design of conventional software, and the MERISE Information System Design methodology (including Entity-Relationship diagrams) is used for the database components. The operational integration of KBS's in SCC's thus raises two kinds of methodology requirements:

- *Knowledge engineering methods & tools:*
  Well-suited methods and tools are required for expertise analysis and knowledge modelling, knowledge verification & validation, or KB Administration and Maintenance.
- *Cooperation with conventional SW development approaches*
  The elaboration of a methodology framework for the cooperation between knowledge engineering and SW engineering methods and tools is an essential requirement to guarantee the safe and efficient cooperation between KBS's and conventional applications within a same operational environment.

Rather than expecting the advent of the ultimate methodology that would allow to develop all types of system components within the same integrated methodology, a pragmatic solution, experimented by MMS, consists in adopting a hybrid method approach. In such an approach, the task of building the integrated application is carried out by developing all the system components within a methodology framework that allows the use of the most suitable existing methods in the successive phases of the development.

This approach of course requires to define correspondences between models for cross validation

purposes but it carries a number of very interesting properties. For instance, it allows to benefit from the experience gained with the existing methods, allows to use existing tools supporting the methods, avoid problems such as compatibility with existing models (SCC's HOOD models for instance) or the costly training of a large number of people to a new method.

A hybrid method approach for KBS development grounded on KADS, HOOD and OMT has been successfully experimented by MMS through the development of the new generation of diagnostic support systems (DIAMS-3). This approach is detailed in the next section.

IV. The hybrid method approach experimented in DIAMS-3

1. Selected methods

Knowledge Engineering methods

The CommonKads method [14] which is now a knowledge engineering method rather popular in Europe supported by off-the-shelves tools has been selected as the DIAMS-3 Knowledge Engineering method. Its founding principle is Knowledge Level Modelling. The purpose of the knowledge-level model is to make the organization of knowledge in the system explicit independently of any representational issue (symbolic representation in terms of rules, frames, etc.) and, a fortiori, of any implementation level issue. **The CommonKads model set** is briefly presented in table 2:

Table 2. The CommonKads model set

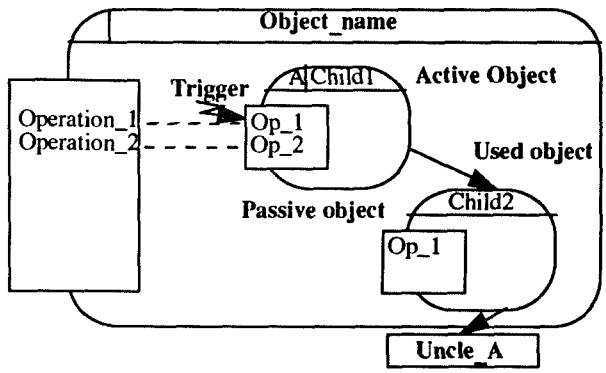
Organizational model	provides an analysis of the organizational environment in which the KBS will run
Task model	Descibes the real-life tasks executed in the organizational environment
Agent model	Describes the properties of agents that perform tasks specified in the task model
Communication model	Describes all transactions between agents
Expertise model	Organizes problem-solving knowledge in four layers: domain, inference, task and strategic knowledge

Software Engineering methods

Having assessed that the association of KADS with object-oriented analysis and design approaches could provide a suitable basis for developments of systems such as DIAMS-3, two complementary methods HOOD and OMT were selected:

- HOOD [12] is a design and development method for large technical and real time software systems. It resulted from the merging of Booch's Object oriented design approach and Abstract Machines methods. The definition of the method was sponsored by ESA and started in 86. Since its birth in 1986, HOOD has become the most commonly used design method in the european space industry. It is now the reference design method for the SW projects sponsored by the European Space Agency. HOOD is a hierarchical design method offering two kinds of interesting relations between objects: the "use" relation to express that one object requires the services of other objects and the "include" relation to express that one object, the parent, is fully implemented by the child objects it contains (cf Figure 1.)

Figure 1. HOOD object: graphical representation



- OMT (Object Modelling Technique) is an object-oriented software development method which extends from problem formulation and requirements analysis, to design and implementation. It has been defined by James Rumbaugh & al. [13] from the General Electric Research center (USA). This method proposes three kinds of models to describe the different views of a system (cf Table3)

Table 3. The OMT model set

Object model	Static, structural view of the system showing objects structure and relationships between them
Dynamic model	Temporal, behavioral view of the system
Functional model	Transformational, functional view of the system

The evaluation work has been focused on the object modelling technique from which the methods draws its name.

2. DIAMS-3 Specification

Two main kinds of output have been provided at the end of this phase:

- Software requirements (following a template close to the Software Requirements Document template recommended in the ESA PSS-05 standard [6]) including both functional and non functional requirements for the overall diagnostic tool.
- A CommonKADS Expertise model for the cognitive parts built with the support of the KadsTool tool. This model is briefly described in the next paragraphs:

Strategic knowledge

The KB is partitioned into knowledge islands (KI's). A KI contains all the knowledge items needed to investigate (i.e. confirm or infirm) some global hypotheses. A strategic-level Investigation Procedure is used to select a path among pending hypotheses and to navigate from KI to KI.

Domain knowledge

Domain knowledge is generally represented by hierarchies of concepts and relations between concepts. A *domain ontology* describes the terms that will be used to formulate statements about the application domain. Domain knowledge may further be specified with the help of some meta-descriptions - *model ontology* - that specify the type and structure of the domain models.

The diagnostic tool model ontology has been mainly represented by two "consist-of" hierarchies structuring:

- the satellite FDIR (Fault Detection and Isolation Recovery) static knowledge and
- the diagnostic session dynamic knowledge introduced as an example in Figure 2.

A complete description of domain knowledge may be found in [1].

Inference & task knowledge

The *inference knowledge* specifies the basic inferences that can be made with the domain knowledge.

The task knowledge describes the problem-solving tasks. Tasks are specified through a task definition and a task body. The task body decomposes the task recursively in terms of activities (other tasks) needed to achieve the task goal. A task description is generally associated to an inference structure and expresses a control flow on the inference structure.

The top-most inference structure and task description of the diagnostic tool Expertise Model are represented in Figure 3. and Figure 4.

3. DIAMS-3 Preliminary Design

HOOD and OMT have been used in a complementary way for preliminary design in the sense that:

- HOOD has mainly been used for the top down decomposition of the application into abstract machines and for an easy representation of interactions of the diagnostic system with external resources such as reasoning schemes. It supported the preliminary design of the diagnostic system shell.

Figure 2. Diagnostic session knowledge "consist_of" hierarchy

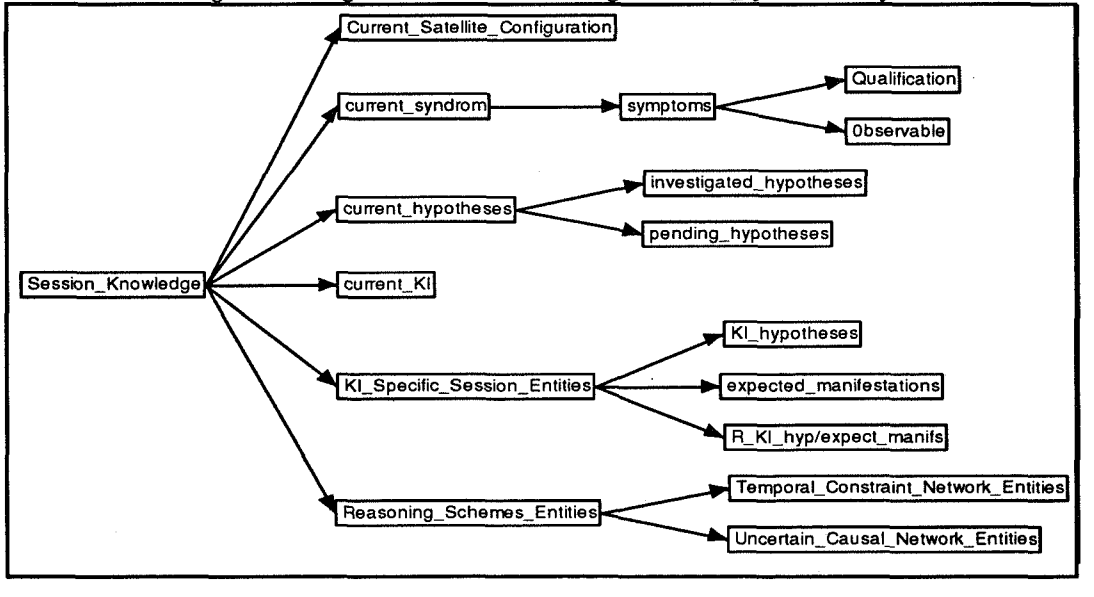


Figure 3. Diagnose Inference structure

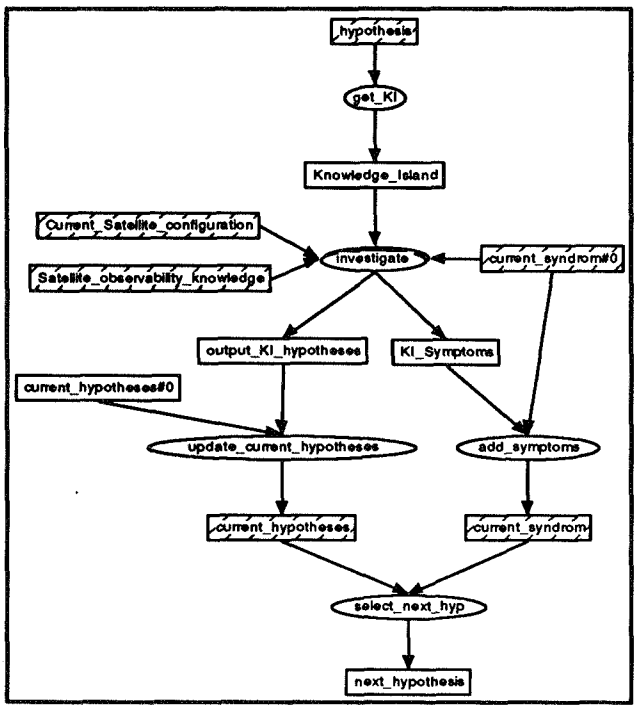


Figure 4. Diagnose Task description

Roles
Input
hi : Initial_hypothesis
Si : Initial Syndrom
Conf: Current_Satellite_Configuration
Obs : Satellite_observability_knowledge
Output
C : diagnostic conclusions
Sf : final syndrom
Control roles
KI : Knowledge_Island – in current investigation
H : Current Hypotheses
S : Current Syndrom
KI_hyp : output_KI_hypotheses – deduced from KI Investigation
KI_sym : KI_Symptoms – observed during KI_investigation
h : next_hypothesis to be investigated
Body
DIAGNOSE(hi,Si,Conf,Obs –> (C,Sf))=
S = Si, h = hi, H = { }
WHILE "select_next_hyp" returns an hypothesis
get_KI(h –> KI) – returns the KI associated to hypothesis h
INVESTIGATE(KI,S,Conf,Obs –> (KI_hyp,KI_sym))
Update_current_hypotheses(H,KI_hyp –> H) – add KI_hyp in H and update hypotheses plausibilities
Add_symptoms(S,KI_sym –> S)
Select_next_hyp (H,S –> h) – select next pending hypothesis h according to diagnostic focusing rules and set h status to "not pending";
END WHILE
C = H, Sf = S
Realization INVESTIGATE
Activates inference diagnose

- The OMT design process is not hierarchical but OMT offers a very powerful object modelling technique including of course modelling of inheritance. OMT has mainly been used to design the domain objects classes and relationships between these classes.

An example of HOOD object graphical description extracted from the documentation generated by the HOODNice tool is provided in Figure 5.

This description shows the decomposition of the object "Diagnoser" which is itself included (with other objects such as "KB_administrator" or "KB_interface") in the decomposition of the top level object called "Diagnostic_System". This figure shows "use" relations between Diagnoser internal objects and external objects (e.g., KB_interface) or objects belonging to the Diagnostic System Software environment (e.g., Temporal Constraint Propagator - TCP- and Valuation Based System -VBS- handling temporal and uncertain reasoning)

An example of OMT sheet extracted from the documentation generated by the OMTool tool is provided in Figure 6. This example shows a preliminary design model for KI_hypothesis and Knowledge_Island domain objects.

4. DIAMS-3 detailed design

Only OMT has been used to support the detailed design activity. This allowed a direct mapping to C++ object classes. OMT has also been used to maintain an up-to-date view of the detailed design model during the coding activity.

Classes identified in OMT preliminary design appear as ONTOS persistent classes in the detailed design model and methods corresponding either to administration methods or to basic inference mechanisms have been attached to these classes. An example of such a persistent class is provided in Figure 7.

Objects identified in the Diagnostic system shell HOOD preliminary design model appear as non-persistent classes in the detailed design model . An example of such a class is provided in Figure 8. In this case, services provided by the "Hypotheses manager" in preliminary design are dispatched in two classes: a semantic class used in the diagnostic process and a graphical class used to manage the Man-System dialog (its content has been masked to simplify the figure).

Figure 5. Diagnoser Hood Object graphical description

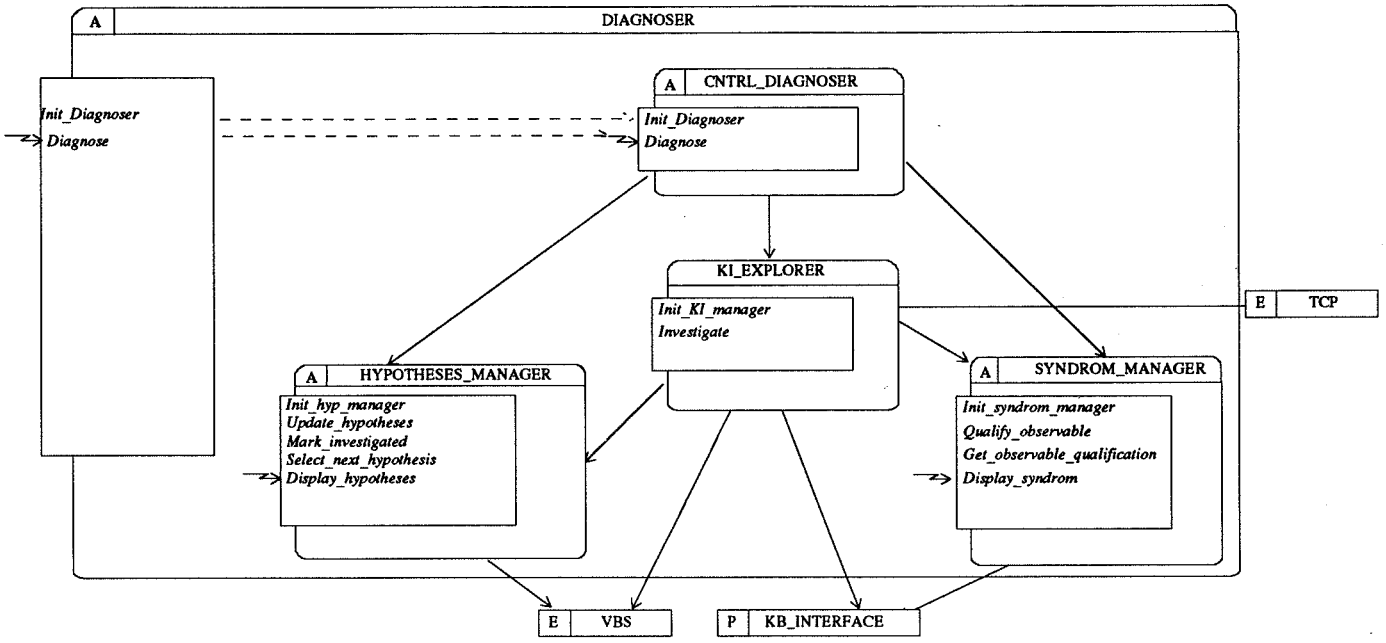


Figure 6. OMT sheet including KI_hypothesis subclasses and KI_hypothesis-KI relationships

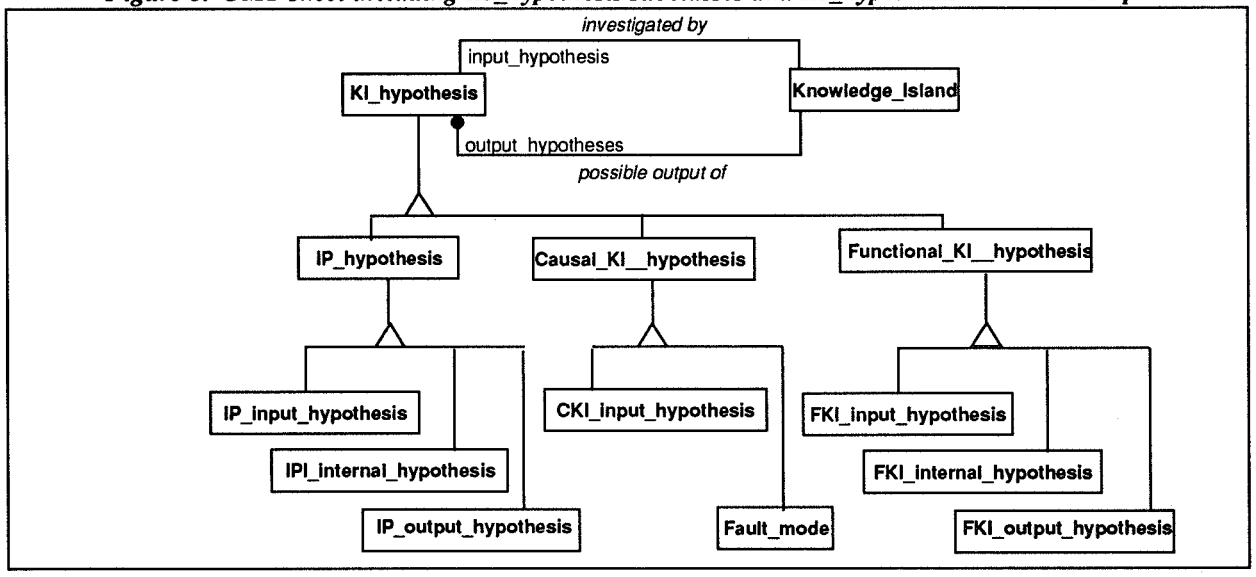


Figure 7. The KI_hypothesis class

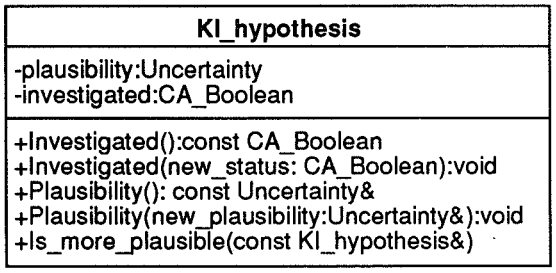
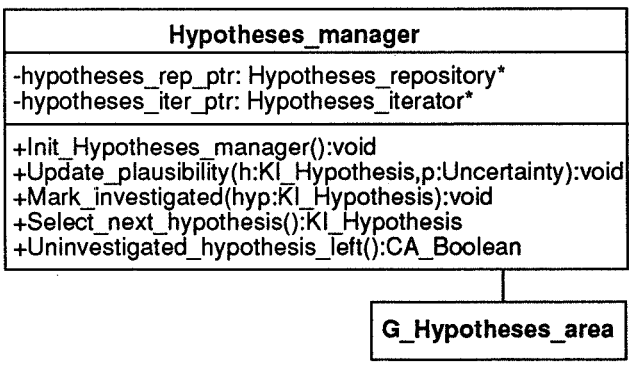


Figure 8. The Hypotheses_manager semantic and graphical classes



5. Experience Feedback

Each of the selected methods carries advantages and drawbacks. Taken as a whole, the set of selected methods exhibits complementary features allowing to progress in the elaboration of guidelines for selecting a lifecycle model and a combination of methods well-suited to a particular application project. This is further detailed hereafter.

Methods evaluation summary

CommonKADS

The CommonKADS modelling approach is mainly focused on the analysis phase and cannot be considered as a comprehensive methodology that provides guidance and support in all phases of operational KBS development projects.The application development experience showed that people with a practical experience in SW engineering got acquainted rather rapidly with the KADS approach.

The use of KADS allowed to establish a common universe of discourse over the project. KADS models were found very useful by the newcomers and eased their integration in the project team.

HOOD

The use of the HOOD method allowed the top-down decomposition of the application into modules. This provided a convenient basis for the specification of the man-system interfaces and the modelling of interactions with external resources (other KBS's, database systems or procedural applications). The HOOD modelling approach has been designed to facilitate the structuration of large projects. In the early phase of the application development, its use indeed simplified the task sharing between team members

However the main drawback of the method resides in its lack of support for the modelling of inheritance, which is a critical requirement when developing KBS, and, correlatively, the absence of C++ code generator in the tools that support the method. This feature prevented the selection of HOOD as the application detailed design method.

OMT

OMT offers a powerful object modelling technique which turned out to be well adapted for the preliminary design of classes corresponding to domain objects and for the detailed design of the whole application. In addition the support tool used allowed to generate C++ code skeletons based on the OMT object model components.

Among the methods investigated, OMT is probably the one which is the closest to the ideal comprehensive methodology that could be applied to all kind of system components - KBS's, conventional applications, database applications, etc. - in all phases of integrated systems lifecycle. Notice for example that MMS is using OMT for two KBS projects: "Architectural concept for Spacecraft Operations Automation" (sponsored by ESA/ESOC) which aims at integrationg various KBS(procedures management, data analysis, planning/scheduling) within the current ESOC control center (SCOS) and "Ogre", a KBS for ARIANE5 tests data analysis and reports generation (sponsored by CNES). However the method is still rather young - support tools of industrial standard are only emerging - and not widely used for operational system developments in space. Notice also that in Europe, ADA remains the reference language for real-time systems developments and that HOOD will probably remain the reference method for such developments for a few years still.

V. A hybrid methodology framework for co-existing conventional/knowledge-based developments

The method cooperation approach straightforwardly derives from the operational continuity principle. This requirement states that as organizations are hard to change, and as old applications and organizations have to be maintained while introducing new system capabilities, it is important that applications be developed on a modular basis to enable an incremental development and maintenance strategy.

This principle at the application level translates into a dual principle at the methodological level that could express as follows: when people have a good working knowledge of a given method that has proved to be well-suited to a given class of system components it is preferable to let them use the known methods and to limit the enforcement of new methods to system components and development phases which are not well covered with the existing methods.

Rather than developing a comprehensive methodology, the proposed approach is thus to define a framework that supports the cooperation between methods.

Table 4 introduces a first instance of such an hybrid approach that synthesizes the main results of the method evaluation work as well as other results coming from a comparison of KADS, MERISE, SADT and OMT methods [9]. This table associates a set of methods or languages to each lifecycle phase. Such sets of methods can be interpreted either as

alternatives methods (e.g., KADS/ OMT for domain objects modelling) or complementary methods (e.g., HOOD/OMT for preliminary design) or as possible mappings between models for cross_validation purposes (e.g., KADS/MERISE where KADS is used for Knowledge_based components and MERISE for SCC operational databases).

The method cooperation approach also requires to manage the correspondence between different representations of the same objects at each step of the development process. This is particularly needed for objects encapsulating knowledge & data exchange services between different subsystems and to perform the cross-validation of models. This question has also been investigated in [9]

Table 4. Method components for operational integration of KBS's in space environments

Lifecycle phase	Data / Domain Objects	Functions / Tasks / Inferences
Analysis	KADS, MERISE, OMT	KADS, SADT, OMT
Preliminary design	OMT	HOOD, OMT
Detailed Design	OMT	OMT
Coding	C++	C++

Conclusion

In this paper, we have presented a hybrid methodology framework that could contribute to the operational integration of KBS's in SCC's as this has been demonstrated on the example of diagnostic support systems.

Experience feedback coming from MMS current KBS projects using OMT for the whole lifecycle will also provide valuable inputs for assessing this hybrid methodology framework.

Further goals for MMS in this area are to refine the proposed hybrid approach through elaboration of rules for the maintenance and updating of hybrid models in the coding phase (including the management of traceability links). The situation of prototyping and V&V activities wrt. the proposed hybrid approach are also being investigated.

Acknowledgments

Work related to some of the topics developed in this paper was partially funded by CEC through UNITE project (ESPRIT project 6083). Other partners involved are: Cap Gemini Innovation(France), Queen Mary Westfield College(UK), Sintef Delab(Norway), Eritel (Spain), ITMI (France).

References

[1] S.Ayache, P.Caloud, D.Cayrac, M.Haziza "MMS Component Application Specification and Design"- UNITE deliverable, November 1993

[2] S.Ayache, P. Caloud, D.Cayrac, M. Haziza "MMS Integrated Application Specification and Design"- UNITE deliverable, November 1993

[3] J.M.Brenot, P.Caloud., L.Valluy, A.Gasquet: "On the design and development choices to bring to operation a diagnostic expert system for the Telecom 2 satellite", Proc. Tooldiag Int. Conf. on Fault Diagnosis, Toulouse, France, 1993.

[4] D. Cayrac, M. Haziza: "Management of Uncertainty and Temporal Dependencies in Real World Diagnostic Systems, Application to the Space Domain", Proc. Tooldiag Int. Conf. on Fault Diagnosis, Toulouse, France, 1993.

[5] D. Cayrac, D.Dubois, M.Haziza, H. Prade.: "Relational Diagnosis Based on a Functional Model", Proc. AIENG conf., Toulouse, France, 1993.

[6] ESA PSS-O5-0 standard Issue 2 (February 1991)

[7] M.Haziza "An Expert System Shell for Satellite Fault Isolation based on Structure and Behavior", in Proc. ESTEC Workshop on Artificial Intelligence and Knowledge Based Systems for Space, Noodwijk, Netherlands, 1988.

[8] M.Haziza "Towards an Operational Fault Isolation Expert System for French Telecommunication Satellite Telecom 2": Proc. ESA Symposium "Ground data systems for spacecraft control", Darmstadt, FRG, 1990.

[9] M.Haziza, S.Ayache, D.Cayrac, P.Y.Lambolez: "MMS first report on the use of selected methods for analysis and design" - UNITE deliverable, November 1993

[10] M.Haziza: "Delivering operational space applications with embedded Knowledge_based systems", World congress on Expert Systems, Lisbon, Portugal, 1994.

[11] M.Haziza, S.Ayache, J.M. Brenot, D.Cayrac, D.-P Vo: "Diams Revisited: Taming the Variety of Knowledge in Fault Diagnosis Expert Systems", to appear in proceedings of SPACEOPS'94 (International Symposium on Space Mission Operations and Ground Data Systems), USA, November 1994.

[12] B. Labreuille, JF. Muller, HOOD: a design method for the space industry, AIAA/NASA Symposium on Space Information Systems, 1990

[13] J. Rumbaugh et al: "*Object Oriented Modelling and Design*", Prentice Hall International, 1991.

[14] G. Schreiber et al: "*KADS: a principled approach to Knowledge Based System Development*", Academic Press, 1993

111118

N95-17533

353975 P-8

SCOSII OL: A Dedicated Language for Mission Operations

Andrea Baldi, ESA/ESOC/FCSD
Dennis Elgaard, CRI
Steen Lynenskjold, CRI
Mauro Pecchioli, ESA/ESOC/MOD

Abstract

The Spacecraft Control and Operations System II (SCOSII) is the new generation of Mission Control System (MCS) to be used at ESOC. The system is generic because it offers a collection of standard functions configured through a database upon which a dedicated MCS is established for a given mission.

An integral component of SCOSII is the support of a dedicated Operations Language (OL). The spacecraft operation engineers edit - test - validate and install OL scripts as part of the configuration of the system with e.g. expressions for computing derived parameters and procedures for performing flight operations, all without involvement of software support engineers.

A layered approach has been adopted for the implementation centred around the explicit representation of a data model. The data model is object-oriented defining the structure of the objects in terms of attributes (data) and services (functions) which can be accessed by the OL.

SCOSII supports the creation of a mission model. System elements as e.g. a gyro are explicit, as are the attributes which describe them and the services they provide. The data model driven approach makes it possible to take immediate advantage of this higher-level of abstraction, without requiring expansion of the language.

This article describes the background and context leading to the OL, concepts, language facilities, implementation, status and conclusions found so far.

Introduction

The *need* for the SCOSII OL has matured through the long experiences ESOC have had with the use of configurable generic MCS's. As any other previous ESOC MCS, SCOSII will be configured through databases containing the mission specific knowledge.

This knowledge will not only need to be efficiently defined, but also validated and then maintained, due to the pre-launch test results and/or the frequent changes which do occur during the lifecycle of a mission.

The SCOSII OL concept is designed to augment the traditional ways an operation engineer specifies mission specific configuration data to cover as well knowledge which is *algorithmic* or *procedural* in nature. Thus it is essential to support the operations engineer in:

- specifying and maintaining the mission knowledge in a natural, concise and intelligible manner - without requiring a detailed software understanding or support of software engineers;
- defining the mission knowledge in context-specific dedicated environments, whereby both the HCI and the allocated constructs are specifically designed for each particular information type;
- validating the specified knowledge by means of 'on-line' checks and testing capabilities.

Andrea Baldi (abaldi@esoc.bitnet) works within the Flight Control Systems Department at the European Space Operations Centre (ESOC), Robert Bosch Strasse 5, D-64293 Darmstadt, Germany. Dennis Elgaard (delgaard@esoc.bitnet) and Steen Lynenskjold (steen@acm.org) both work for Computer Resources International, Bregnerødvej 144, DK-3460 Birkerød, Denmark. Mauro Pecchioli (mpecchio@esoc.bitnet) works within the Mission Operations Department at ESOC.

The work described in this article was carried out at ESOC under a contract with the European Space Agency.

Background and Context

For any mission has been the demand to *derive* information from the format which is provided through the spacecraft telemetry parameters. The most frequently used derivation is that of applying a (linear) calibration to convert raw values into engineering units. The calibration is defined by providing value pairs as part of the database configuration.

Although calibrations satisfy a large percentage of the derivation needs, they do not provide a sufficient mechanism as there is as well a need to compute *derived* values by combining other values using an algorithmic transformation.

In the Multi-Spacecraft Support System (MSSS) these algorithms were specified on paper by an operations engineer and subsequently coded by a software engineer. In SCOSI the operations engineer writes the algorithm directly in FORTRAN expanded with a few syntactical constructs to e.g. reference a previous value of a parameter. In both cases the resulting FORTRAN code is compiled and linked with the operational control system software. An error in the algorithm will not be detected before a run-time crash occurs. The turnaround time for changes has from an operational perspective a significant and unwanted delay. Neither systems support version and configuration control functions.

The Spacecraft Performance and Evaluation System (SPES) offers a significant improvement as it allows the users through a dedicated language to define expressions, compute averages, etc. SPES is however limited to work in an off-line context on historical values and has no integration with the control system as such.

The possible largest driver for the requirements is the wish to formalise and incorporate executable operation procedures written in the OL within SCOSII. Whereas algorithms for derived values do not necessary have to be explicit in the run-time context, procedures do have to: one property of a procedure is its interactive nature involving a close dialogue with a human operator through a procedure execution display.

Within ESA, check-out systems have for some time provided capabilities of defining test procedures through special languages; the most significant ones being ETOL (ESA Test Operations Language), ref. [10], and ELISA (Extended Language for Instrument and Spacecraft AIV), ref. [9]. These check-out languages focus on regression testing capabilities.

Two ESOC studies have demonstrated the feasibility of executable procedures within control systems, namely the Expert Operator's Associate (EOA) study, ref. [12], and the Meteosat WorkStation (MWS) study, ref. [13] - the latter now being used operationally. Both projects focused on the internal representation of procedures and the interactive nature of their execution with close coupling to the human spacecraft operator.

The User Terminal Study at ESTEC, ref. [8], has shown the advantages of an object-oriented language in combination with a mission model. The User Language Study at ESOC, ref. [7], was initiated with the purpose of providing inputs to the SCOSII OL and has proven a number of concepts; in particular the advantages of a layered implementation centred around the explicit representation of a data model. Both studies focused on the configurability aspects of the system and associated language capabilities.

From a technological view the existence of powerful UNIX utilities such as lex and yacc, the ideas behind database languages as SQL, advances in workstation performance, and the maturity of object-oriented concepts have further made it possible to implement the OL.

SCOSII, ref. [1][2][3][4][5][6][14], is the new generation of generic control systems to be taken into use at ESOC; the first client missions being Huygens (97), Artemis (97) and

Envisat (98). SCOSII is a distributed control system running on powerful UNIX workstations connected through a local area network. SCOSII has been engineered for high performance throughput; in particular to optimise the parallel access to real-time and historical data. Further emphasis is put on the configurability of the system to incorporate a mission model, hereby offering a higher level of abstraction than that traditionally provided by telemetry parameters and telecommands. A new Human-Computer Interaction (HCI) concept has been adopted based on closer data integration and referential capabilities.

Concepts

SCOSII is a generic system which is configured by adding *missing specific knowledge*, which may be categorised into:

- *declarative knowledge*, e.g. calibration curves, parameter structures, etc.; specified through dedicated form based HCIs;
- *expressive knowledge*, e.g. derived parameters, command validation conditions, etc.; specified through the OL;
- *procedural knowledge*, e.g. operation procedures, report procedures, etc.; specified through the OL;
- *special knowledge*, i.e. non-generic mission information typically requiring a software expansion to SCOSII.

It is difficult to define the *borderline* of when to use *declarative* or *expressive* knowledge, i.e. when to use the OL. The definition of specific items within the database have typically *both* a declarative and an expressive part. The identifier, description, etc. of a Parameter is defined by declarative knowledge, whereas its validity criteria is defined by expressive knowledge. Due to this 'mixture' of declarative and expressive knowledge inherent to most database parts, the way the user interacts with the system needs to reflect this fact. Neither a pure (traditional) forms interface nor a pure OL definition environment would suffice, both need to be accessible in a homogeneous manner from within the same HCI.

An operations language needs to interact with the control system to be able to *access data* held by the control system which is of operational importance to get e.g. the validity status of a telemetry parameter; *request services* to e.g. send a telecommand; and *change data* to e.g. store the results on an evaluation of a derived parameter.

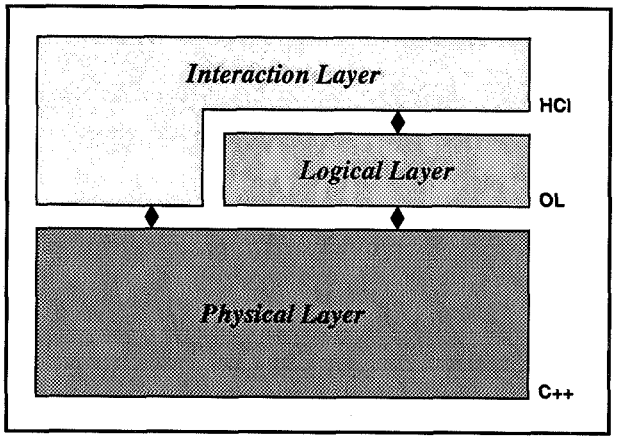


Figure-1 Layered Model

A *layered approach* has been adopted for the SCOSII OL as shown in Figure-1. The three layers are:

- *Interaction layer*, i.e. the user interface of the system which may interact with the physical layer directly or with the logical layer;
- *Logical layer*, centred around the OL containing the data entities which are manipulated via constructs in the language;
- *Physical layer*, providing the generic services of the control system.

The access from the logical to the physical layer is dictated by an explicit *data model*. The data model is *object-oriented* as it represents physical layer objects with attributes and services accessible to the OL.

It supports the explicit representation of inheritance, aggregation and association relations. This enables the OL to facilitate navigation through related objects, e.g. from a command to the parameter used within its post-execution verification checks.

The data model serves as a 'contract' between the logical and physical layers, it can not be changed through the OL itself. This does not imply that the data model is *static*, changes are just controlled through a mechanism within the physical layer. Any change to the data model is propagated to the logical layer.

The physical layer within SCOSII is itself based on an object-oriented implementation, i.e. the differences in representation between the logical and physical layers are less than would otherwise have been the case. The direct implication of this is that the logical layer is 'slim': it mainly serves to present physical layer objects to the operations engineer while hiding implementation details and offering protection against illegal access. The intelligent behaviour always rests within objects of the physical layer, i.e. if the physical layer does not support a certain function it will neither be available within the OL.

SCOSII supports the representation of a *mission model*, allowing to organise the mission knowledge according to a structural representation of system elements, e.g. a gyro or a heater. The OL can access these higher level objects in the same way as any other object within the physical layer, i.e. it does not require a language expansion to take advantage of these.

It is transparent to the OL whether it accesses *static* (database configuration data, e.g. parameter characteristics) or *dynamic* (processing data, e.g. latest parameter value) data. Although the OL does offer facilities to explicitly request *historical data*; the concept of *time* is nominally managed through the application using the OL. A parameter display may be put into *retrieval mode*, the validity of each parameter is calculated on the basis of *current values* of any contributing parameters.

It is further transparent to the OL that SCOSII is a distributed system. All aspects dealing with data distribution and synchronisation are handled fully by the physical layer.

The OL is an *interpreted language*. The reasons for this choice have mainly been that at least operation procedures are interactive of nature involving communication with a human operator - for which an interpretation was believed most adequate.

All OL definitions form part of the database configuration of a SCOSII system. They are therefore underlying strict version and configuration control.

Language Facilities

The OL is a *strongly typed* language, which enables the detection of a range of errors at preparation time during database configuration rather than causing an error at execution time. The data model forms part of the type system within the OL; accessing the physical layer objects in a wrong way will be detected prior to its execution.

The executable unit within the OL environment is an *OL Script*. A script may be as simple as a single boolean expression or as complex as the full directives of a large flight operations procedure. A script is composed of two parts: a *declaration part* (local variables and function definitions) and an *executable part* (statement list).

The access to the physical layer objects is governed through the explicit existence of an object-oriented data model. Figure-2 illustrates a segment of a script to calculate the value of the derived Parameter P117. If the status of the limit of Parameter P112 is above limits, then the engineering Value of P117 is set to the upper limit definition of P112; otherwise it is set to be the engineering Value of P112.

```
{
if (P112.limit == ABOVE_LIMITS) then
    P117 := P112.limit.upper;
else
    P117 := P112;
endif;
}
```

Figure-2 Operations Language Example

Figure-3 shows the data model corresponding to this example. A *Parameter* is characterised by its *name*, *description*, *limit*, *raw* and *engineering* Values. Each Class may have a *default attribute* (marked with a '*'): for the Parameter the default is its engineering Value. A Parameter offers a service *delta* which allows to access historical samples. A *Value* is characterised by its *value* (default) and *validity*. A *Limit* is characterised by its *status* (default), *lower* and *upper* limit definitions. Notice that due to the concept of default attributes, the expression 'P112' evaluates as 'P112.eng.value'.

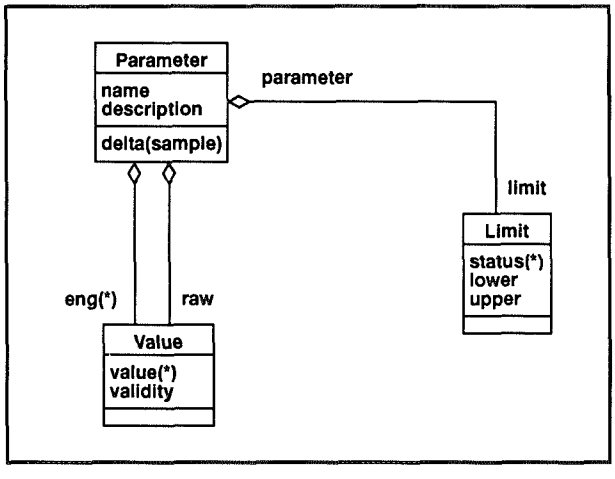


Figure-3 Logical Layer Data Model

Figure-4 shows the representation of a *Heater* system element within the logical layer. A Heater is characterised by its *switch-status* (on-off) and *power-status* (on-off) attributes, and offered service to *switch* it either on or off. The OL can operate on heaters in the same manner as on parameters shown earlier.

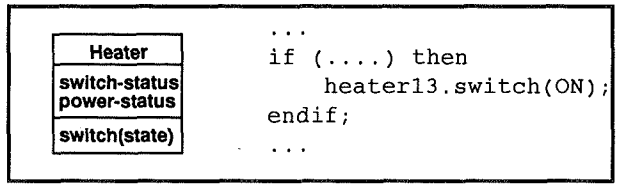


Figure-4 System Element Logical Layer Data Model and OL Example

The OL is, besides from its integration with the data model, a straight-forward imperative language. Table-1 provides an overview of the major language constructs.

Table-1 Operations Language Constructs

Statements	*Expressions*	*Functions*
assignment	value	mathematical
wait	reference	statistical
function invocation	function invocation	bit manipulation
goto-label	boolean expression	time
if-then-else	numeric expression	object creation
select-case	string expression	object copy
while-do	time expression	
repeat-until	list expression	
for-in-list -do	set expression	
for-to-step-next	matrix expression	
	vector expression	
	map expression	

The generalised approach of interfacing physical layer objects governed by the data model is not in all cases adequate. A trade-off has to be made whether to provide a more targeted syntax for particular kinds of knowledge. It is expected that specialised 'mini languages' extending the OL syntax will evolve - typically also offering dedicated HCI support. However, the baseline is that these shall be *mapped* onto the kernel OL at the syntactical level, i.e. in terms of macro expansion. This ensures that the intelligent behaviour stays within the physical layer of the MCS.

Implementation

The OL facility is implemented as any other SCOSII software: it is specified and designed using an object-oriented method (OMT, ref. [11]), and programmed in C++. The UNIX utilities lex (scanner generator) and yacc (parser generator) are used to construct the parse tree.

Due to the fact that the OL scripts form part of the database configuration and hence are defined in the preparation phase, the parse tree is built already at this stage to improve the performance in the execution phase. The parse tree structure is used directly by the interpreter.

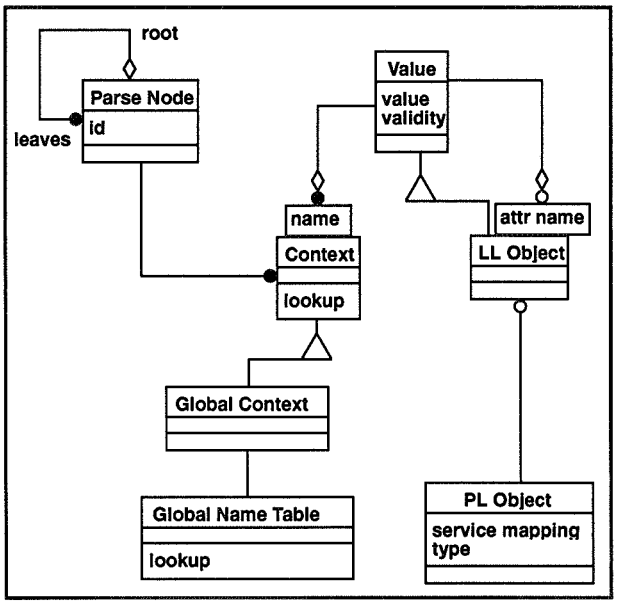


Figure-5 Physical Layer Interface Class Diagram

The *physical layer interface* is illustrated in Figure-5. A *Parse Node* is a component of the parse tree and is characterised by an *identifier*. It references its *root* Parse Node and all of its *sub* Parse Nodes. A Parse Node is evaluated within a particular Context. A *Context* maps identifiers onto Values and offers a *lookup* service. The *Global Context* is a special kind of Context which interfaces a Global Name Table provided by the Physical Layer (PL). The *Global Name Table* offers a *lookup* service taking as input a character string (e.g. "P112") and returning a reference to the corresponding PL Object.

A *Value* is characterised by its *value* and *validity status*, which is used to propagate the effects of non-valid values throughout the evaluation of expressions: if a Value is computed on behalf of non-valid Values, it is itself to be considered non-valid.

A typical example of a non-valid Value is the state of a switched-off (or redundant) unit which still is being sampled and echoed through telemetry.

A *LL Object* is a special kind of Value. It is structured as a record, containing a Value for each of its attributes.

Any object within the PL which needs access from the LL inherits the properties of the PL Object, hereby ensuring the proper interface to the LL. A *PL Object* is characterised by its *type* and contains a *service mapping* relating requests from the LL onto C++ functions of the PL. All LL Objects are attached to one PL Object. At run-time only the PL Objects actually used are related to LL Objects.

An initiative is currently being undertaken to further generalise the physical layer interface by adopting the Model-View-Controller (MVC) architecture, ref. [15], with the purpose of using identical interfaces from both the interaction and the logical layers to the physical layer, see Figure-1. The first prototypes with this architecture have demonstrated promising results.

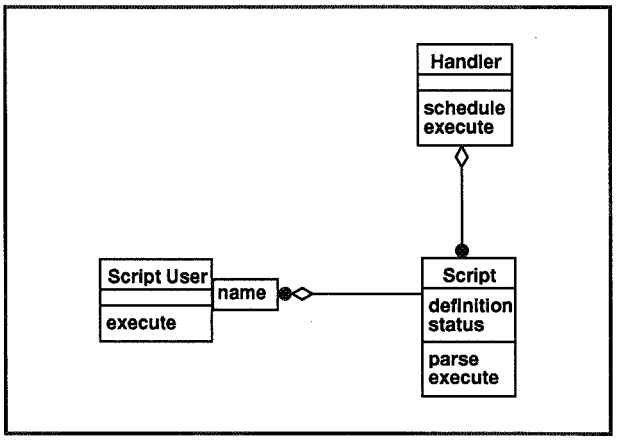


Figure-6 Script Class Model

The Handler, illustrated in Figure-6, controls the execution of any Script. It offers two services: *schedule*, which determines the order in which scripts are executed, and *execute*, which invokes the script execution.

A Script is characterised by its *definition*, i.e. a textual representation of the script, and its *status* - e.g. whether it has been parsed. It offers two services: *parse*, which builds the parse tree of the script, and *execute*, which requests the execution of the script. Any application using scripts have to inherit from the Script User class, which provides the mechanism to interface the OL environment and request the execution of scripts.

The initiative to execute scripts nominally comes from an application using the OL. The Handler has to deal with the incoming execution requests. Currently a very simple scheduling mechanism is implemented; it is foreseen to expand this into a finer-grained mechanism taking aspects, like priorities and pre-emptive scheduling, into account.

Nominally a script will be version controlled as part of its using entity: e.g. the validity criteria of a parameter specified as an OL boolean expression is seen as part of the corresponding parameter version. If the validity criteria is changed, then a new version is associated with the whole of the parameter it belongs to. The granularity in terms of at which level of detail to manage versions is decided on a mission specific basis.

No language constructs to deal with parallelism or script execution synchronisation are provided. It is believed that such aspects are better managed by the physical layer. Within the OL *conditions* can be defined as e.g. an interlock (execute upon successful verification) between two operation procedure execution requests. The physical layer knows about the conditions and observes these while servicing the related execution requests.

At this stage only basic OL editors and execution displays are provided. It is expected to expand the tools with a debugger and test tool, enabling the operations engineer to test and validate Scripts locally on a workstation.

Status

SCOSII is under development. A Basic System has recently been delivered comprising functions equivalent to those offered in the existing generic MCS's used at ESOC. A reduced OL facility covers only expressive knowledge and simple tools. Further evolutionary releases are planned:

- *release 1 (1Q95)*, adds e.g. mission modelling capabilities and executable operation procedures. The OL facility covers procedural knowledge and simple tools.
- *release 2 (1Q96)*, adds e.g. advanced mission modelling and semi-automatic operation procedure execution. The OL facility is complete with tools.
- *release 3 (1Q97)*, adds e.g. integration with knowledge based applications for automatic operation procedures execution.

FOPGEN, a WYSIWYG tool to support editing, display and printout of operational documentation, will be fully integrated with SCOSII. It provides advanced editing features and read/write access to the SCOSII mission database. FOPGEN will generate operation procedures in the SCOSII OL.

In parallel with the SCOSII development, two major studies have been initiated: *ATOS-4* exploits the use of knowledge based technology in e.g. the context of procedure execution based on SCOSII and the OL; *Productline for Compact Ground Facilities* investigates the integration of check-out and operation control systems, with particular emphasis on the language aspects.

The Committee for Operations and EGSE Standardisation (COES) is currently active to standardise the ground segment infrastructure systems within ESA. A particular subject covers the standardisation of the human-computer interaction of which a dedicated language is seen as an integral part.

The SCOSII OL will be a significant contributor to this standardisation work; the OL itself will be made compliant to the forthcoming standard.

Conclusions

The SCOSII OL provides support to the operations engineer for the configuration of a MCS with mission specific data to include expressive and procedural knowledge, hereby clarifying the borderline between the mission specific and generic elements of a MCS. The turn-around time for a change is drastically reduced as it does not involve any software modifications.

It does not cover the declarative knowledge for which the existing forms based HCI have proven to be efficient. A mixed approach has hence been adopted where only a subset of the configuration data is specified through the OL.

The existence of an explicit object-oriented data model ensures a clear framework for the interface to the physical layer of SCOSII.

The language is on purpose 'kept simple and stupid', expecting the intelligent behaviour to be provided by the physical layer objects. This facilitates improved performance within the OL environment.

The language is bound to SCOSII. As there is no intelligent behaviour within the logical layer, it depends upon the level of services offered by the physical layer. The direct implication of this is that although the architecture concepts could be adopted, it makes little sense to port the language environment to a different platform than SCOSII.

The data model approach, although flexible, has the possible disadvantage that porting OL scripts between missions can be difficult as each mission could have their own different data model. This is however a property of any generic system, not just the SCOSII OL environment.

With the planned expansions of SCOSII to cover extensive mission modelling capabilities, the added level of abstraction within the physical layer will allow the OL to take immediate advantages of this due to the generalised data model approach, without requiring syntactic nor semantic changes to the language. It is expected that the full advantages of the SCOSII OL will be demonstrated at that stage.

References

[1] SCOSII: ESA's New Generation of Mission Control Systems - The User's Perspective, P Kaufeler, M Pecchioli, I Shurmer, ESOC - *these proceedings.*

[2] A New Communication Protocol Family for a Distributed Spacecraft Control System, A Baldi, M Pace, ESOC - *these proceedings.*

[3] SCOSII: ESA's New Generation of Control Systems, M Jones, N Head, K Keyte, P Howard, S Lynenskjold - *these proceedings.*

[4] SCOSII - An Object Oriented Software Development Approach, M Symonds, S Lynenskjold, C Müller - *these proceedings.*

[5] SCOSII User Requirements Document, ESOC DOPS-SYS-URD-001-AMD, Issue 3, February 1994.

[6] SCOSII Software Requirements Document, ESOC SCOSII-SYS-SRD, Issue 0.6, June 1994.

[7] User Language Study, VEGA ULS.RST.REP.005 (ESOC), Issue 1.0, October 1992.

[8] Object-Oriented User Language for Satellite Check-out, Spacebel Informatique (ESTEC), 1992.

[9] Control File Language for Cluster AIT, CRI CL-CRI-ID-0006 (ESTEC), Issue 3, February 1993.

[10] User Reference Guide for ERS-1 Check-out Activities, CRI ER-MA-CRI-AV-065, Volume 1, Issue 1, April 1987.

[11] Object-Oriented Modelling and Design, Rumbaugh et.al., Prentice-Hall 1991.

[12] Expert Systems for Automated Spacecraft Operations, CRI/Matra EOA-CRI-FINAL-0001-1991 (ESOC), March 1993.

[13] Definition Study for a Meteosat Workstation, VEGA MWS.RST.REP.002 (ESOC), March 1989.

[14] SCOSII - A Distributed Architecture for Ground System Control, Vitrociset Keyte, International Symposium on Spacecraft Ground Control and Flight Dynamics, Brazil, February 1994.

[15] A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System, ParcPlace Systems, August 1988.

11119

N95-17534

353982

P - 10

DIAMS Revisited: Taming the Variety of Knowledge in Fault Diagnosis Expert Systems

M. Haziza , S. Ayache, J.-M. Brenot, D. Cayrac, D.-P. Vo

Matra Marconi Space, 31, rue des Cosmonautes, 31077 Toulouse Cedex – FRANCE.

e-mail : haziza-marc@mms.matra-espace.fr

Abstract

Matra Marconi Space (MMS) has been developing spacecraft diagnostic support systems for eight years. The DIAMS program, initiated in 1986, led to the development of a prototype expert system, DIAMS-1, dedicated to the Telecom 1 Attitude and Orbit Control System, and to a near-operational system, DIAMS-2, covering a whole satellite (the Telecom 2 platform and its interfaces with the payload), which was installed in the Satellite Control Center in 1993. The refinement of the knowledge representation and reasoning is now being studied, focusing on the introduction of appropriate handling of incompleteness, uncertainty and time, and keeping in mind operational constraints. For the latest generation of the tool, DIAMS-3, a new architecture has been proposed, that enables the cooperative exploitation of various models and knowledge representations. On the same baseline, new solutions enabling tighter integration of diagnostic systems in the operational environment and cooperation with other knowledge intensive systems such as data analysis, planning or procedure management tools have been introduced.

I. Introduction

Spacecraft (S/C) operations have pioneered the introduction of the Knowledge-Based Systems (KBS) technology in Space. The prototyping activities conducted in the eighties have allowed to demonstrate the potential of KBS to assist in controlling space systems. Knowledge-Based Systems in S/C Control Centers (SCC) have proven to have a high potential for

- assisting spacecraft engineers in monitoring and analyzing S/C data, and in diagnosing on-board failures from the knowledge of the S/C state obtained through the telemetry.
- assisting S/C engineers in complicated operations where the exact sequence of operations is determined by external constraints and by the actual S/C state at each step.

Spacecraft Assembly, Integration and Test (AIT) is also becoming a knowledge intensive activity that requires appropriate knowledge-based assistance. Due to the increasing complexity of space systems, an increasing number of parameters have to be tested before launch through more and more elaborated test procedures. At the same time, the duration of the AIT phases is continuously decreasing. This makes the AIT phase a critical phase in almost all present space projects and increases the pressure on the development teams.

The use of knowledge based systems for emergency management, fault diagnosis, resource management, replanning/rescheduling, etc. and the operational integration of such facilities in future ground infrastructures (SCC's, AIT environments) should help lowering the risks in problem diagnosis and selection of recovery actions, avoiding mis-diagnosis that might endanger the system in-orbit or under test, and eventually reducing the overall cost of the AIT & operation phases.

These general considerations motivated the launch of the DIAMS program by the mid-eighties. DIAMS is a step-wise fault diagnosis expert systems development programma initiated by Matra Marconi Space with support from CNES in 1986.The analysis of the DIAMS programma illustrates the progressive approach adopted by MMS to master the inherent complexity of the knowledge required while delivering successive generations of knowledge-based tools that can actually provide support in spacecraft operations.

II. DIAMS-0: the first steps

First experiments in the domain of diagnosis were conducted in 86. An early mock-up was developed in Smalltalk. It allowed to confirm some basic knowledge representation and reasoning principles

and particularly the importance of model-based approaches and object-oriented knowledge representations.

The Object-Oriented (OO) paradigm was found well-suited to the implementation of knowledge -based systems. In the OO paradigm, each elementary problem-solving competence may be attached as a method to one or several domain object classes.

The Model-Based approach clearly distinguishes on the one hand the application domain which is modelled in terms of functional or behavioral components and on the other hand generic reasoning mechanisms that can interpret such models and work on them. KBS implementing the model-based approach may be decomposed into domain-independent modules - the *KBS shell* - on the one hand and *domain-specific Knowledge Bases (KB)* on the other hand. The KBS shell implements the core of the inference process (basic knowledge representation and reasoning mechanisms, general problem-solving strategy) and the external communication services (user interface, interface with the operational environment). It is generally reusable for other target systems of the same nature, possibly through customizing of the external communication services. The Knowledge Bases are generally specific to the target system (a specific S/C system or subsystem for instance).

III. DIAMS-1: Establishing the founding principles

The development of a first generation of diagnostic tools, DIAMS-1, started in 1986. The project was co-sponsored by the French Space Agency. It led to the delivery of a prototype Expert System dedicated to the TELECOM 1 Attitude and Orbit Control System (AOCS) [7]. The selected implementation platform was the SUN/UNIX environment and an object-oriented dialect on top of Prolog called Emicat. Graphical interfaces were developed on top of Sunview. The prototype was installed in the TELECOM 1 SCC and evaluated by the operation staff in 1989 [8].

Setting up the basic knowledge representation and reasoning mechanisms

Knowledge Islands

One of the main advances realized through DIAMS1 was the decomposition of the knowledge base into different categories of so-called *Knowledge Islands* (KI) representing the different domains of expertise required for diagnosis

- hierarchical decomposition of the system into functions with identification of basic commands and observables
- qualitative models of behavior
- shallow knowledge required for solving the most common problems or to deal with situations where the expert understanding is not deep enough to include a functional or a behavior model

The notion of knowledge island turned out to be particularly well-suited to the management of the different natures of knowledge. It greatly facilitated the KB maintenance and incremental refinement. It also made easier the local implementation of new types of knowledge, including new or refined knowledge representation paradigms designed to achieve a finer representation.

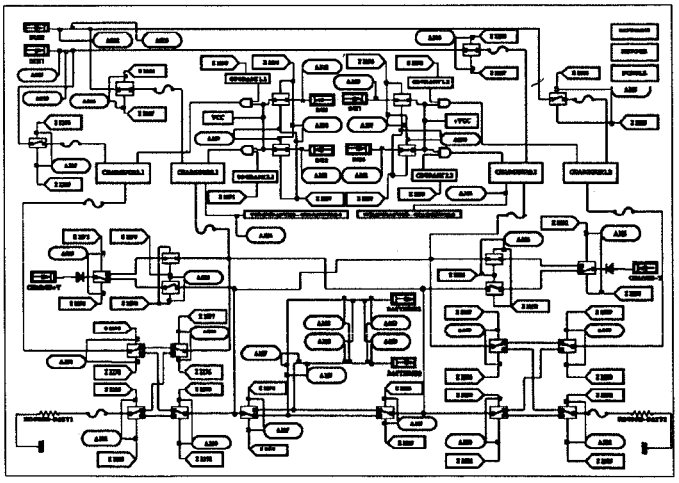
Functional knowledge

The functional model consists of a set of functional diagrams, grouped into knowledge islands, and describing at the component level:

- the functional elements of the system,
- the functional links, representing possible influences between functional elements,
- the observable parameters (telemetry) associated to some of the functional links, and the available telecommands.

The functional model is hierarchical and its deeper level corresponds to the limits of the satellite commandability and observability. It depicts telecommands and telemetries connections and corresponds to the switching diagrams used in S/C operation engineering activities (figure 1).

Figure 1. Example of functional diagram



For each functional element, a propagation function defines how abnormal influences received are propagated to other elements, under the assumption that it is nominal (not faulty). It describes how this component responds to abnormal input influences, or how its inputs can be abductively suspected when its outputs are in abnormal states.

The main justification of this hybrid model based approach is that, because the systems modelled are very complex, the functional elements do not have a general description of their behavior. In other words, the model is not built to provide predictions of all the possible behaviors of the modelled system. It is rather a qualitative representation of the possible fault propagation between the components of the system. The fault modes of the suspected unit(s) are defined only by their signatures in terms of abnormal output(s). Fault modes do not need to be systematically identified a priori. Interactions between components can stand for all kinds of physical signals (e.g. electrical, command signals, thermal influences). A very restricted set of states has been shown sufficient in most cases to represent the propagation of faults over the functional layouts.

Diagnostic reasoning in a functional KI may be decomposed into three fundamental tasks which are:

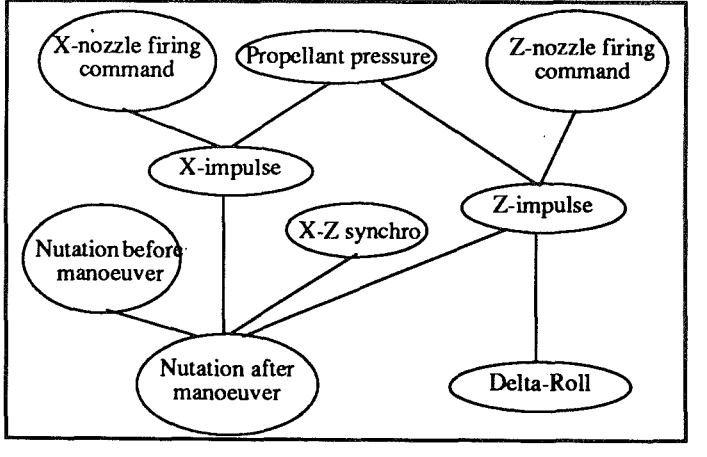
- hypotheses generation: given suspect links pointed out by a behavior analysis or by previous analyses in other functional KI's, find out which functional elements might account for the symptoms. This result is achieved by backward propagation of the anomalies through the links between the functional elements, using the propagation functions abductively.
- hypotheses elaboration: given the set of suspected functional elements given by the reasoning in the previous step, determine what the impact of their fault would be on the observables of the KI currently investigated. This is achieved through forward propagation through the links, using the propagation functions deductively.
- hypotheses discrimination, that is discriminate among the hypotheses coming from the first step by adding more information about other observable parameters generated at the second step. The principle of the diagnosis is then to enter a discrimination loop between the possible causes. The system selects an observable according to various criteria, like the reliability of the measure or the discrimination power of the observable, and then asks for its qualification. Depending on the nature of the response, some possible causes are discarded (the ones which are incompatible with the qualification of the observable given by the user). If there are still discriminating observable parameters, another step of the loop is entered, otherwise the result of the diagnosis is either a single cause or a set of non discriminated possible causes.

Behavior knowledge

The behavior Knowledge meets the requirement for system level knowledge that allows to rapidly get a partial conclusion about the origin of the problem (reconfiguration criterion, global fault corresponding to some system state variables) and then to focus the attention on some subfunctions of the functional model and so to limit the exploration of the functional model to these subfunctions.

Standard forms were defined to capture the AOCS behavior knowledge. These forms were used to specify in a systematic way all the observables (e.g., the roll angle), system variables (like the nozzle firing command or the nozzle state variable) and the observable manifestations (e.g., the displacement of the S/C nutation center along the roll axis after an actuation sequence) necessary to represent the behavior of the system together with the relationships existing between these different elements. The behavior model also contained a number of causal relationships representing the AOCS automatic reconfiguration logic. Once this information was entered in the KB, the KBS shell could build the causal graphs relating system variables, fault modes, and observable manifestations, and discriminate between them using the same generic inference mechanisms as in the functional model (figure 2).

Figure 2. Examples of behavioral relationships



Lessons learned from the experimentation phase

The main results of the experimentation phase were gathered in a document jointly elaborated with the Telecom 1 operations [8]. The experimentation of the prototype was very useful in clarifying the situation and mission of the expert system in the SCC and in refining the operational requirements. It confirmed DIAMS-1 basic knowledge representation and reasoning mechanisms. The general conclusion was that the DIAMS approach improved the communication between the S/C manufacturer and the SCC staff, and that, as a model-based system, DIAMS provided the SCC staff with a better knowledge of the S/C functions and behavior. The experimentation phase also indicated how additional functionalities could be implemented in future versions of the system.

The DIAMS-1 experimentation phase demonstrated that the approach chosen was ripe for being applied in large scale applications. It convinced the French Space Agency to start the development of a full scale diagnostic support system for TELECOM 2 satellites.

Two of the technical lessons learned during the experimentation phase are worth being recalled here:

- An important part of the S/C knowledge is available under graphical form (functional diagrams for instance). The experimentation emphasized the importance of the graphical model edition and animation services. Graphical model editors are needed for instance for building the functional model and checking the graphical consistency of its hierarchical decomposition. Model animators are needed to display and to animate the appropriate diagrams during reasoning. Models editors and animators require a development tool which offers an object-oriented language for modelling the domain semantics (semantic objects) and integrated graphical utilities to manage the interactions between the semantic objects and their graphical representations.
- It was also remarked that some basic mechanisms could be reused in the framework of the S/C project to support a number of design activities. The hypothesis elaboration mechanism could be for instance adapted to perform impact analyses - e.g., to figure out the impact of a given fault or a given telecommand on the system observables. Impact analysis is one of the main techniques used for instance to elaborate the TM/TC plan or to analyze failure modes effects and criticality (the FMECA) during the S/C design phase. TM/TC Plans and FMECA also are major sources of information for the construction of the KB and the optimization of the diagnostic strategy.

IV. DIAMS-2: Maturing the knowledge modelling and the development process

Through DIAMS-2, MMS addressed the development of a fault isolation tool covering a whole spacecraft: french telecommunication satellite TELECOM 2. This project was the consequence of the very positive results of the development and evaluation of the DIAMS-1 prototype [9][2][3][4].

DIAMS-2 was developed over a period of 4 years from 1989. The selected implementation platform was the KEE/CommonLISP object oriented environment which was considered the reference environment for KBS development when the DIAMS-2 project was started. It also complied with the semantic-graphic integration requirement that resulted from the DIAMS-1 experimentation.

Refining Knowledge Modelling

DIAMS-2 is a hybrid system combining decision tree based symptoms - hypotheses associational reasoning to initiate diagnosis and to focus the reasoning on particular functions and components and the DIAMS-1 model-based techniques to complete diagnostic reasoning on particular functions and to provide the final isolation of the fault.

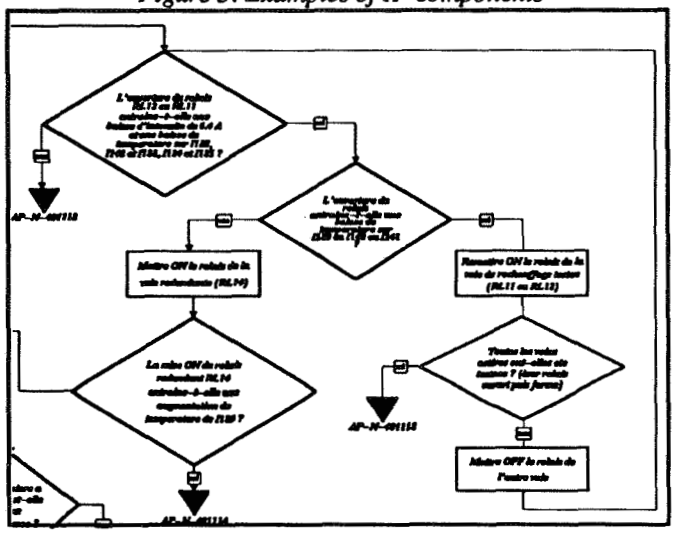
Investigation Procedures

The decision-tree based knowledge, called Investigation Procedures (IP) in the latest generation of the tool, adds a strategic layer on top of the functional model. It is used to select among pending hypotheses and to focus the attention on definite parts of the functional model (figure 3).

IP modelling starts at the system level, implementing a top-down approach. The used knowledge is elaborated by S/C operation engineers during the mission preparation phase. It corresponds to the Contingency Operations section of the Operations Preparation Handbook. IPs can be enriched on the basis of anomalies experienced during the S/C in-orbit lifetime.The knowledge is represented as decision trees whose nodes are either binary tests (e.g., testing whether a given parameter is abnormal) or actions on the satellite (e.g., sending a

telecommand that will allow to discriminate between candidate hypotheses).

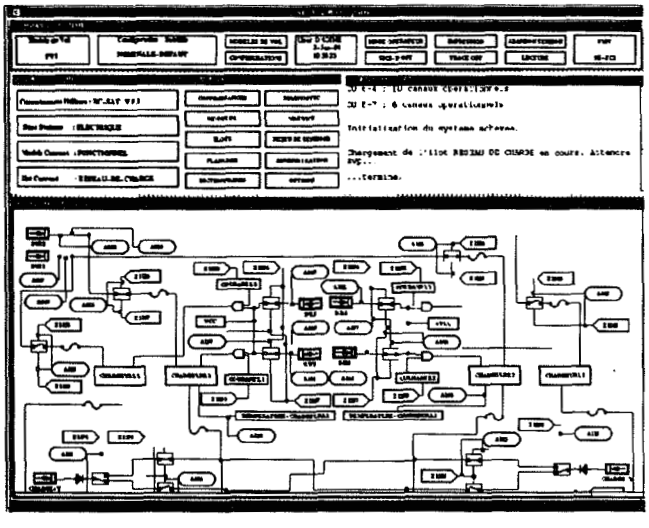
Figure 3. Examples of IP components



A diagnostic session starts when the user inputs a set of anomalies. The initial tests implement a discrimination strategy at system level. These tests are mainly membership tests which aim at localizing the satellite subsystem where the primary anomalies have occurred. This kind of procedures can often be automated.

At subsystem level, the diagnostic strategy consists in using as far as possible higher level observations and characterizations of the satellite behavior or evolution, in order to simplify or even avoid in-depth analyses involving the functional model. Connections with the functional model are reached when tests involve large numbers of telemetries and need reference states to compare the current situation with.

Figure 4. Investigation of a functional KI with DIAMS-2



Maturing the Development Process

Moving to a full scale industrial application raises stringent requirements in terms of Knowledge Management and KBS Development Methodology. With support from CNES, MMS elaborated a first set of Software Engineering principles and Quality Assurance rules applicable to KBS projects that benefited from the experience acquired in DIAMS-1.

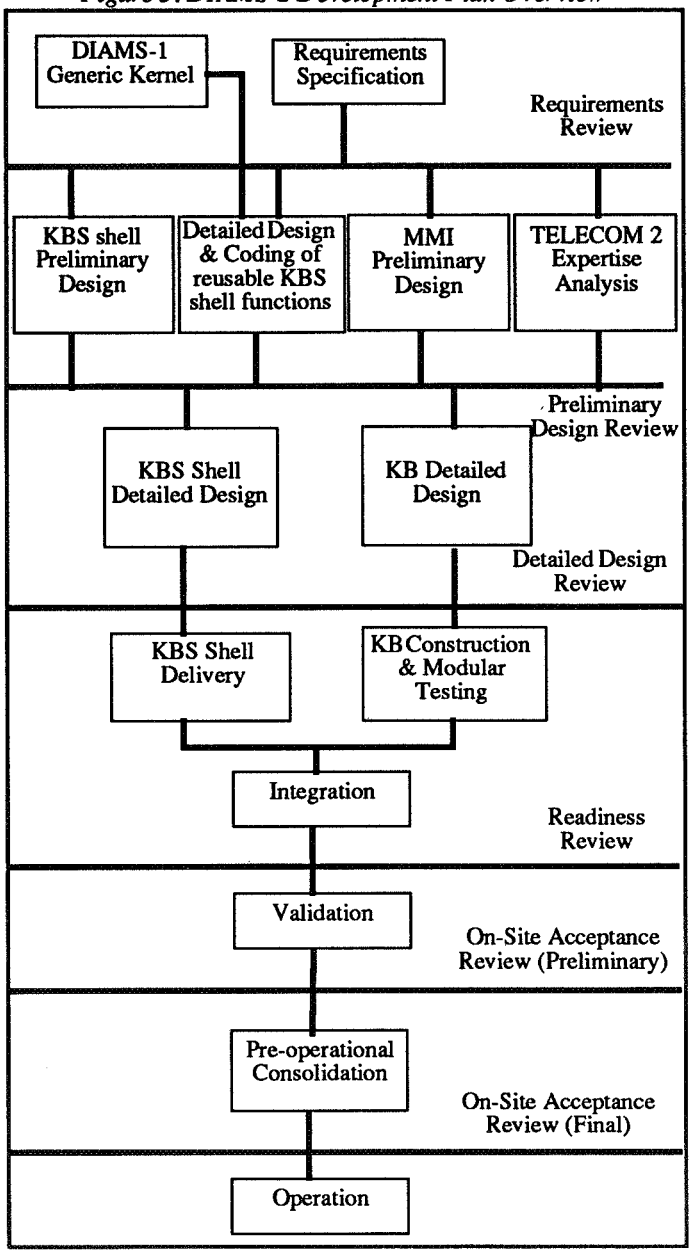
The construction of the Knowledge Base was conducted by a dedicated team independent from the KBS shell development team. The KB development team performed the capture of knowledge and the construction of the KB using well-suited methods and tools in compliance with the representational constraints of the operational environment. It also maintained close relationships with the target system project organization - essentially through cooperation with the TELECOM 2 operation engineering team; the System, Subsystem and Integration specialists of the S/C project did not directly participated in the construction of the KB.

The development of a KBS shell is rather similar to a conventional SW development, and requires the same kind of methods and tools for design, coding and testing. The design of the DIAMS-2 KBS shell inherited most of the basic knowledge representation and reasoning mechanisms already implemented in the DIAMS-1 prototype and validated during the experimentation phase. A dedicated team assumed the design, coding and testing of the tool basic functionalities. A third team, independent from the development teams, was in charge of the quality control and of the integration and final validation of the KBS.

A pre-operational consolidation phase was scheduled in the continuation of the KBS development phase. Its goals were

- to familiarize the SCC staff with the KBS
- to experiment and eventually to enact the KBS utilization and maintenance procedures
- to consolidate and validate the external interfaces with the SCC information system, including the S/C and Simulator data access procedures.
- to calibrate tests and explanations on-site with the end-users.
- to refine some knowledge islands to account for the in-orbit experience (e.g., the S/C in-orbit thermal behavior).

Figure 5. DIAMS-2 Development Plan Overview



Integrating the end-user in the development process

Cooperation between the KB development team and the SCC staff is needed, during the construction of the KB, to ensure consistency between the knowledge representation formalisms used in the SCC and those used in the KB. A close cooperation is also needed when the system is transferred from the development site to the operation site.

In DIAMS-2, the integration of the end-user in the development cycle was founded on the following principles.

- The S/C User's Manual (UM) remained the reference document for the transfer of information between the S/C manufacturer and the SCC. The level of decomposition of the models was the UM's one, and the same graphical representation modes, and variable identifiers were used.
- Operation Engineers from the S/C project were involved in the development process to continuously maintain consistency between the DIAMS-2 KB and the S/C User's Manual.
- A TELECOM 2 SCC representative was included in the KB development team. His mission was to check that the knowledge representation used (symbology, nomenclature) was consistent with the one used in the SCC, that the functional model was compatible with the hierarchical view of the S/C and the monitoring sets defined in the SCC, and that the observables used were actually accessible through the SCC. Conversely the KB was developed in such a way that the SCC engineer could draw benefit from the KB design and development activity.

Remark: The TELECOM 2A/2B launch campaigns took place during the DIAMS-2 KB Detailed Design phase. This resulted in a lack of availability from both the S/C operation engineering team and the SCC personnel. A first consequence was that an important effort had to be devoted to the refinement of the KB during the pre-operational consolidation phase. This again confirmed the crucial importance of a right phasing with the S/C and SCC development activities, and more generally of a tighter integration between the KBS, SCC and S/C development processes.

V. DIAMS-3: the Integration Age

In DIAMS-2, comprehensiveness and efficiency was privileged against fineness of representation and reasoning. Simplified representations of knowledge, generally well-suited to the practical problems faced in spacecraft operations were introduced as a first approximation. However, in some specific knowledge islands, refined representation and reasoning techniques are required to appropriately handle time, incompleteness and uncertainty. This last refinement step is now being considered through the development of a new generation of diagnostic tools called DIAMS-3 that started in 1992 [5].

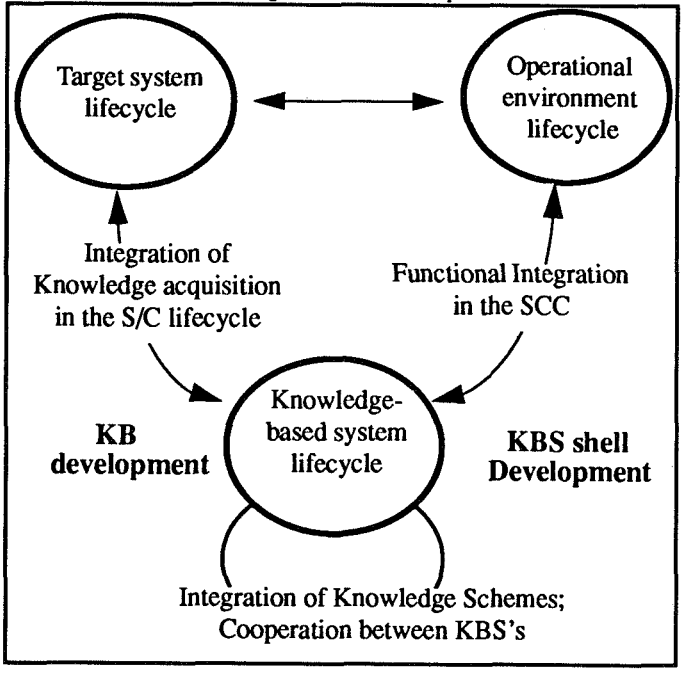
Other important objectives of DIAMS-3 concern the reduction of the knowledge acquisition efforts, tighter integration with other knowledge-based tools like data analysis or procedure management tools, and more generally the complete integration of the diagnostic system in the operational loop [10].

C++ is the implementation language retained for DIAMS-3. Beyond porting the DIAMS-2 machinery into C++, DIAMS-3 provides generic model edition services and a set of libraries of operational standard for handling time, incompleteness and uncertainty and for cooperation with other knowledge-based tools (knowledge interchange format and protocol, mapping engine, exchange monitor, etc.). These libraries and basic services, all developed in C++, will be reused in other KBS development projects.

Integration Issues

The different integration issues raised by the operational integration of the diagnostic tool in SCC's or AIT environments have been addressed through a European project called UNITE, co-sponsored by the Commission of the European Communities. They are illustrated hereafter (figure 6).

Figure 6. Main Integration Issues explored in UNITE



1) A first issue concerns the *integration of different knowledge schemes within a given KBS*. Diagnostic systems in Space indeed require the implementation and integration of different knowledge representation and reasoning paradigms:

- they need to handle different domain models representing different views of the satellite system (e.g., thermal view, mechanical view, electrical view, etc.).
- the input information, be it provided by human users or by SCC monitoring facilities, is sometimes numeric but more often symbolic, intrinsically uncertain and imprecise, with a validity time frame.
- the basic inference mechanisms are themselves, e.g., exploiting uncertain and imprecise symbolic transfer functions (such as qualitative fault propagation functions) which may need to handle time to reflect the variation of dynamics between different views of the system.
- diagnostic reasoning deals with qualitative temporal propositions with a start, an end and a persistence.
- dependency tracking and maintenance of consistency between different reasoning contexts, or the management of the assumptions and time-constraints under which statements are valid, may require the parallel handling of several uncertain and time-dependent alternative hypotheses.

One of the goals is to give the knowledge engineer the flexibility to choose the most appropriate knowledge representation for some aspects of the problem (e.g., various representations of time and uncertainty), and yet process them in an integrated manner.

2) A second kind of need is concerned with the *sharing and exchange of knowledge between KBS's* that need to cooperate to achieve some global problem solving task. For instance monitoring, diagnostic and data analysis tools need to cooperate to detect and then locate the origin of anomalies. They may need to exchange knowledge or complex information. As the formalisms used to represent this information may vary from KBS to KBS, it is necessary to set up translation mechanisms, from the formalisms of each KBS to a common Knowledge Interchange Format and vice-versa. The approach followed by MMS in that domain is experimental. The goal being to assess the level of maturity and the applicability of existing solutions like those elaborated within the Knowledge Sharing Effort [12].

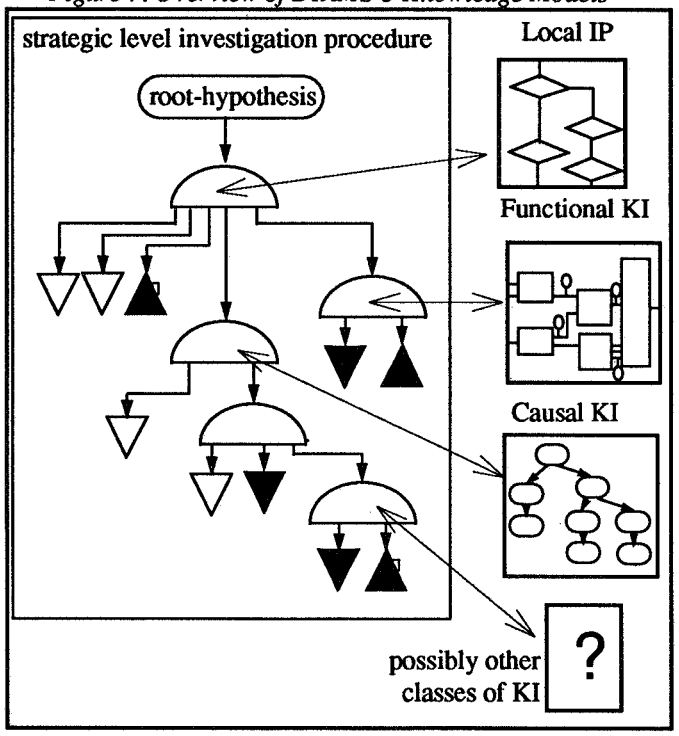
3) *Functional Integration* regards cooperation between the KBS and conventional software modules or database management systems for the construction of fully integrated operational applications. The methodology issues raised by the operational integration of the diagnostic tool in the SCC are investigated in [1]. Functional integration requires a hybrid methodology framework for co-existing conventional / knowledge-based developments.

4) Finally the DIAMS experience feedback has emphasized the importance of a better *integration of the knowledge capture tasks in the S/C lifecycle.*

Integration of knowledge models

The following figure provides a synthetic view of the different types of knowledge models explored through DIAMS-1 and DIAMS-2 and further refined and integrated in DIAMS-3 (figure 7).

Figure 7. Overview of DIAMS-3 Knowledge Models



Causal KI

In the latest version of the tool, behavioral knowledge (also called *causal knowledge*) is composed of a reduced set of FMECA related to a family of symptoms, that allows to explore and refine some higher level hypothesis. This is a natural extension of the notion of behavior model explored in DIAMS-1.

Incompleteness is inherent to FMECA. A more flexible representation of the effects of fault modes has been proposed that eases expression of knowledge, down to the relevant level of detail (i.e., events chronologies), and that does not make any assumption about what is not said explicitly [6].

Handling of time, incompleteness and uncertainty

Some improvements brought by DIAMS-3 should allow to better handle time, incompleteness and uncertainty. Different techniques have been proposed for handling incompleteness, uncertainty or time-dependency. The investigation of the current practice shows that many difficulties in terms of performance or complexity have been experienced in deploying these techniques in industrial contexts and that ad hoc adaptations or simplifications are generally done by the development teams to match the industrial constraints. Beyond adequation to the specific knowledge representation and reasoning needs of the diagnostic tool, performance and complexity thus shall be the main criteria for the assessment of candidate solutions in that domain.

For instance, the information available about the symptoms is incomplete: many observables are not fully monitored in real time. Allowing the users to express their uncertainty about the interpretation of the observable was also recognized as a need. Indeed, some observations involve complex combination and abstraction of elementary pieces of data, followed by a high level interpretation of the result. Adequate formalisms are needed to handle incompleteness and allow expression of uncertainty about the presence/absence of a manifestation.

From a discrimination point of view, graduality in the uncertainty of the fault effects and in the characterization of the observables has been introduced. It allows a ranking of the solutions given by the system. As the diagnostic process is iterative, it was also found useful to have advice with respect to the selection of the next observables to be tested. This is achieved through a utility function that assesses the impact of the test of a manifestation on the possibility of fault mode.

Application developers will be provided with libraries of basic knowledge representation and reasoning mechanisms that can be easily included into application programs without imposing the use of

any particular development tool for the implementation phase. Considering the current trends in Information Technology, libraries of C++ objects seemed to be the best possible choice for DIAMS-3.

A first set of libraries of reasoning schemes have been selected, developed or re-developed in C++, and appropriately encapsulated to answer DIAMS needs:

- A new reasoning scheme which allows to represent and process incomplete and uncertain relations between faults and manifestations (such as FMECA) in a diagnostic context. The core model, based on the possibility theory, includes consistency-based and abductive diagnostic algorithms eploiting uncertain observations, as well as additional tools to measure the utility of tests and the discriminability of a set of fault modes [6]. Extensions of this model to the processing of functional knowledge are being developed.
- A Valuation Based System (VBS) which allows uncertain reasoning in a causal graph with various formalisms, e.g. bayesian, possibilistic, Dempster-Shafer's Theory of Belief, etc.
- A Time Constraint Propagator (TCP) which enables the comparison of an actually observed chronology of events with an a priori knowledge about the causal relationships between events. An hypothesis is confirmed by the TCP when all observed events occur at scheduled dates. If any of the observed events occurs outside the expected time window then the hypothesis is inconsistent and therefore is discarded. When the hypothesis-related events have not yet occurred - the hypothesis can be neither confirmed nor discarded - the hypothesis is said incomplete and TCP provides the validity interval for that hypothesis.

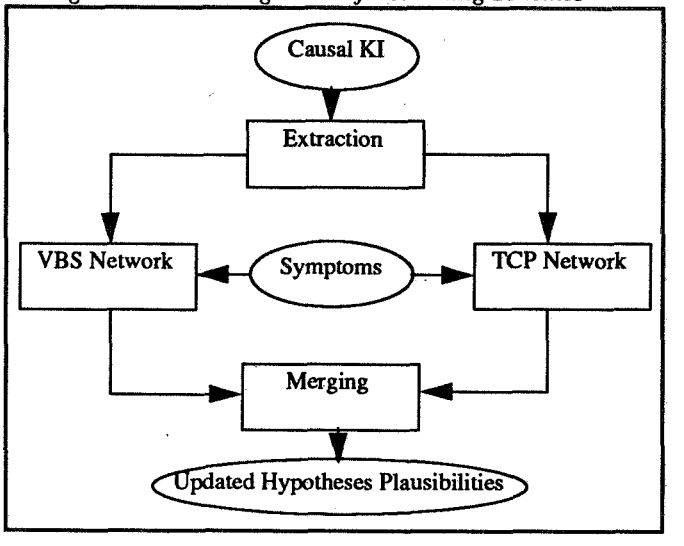
Integration of reasoning schemes

The joint utilization of the TCP and VBS in a diagnostic context is illustrated by figure 8.

Sometimes such a (weak) integration approach may not be sufficient. Reasoning threads may be too intertwined to be processed efficiently in a separate way. A prototype has been developed to tackle this kind of problem and to evaluate the candidate technology. It addresses the so-called "strong integration" of temporal and uncertain reasoning in a model based diagnostic context. The computational approach consists in generating an ATMS network - Assumption-based Truth Maintenance System - to compute explanations for symptoms. A possibilistic, temporal, cost-bounded ATMS machinery is used. The cost-bounded feature allows to focus of the reasoning process and to limit computational costs.

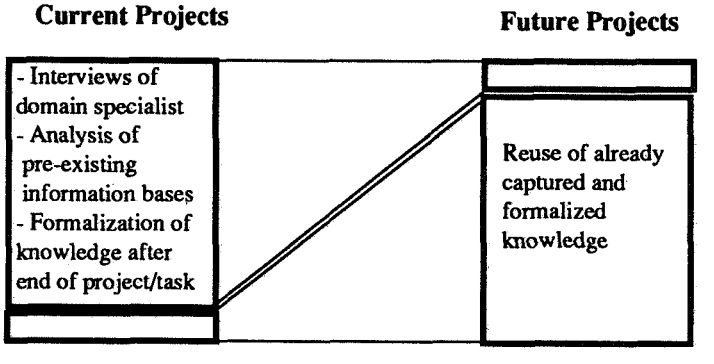
The main risk identified for strong integration is performance. The strong integration approach is currently considered as experimental and is not included in the DIAMS technical baseline.

Figure 8. Weak Integration of Reasoning Schemes



Integration of knowledge acquisition in the S/C lifecycle

The reduction of the knowledge acquisition costs was a permanent concern in each phase of the DIAMS program. A first conclusion was that, in order to improve the interactions with S/C specialists, the knowledge modelling activity should benefit to the S/C project tasks. The goal in DIAMS-3 is now to reach a level of expressiveness and genericity such that the DIAMS knowledge bases could be built and reused throughout the satellite lifecycle. This should contribute to significantly reduce the knowledge acquisition costs.



VI. Concluding Remarks

The DIAMS program followed a spiral approach, each cycle partially or fully implementing a reference development cycle. The DIAMS spiral lifecycle model is summarized in table 1. Matra Marconi Space is now involved in a tool improvement cycle (DIAMS-3) that would enable a tighter integration of the diagnostic system in ground infrastructures. A more general objective is to set up the techniques, methods and tools that will allow to consider the KBS technology as a baseline technology for the development of future S/C Control Centers or AIT Environments.

The knowledge acquisition issue remains pivotal. It comes down to the following two questions

- How to maximize the reuse of already formalized and managed knowledge?
- How to adapt the S/C project tasks and deliverables so that knowledge could be acquired 'on the fly' during S/C developments ?

A number of solutions have been proposed to proceed in this direction. The on-going experiments should prove that these solutions are ripe for introduction in S/C projects.

Acknowledgments

Work related to DIAMS-1 and DIAMS-2 has been supported by CNES since 1986. Work related to DIAMS-3 has been partially funded by CEC through UNITE project (ESPRIT project 6083) since 1992. Other partners involved are: Cap Gemini Innovation (France), Queen Mary Westfield College (UK), Sintef Delab (Norway), Eritel (Spain), ITMI (France).

References

[1] AYACHE S., HAZIZA M., CAYRAC D.: "Delivering Spacecraft Control Centers with Embedded Knowledge-Based Systems: the Methodology Issue", Intl. Symposium on Space Mission Operations and Ground Data Systems, Greenbelt (USA), 1994

[2] BASTIEN-THIRY C., MAURIZE J.C.: "SE-TC2 : The first expert system in a CNES Satellite Control Center", ESA Workshop on Artificial Intelligence and KBS for Space, Noordwijk (NL), 1993.

[3] BRENOT J.M., CALOUD P., VALLUY L.: "The development of an operational expert system for the Telecom2 satellite control centre", ESA Workshop on Artificial Intelligence and KBS for Space, Noordwijk(NL), 1991.

[4] BRENOT J.M., CALOUD P., VALLUY L., GASQUET A. : "On the design and development choices to bring to operation a diagnostic expert system for the Telecom 2 satellite", Intl. Conf. on Fault Diagnosis, Toulouse (F),1993.

[5] CAYRAC D., HAZIZA H. : "Management of Uncertainty and Temporal Dependencies in Real World Diagnostic Systems, Application to the Space Domain", Intl. Conf. on Fault Diagnosis, Toulouse (F), 1993.

[6] CAYRAC D., DUBOIS D., HAZIZA M., PRADE H.: "Possibility Theory in Fault Mode Effects Analyses", IEEE World Congress on Computational Intelligence, Orlando (USA), 1994

[7] HAZIZA M.: "An Expert-System shell for satellite fault isolation based on structure and behaviour", Workshop on AI Applications to Space Projects, Noordwijk (NL), 1988.

[8] HAZIZA M., GIBET L., DEBAY P.: "DIAMS: An Expert-System shell for satellite fault isolation - The User Feedback", 3rd Intl. Conf. on Human-Machine Interactions and AI in Aeronautics and Space, Toulouse (F), 1990

[9] HAZIZA M.: "Towards an operational fault isolation expert system for french telecommunication satellite TELECOM 2", Intl. Symposium on Ground Data Systems for Spacecraft Control, Darmstadt (G), 1990

[10] HAZIZA M.: "Delivering Operational Space Applications with Embedded Knowledge-Based Systems", World Congress on Expert Systems, Lisbon (P), 1994

[11] NECHES R., FIKES R., FININ T. GRUBER T., PATIL R., SENATOR T., SWARTOUT W.: "Enabling Technology for Knowledge Sharing", AI Magazine, Fall 1991

Table 1. The DIAMS Spiral Lifecycle Model

Phase	DIAMS-0	DIAMS-1	DIAMS-2	DIAMS-3
Characterization in the large	X	(X)		(X)
Characterization in the small	X	(X)		(X)
Analysis	Test-Bed Implementation (Smalltalk)	X	(X)	(X)
Architectural Design		X	(X)	(X)
Detailed Design and Coding		Prototype Implementation Experimentation Phase	X	X
Verification & Validation			X	X
Operation & Maintenance			X	X

111120

N95-17535

353990

P. 6

USING GRAPHICS AND EXPERT SYSTEM TECHNOLOGIES TO SUPPORT SATELLITE MONITORING AT THE NASA GODDARD SPACE FLIGHT CENTER

Peter M. Hughes & Gregory W. Shirah
Automation Technology Section (Code 522.3)
Mission Operations and Data Systems Directorate
NASA/Goddard Space Flight Center
Greenbelt, Maryland 20771

Edward C. Luczak
Computer Sciences Corporation
7700 Hubble Drive
Lanham, MD, 20706

ABSTRACT

At NASA's Goddard Space Flight Center, fault-isolation expert systems have been developed to support data monitoring and fault detection tasks in satellite control centers. Based on the lessons learned during these efforts in expert system automation, a new domain-specific expert system development tool named the Generic Spacecraft Analyst Assistant (GenSAA), was developed to facilitate the rapid development and reuse of real-time expert systems to serve as fault-isolation assistants for spacecraft analysts. This paper describes GenSAA's capabilities and how it is supporting monitoring functions of current and future NASA missions for a variety of satellite monitoring applications ranging from subsystem health and safety to spacecraft attitude. Finally, this paper addresses efforts to generalize GenSAA's data interface for more widespread usage throughout the space and commercial industry.

INTRODUCTION

A group of spacecraft analysts are responsible for the proper command, control, health and safety of each spacecraft managed by NASA's Goddard Space Flight Center (GSFC). During numerous contacts with the satellite each day, these analysts closely monitor real time data searching for combinations of telemetry parameter values, limit violations, and other indications that may signify problems or failures. This is a demanding, tedious task that requires well-trained individuals who are quick-thinking and composed under pressure. However, as our satellites become more complex, this task is becoming increasingly more difficult for humans to conduct at acceptable levels of performance [Ref. 2].

At GSFC, fault-isolation expert systems have been developed to support data monitoring and fault detection tasks in satellite control centers. Based on the lessons learned during these efforts in expert system automation, a new domain-specific expert system development tool named the Generic Spacecraft Analyst Assistant (GenSAA), was developed to facilitate the rapid development and reuse of real-time expert systems to serve as fault-isolation assistants for spacecraft analysts. Although initially developed to support GSFC's satellite operations, this powerful tool can support the development of highly graphical expert systems for data monitoring purposes throughout the space and commercial industry.

This paper describes GenSAA's capabilities and how it is supporting monitoring functions of current and future NASA missions for a variety of satellite monitoring applications ranging from subsystem health and safety to spacecraft attitude. Finally, this paper will address efforts to generalize GenSAA's data interface for more widespread usage throughout the space and commercial industry.

GenSAA OVERVIEW

GenSAA is an advanced software tool that allows the rapid development of intelligent graphical monitoring systems. Through the use of a highly graphical user interface and point-and-click operation, GenSAA facilitates the rapid, "programming-free" construction of graphical expert systems to serve as real-time fault-isolation assistants for spacecraft analysts.

GenSAA expert systems are easily built and maintained using an integrated set of utilities called the GenSAA Workbench which are used to define the expert system's telemetry data interface, rule base, and X/Motif-based user interface. GenSAA insulates the expert system developer from the complicated programming details of the systems with which the expert system will interface. This tool promotes the use of previously developed rule bases and graphic objects, thus facilitating software and knowledge reuse and a further reduction in development time and effort.

The development of GenSAA was motivated by the lessons learned from a research effort to evaluate the value and effectiveness of using graphical rule-based expert systems for fault detection purposes. The project, which was named the Communications Link Expert Assistance Resource (CLEAR), was quite successful. Although CLEAR was initially conceived to serve as a proof-of-concept prototype, it was ultimately used to support real-time operations for NASA's Cosmic Background Explorer (COBE) satellite where it was instrumental in demonstrating the advantages that expert systems offer mission operations. More importantly, CLEAR provided insights into how expert systems could be developed more quickly and with less effort. GenSAA addresses this issue by insulating the expert system developer from the programming details by employing a "drag and drop" method of developing these systems.

In addition to meeting the previous objective, GenSAA was created as an alternative to high-end, complex and expensive commercially available expert system development environments. In an attempt to meet a wide variety of application needs, these general-purpose programming tools are often too complex to be effectively used by domain experts (spacecraft analysts in this case) to create graphical expert systems. They typically require weeks of training and specialized programmers to implement the data interface, graphical user interface, or rule base for each expert system. GenSAA empowers the spacecraft analysts to easily select the data to be monitored, layout and define the behavior of the expert system's user interface and build rules for fault detection purposes without the intervention or delay of programmers.

GenSAA consists of two major components: a Workbench and a Runtime Framework. [see figure 1]. The Workbench is used to specify expert systems in an offline mode (i.e., not connected to a live data source). The Workbench creates several resource data files that are read into the Runtime Framework which uses these resource files and connects to the data source.

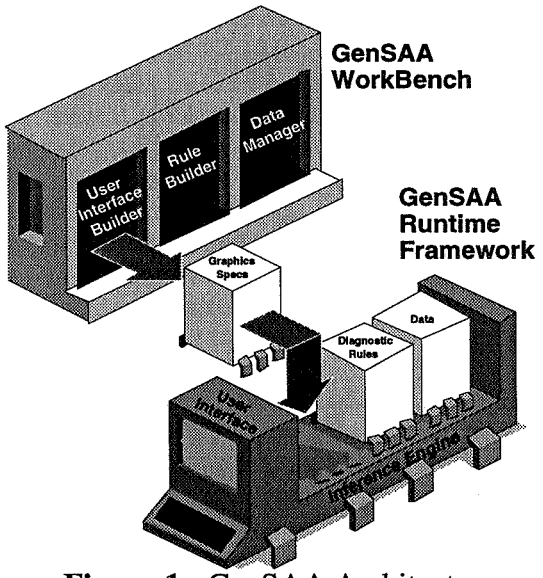


Figure-1: GenSAA Architecture

The GenSAA Workbench consists of a Data Manager, a Rule Builder, and a User Interface Builder. The Data Manager is used to select the telemetry data that is desired for use by the expert system; the Rule Builder is used to create expert systems rules based on the telemetry data; and the User Interface Builder allows the user to create graphical user interfaces to display the telemetry data and the data inferred from the expert system rules. The GenSAA Workbench is tightly integrated and easy to use, employing direct manipulation techniques such as "drag and drop." The Workbench also provides

mechanisms to automatically generate expert system rule statement syntax.

The GenSAA Runtime Framework is the executive for a GenSAA Expert System. It controls the user interface, distributes the real-time data received from the data server, and manages rule execution. The core element of the Runtime Framework is the 'C' Language Integrated Production System (CLIPS). CLIPS is an inference engine and rule-based programming language that was developed at the NASA Johnson Space Center. It is widely used throughout NASA, other government agencies, academe, and the commercial sector.

Expert systems that are created using GenSAA require no source code development, and therefore facilitate very rapid development life cycles. Changes and enhancements to existing expert systems can also be made rapidly at very low cost.

GenSAA runs on Sun and Hewlett-Packard UNIX workstations using X-windows with Motif. Earlier this year GenSAA was delivered to operations for acceptance testing. At the time of publication, it is expected that GenSAA will be in operations in a number of divisions at GSFC and at a few external sites.

The next sections describe several specific applications of GenSAA at GSFC. The first group of applications is associated with spacecraft attitude monitoring. The second group is associated with the monitoring of spacecraft and their payloads. The applications are currently under development and should become operational soon.

GenSAA APPLICATIONS SUPPORTING FLIGHT DYNAMICS

GSFC's Flight Dynamics Division (FDD) is responsible for maintaining the orbit and attitude of many Goddard spacecraft. The FDD has used Heads Up Displays (HUDs) for previous missions to graphically portray attitude

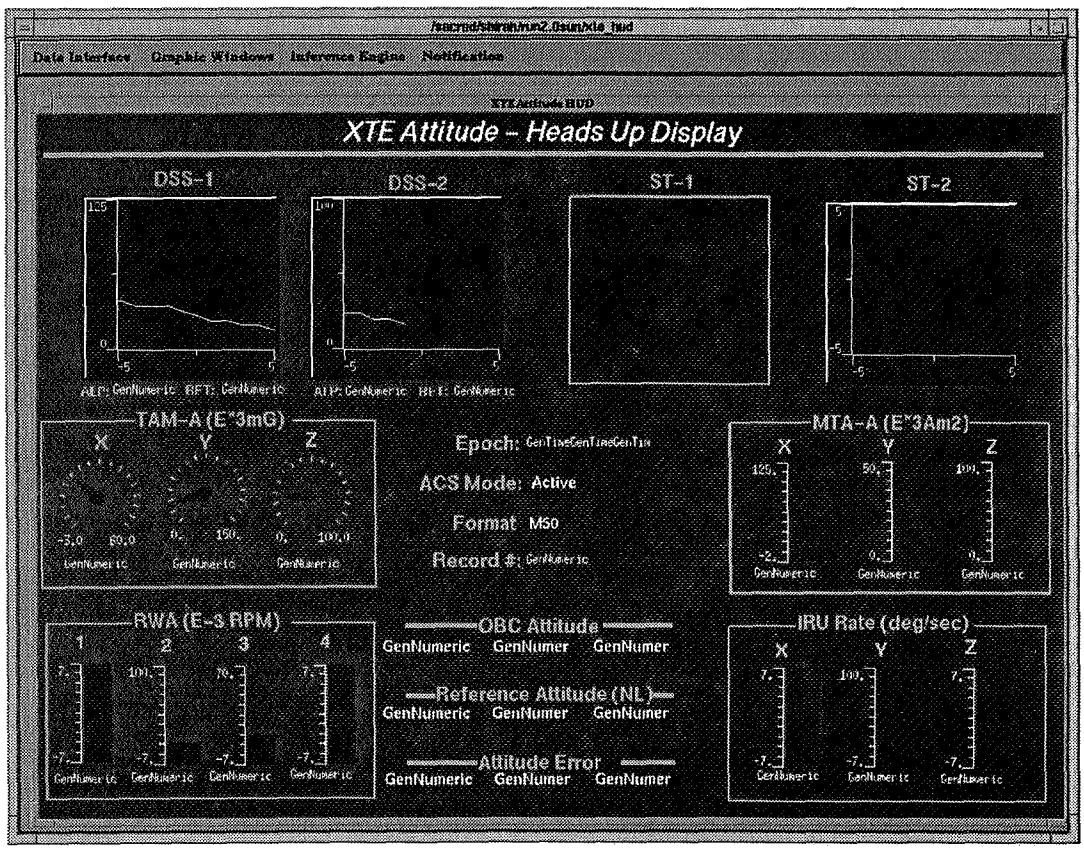


Figure-2: XTE HUD Created Using GenSAA

and orbit parameters in a manner that is similar to the gauges and dials that appear in an airplane cockpit. These HUDs enable flight analysts to quickly view the basic orbit, attitude, and sensor status of a given spacecraft.

The FDD is using GenSAA to create the HUDs for the X-ray Timing Explorer (XTE) spacecraft, the Submillimeter Wave Astronomy Satellite (SWAS) spacecraft, and the Solar and Heliospheric Observatory (SOHO) spacecraft. These missions are among the first FDD missions to be supported on UNIX workstations using X-windows. By using GenSAA, the FDD expects to reduce the effort needed to create the HUD while increasing the ability to respond to change requests.

The XTE and SWAS HUDs are using GenSAA's inference engine to infer engineering unit values based on raw telemetry values. The inferred engineering unit values are displayed on the HUD via graphical and textual user interface objects. Values that are displayed include: magnetometer, gyroscope, and torquer bar biases and rates, guide star and sun sensor positions, and predicted versus actual attitude. Figure-2 is an example of a prototype HUD generated with GenSAA for the XTE mission.

The FDD is also using GenSAA to support the SOHO mission. The HUD for SOHO is similar to the XTE and SWAS HUDs, however, SOHO is enhancing the GenSAA Runtime Framework by embedding a number of 'C' functions to compute the spacecraft real-time attitude based on the current telemetry data received from the data server. Although the SOHO HUD development team had the option to link these functions with the inference engine for invocation via expert system rules, this group chose to embed the functions to optimize performance of these computationally intensive attitude algorithms. This situation demonstrates one advantage of having direct access to the source code of GenSAA.

GenSAA APPLICATIONS SUPPORTING MISSION OPERATIONS

In GSFC's Mission Operations Division (MOD), GenSAA is being used to support real time satellite monitoring in the control centers. GenSAA will be used to build simple advisory expert systems that monitor spacecraft telemetry and ground system parameters. Monitoring these parameters during spacecraft contacts has traditionally been the responsibility of satellite operators.

Two of the primary objectives of this organization's applications are to expedite the fault detection and resolution process and to reduce the amount of data (telemetry points) that human operators must monitor in order to assess the current health and status of the spacecraft and the scientific instruments onboard. With GenSAA, spacecraft engineers will develop simple expert systems that will assist console analysts by reducing the number of data points they must monitor from hundreds of sensor values to dozens of derived system level status points.

GenSAA does not constrain the user in how to represent the system being monitored. Some groups are planning to model the functional operations of the system (i.e., functions across subsystems) while others are planning to develop physical models of the system being monitored. For example, the Solar Anomalous and Magnetospheric Particle Explorer (SAMPEX) project plans to develop a series of GenSAA expert systems to monitor the scientific instruments (LEICA [See Figure 3], MAST/PET and HILT) and some of the spacecraft's subsystems including the Small Explorer Data System (SEDS), the attitude control system (ACS), and thermal system.

In contrast, members of the Gamma Ray Observatory (GRO) Flight Operations Team plan to develop discrete expert systems for both functional and physical perspectives. This team plans to develop expert systems to monitor the power subsystem, communications function and a high level health and safety monitoring system. In addition to the above mentioned missions, GenSAA will support satellite operators for Transportable Payload Operations Control Center (TPOCC) based missions including, but not limited to, Wind/Polar, SWAS, XTE, SOHO, Tropical Rainfall Measuring Mission (TRMM) and the Advanced Composition Explorer (ACE) missions.

GenSAA is expected to provide numerous benefits to the mission operations arena at GSFC. In addition to assisting the satellite operators with the data monitoring task,

GenSAA will reduce the development time and effort of the these systems; serve as a training tool for student controllers; and protect against the loss of satellite operations expertise, especially during periods of personnel turnover. This last benefit even spans beyond a single mission; control center expert systems that capture fault-isolation knowledge preserve expertise from mission to mission which may prove to be beneficial as we embark on multi-mission flight operations teams (i.e., a single set of operators responsible for operating multiple satellites) as a means to reduce satellite operations costs.

GENERALIZING GenSAA FOR BROADER USE

A variety of groups outside of GSFC's Flight Dynamics and Mission Operations Divisions have expressed an interest in using GenSAA to monitor their real time data. However, application to other domains has been limited because GenSAA is currently designed to interface to GSFC-specific ground system formats. To broaden GenSAA's potential application, work was begun earlier this year to generalize its data interface to enable it to receive data in other formats.

The approach adopted is to create bridge processes that interface GenSAA to external data sources. A bridge receives data from an external source and converts it to a format that GenSAA understands. A bridge template is being developed that will be used to simplify the construction of bridges for specific interfaces. To facilitate reuse and to accelerate the application of GenSAA to new domains, the GenSAA Project will maintain a library of bridges to databases and other data sources.

To build a new bridge, the installer creates a file containing a description of the variable names and data types to be received from the external interface. This file is used to automatically generate a large portion of the bridge software. The installer must also write a small amount of program code that will request and receive the data. Finally, these software components are

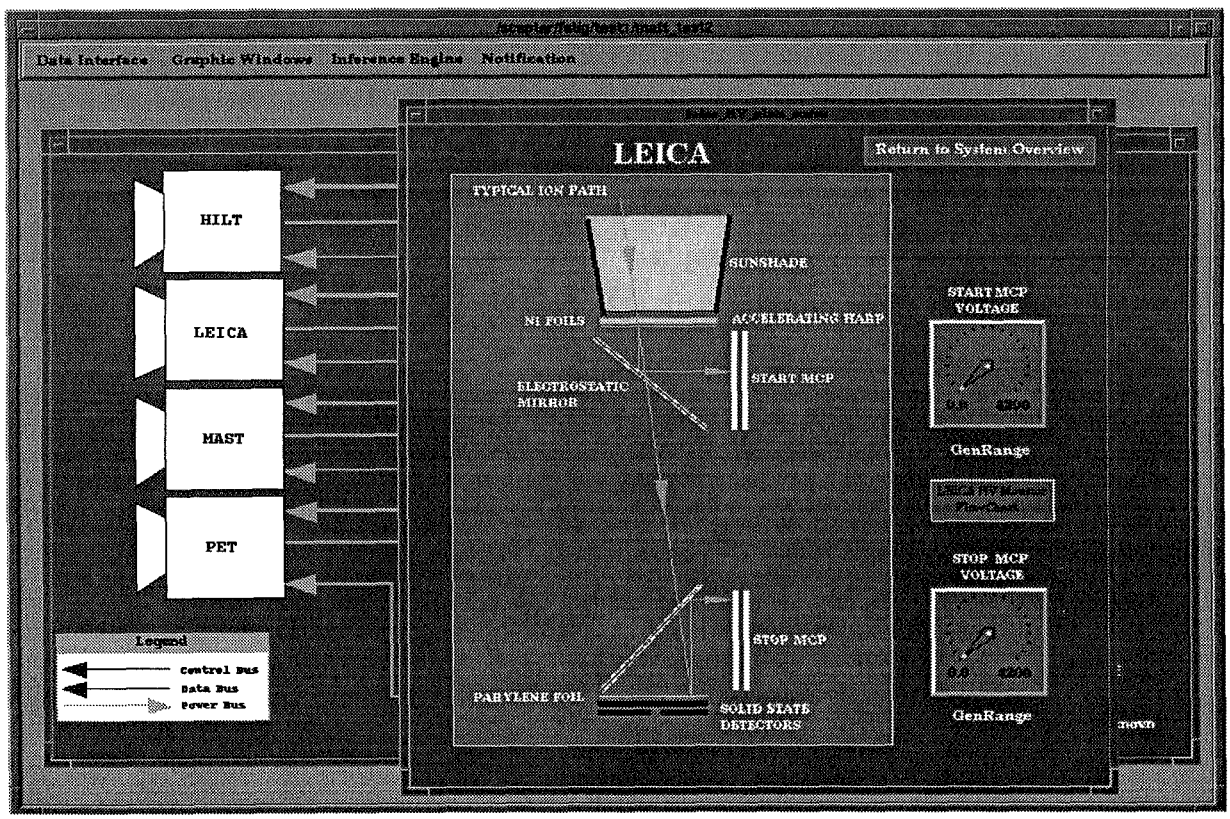


Figure-3: Leica Status Monitor Created Using GenSAA

linked together to form a bridge which provides data conversion capabilities enabling the use of GenSAA in new domains.

AUTOMATING SATELLITE OPERATIONS WITH GenSAA

During the past year, a new research project was started to develop a proof-of-concept prototype that demonstrates how expert system technology can be used to automate routine, nominal-situation control center operations that involve both monitoring and commanding actions. [Ref. 6]

The project is enhancing GenSAA to enable the automation of nominal pass operations for the SAMPEX spacecraft. The enhanced software, called the Generic Inferential Executor (Genie), will perform monitoring and commanding operations in the SAMPEX Payload Operations Control Center (POCC) as specified in a pass script that is defined by members of the Flight Operations Team (FOT). The pass script defines precondition tests, actions, results checks, decision branches, and background monitoring activities. In nominal situations, Genie will execute the pass script without the intervention of FOT members; if an unexpected situation arises, an FOT member will be alerted. Automated operations include verifying the pre-pass readiness test data flow, examining spacecraft event log messages, starting configuration monitors, evaluating system events, initiating the uplink of the daily command load, and initiating dumps from the spacecraft.

The automation prototype will be demonstrated during a live SAMPEX pass. It is anticipated that the results gathered on this project will influence the development of enhanced ground system software that will automate operations in future GSFC missions, including the Earth Observing System (EOS) project.

CONCLUSION

GenSAA is being used to develop several expert systems that will support current and upcoming spacecraft missions. GenSAA is making it easier for spacecraft analysts to build expert systems, and to thereby preserve and apply their spacecraft knowledge in automated monitoring systems.

Reduction of spacecraft mission cost is a high priority at GSFC. GenSAA is providing a means of reducing the cost of developing mission support software while increasing operations automation using expert system technology. GenSAA is well suited to support monitoring, fault detection, and fault isolation for spacecraft missions. GenSAA is now being generalized to support other application domains, and is being enhanced to support both monitoring and commanding operations.

REFERENCES

1. Hughes, P.M. (1989, October). Integrating Expert Systems into an Operational Environment. The American Institute of Aeronautics and Astronautics Computing in Aerospace VII Conference, Monterey, California.

2. Hughes, P.M., & Luczak, E. (1991, October). GenSAA: Advancing Spacecraft Monitoring with Expert Systems. The American Institute of Aeronautics and Astronautics Computing in Aerospace VIII Conference, Baltimore, Maryland.

3. Hughes, P.M., & Luczak, E. (1991, May). The Generic Spacecraft Analyst Assistant (GenSAA): A Tool for Automating Spacecraft Monitoring with Expert Systems. 1991 Goddard Conference on Space Applications of Artificial Intelligence, Greenbelt, Maryland.

4. Hughes, P.M., Shirah, G. & Luczak, E. (1993, October). Advancing Satellite Operations with Intelligent Graphical Monitoring Systems. The American Institute of Aeronautics and Astronautics in Aerospace IX Conference, San Diego, CA.

5. Luczak, E.C., et. al. (1994, February). User's Guide for the Generic Spacecraft Analyst Assistant (GenSAA) for TPOCC, Release 2.0. NASA/GSFC Code 520 DSTL publication 93-014. Greenbelt, Maryland.

6. Luczak, E.C., et. al. (1994, March). Operations Concept for the SAMPEX Pass Automation Demonstration. NASA/GSFC Code 520 DSTL publication 94-003. Greenbelt, Maryland.

111121

N95-17536

353996 p. 5

Development and Use of an Operational Procedure Information System (OPIS) for Future Space Missions

by

N.Illmer[1] , L.Mies[1] , A.Schön[1] , A.Jain[2]
1.Deutsche Forschungsanstalt für Luft- und Raumfahrt e.V. (DLR)
D-51140 Köln, Germany,

2. WIB GmbH, Lassenstraße 11-15, D-14193 Berlin, Germany

Abstract

A MS-Windows based electronic procedure system, called OPIS (Operation Procedure Information System), was developed. The system consists of two parts, the editor, for "writting" the produre, and the notepad application, for the usage of the procedures by the crew during training and flight. The system is based on standardised, structured procedure format and language. It allows the embedding of sketches, photos, animated graphics and videosequences and the access to offnominal procedures by linkage to an appropriate database. The system facilitates the work with procedures of different degrees of detail, depending on the training status of the crew. The development of an "language modul" for the automatic translation of the procedures, for example into Russian, is planned.

Introduction

The scientific output of a manned space mission is highly dependent on the correct execution of an experiment according to instructions called "procedures" the astronaut has to follow. The procedures of today (at least for spacelab missions) are very explicit paper versions and require hours of crew time just to read. For the future, especially for long-duration missions, the possibilities of modern computers and text processing should be used to improve the procedure standard allowing for the transition to the use of electronic procedures on board. OPIS, a development of DLR in cooperation with WIB, is a step in this direction.

For the European mission Euromir 94, it is planned to use OPIS, installed on the portable Crew Support Computer (which is an `IBM Thinkpad`), as the prime tool for the performance of one material science experiment. The post-flight evaluation of its practicallity will be a milestone for it´s further development (e.g. prime tool for procedures on Euromir ´95).

Approach

The source that safeguards the experiment success in current SpaceLab missions is called the Payload Flight Data File, a complement of books containing the crew work schedule,

procedures and reference documents. A similar set of documents exists for the use on Russian MIR missions. Some shortcommings are associated with this type of flight documentation:

- *large volume and high weight of files*
- *time consuming implementation of paper uplinks into the documents*
- *long procedures in checklist format tend to tire out crewmembers finally leading to mistakes*
- *embedding of graphics, sketches etc. is difficult*
- *usage of animated graphic sequences or videoclips within the procedure, or the linkage to a database is impossible*

Our idea was to firstly develop a procedure format better suited for the work on a computer than the checklist format in use by NASA[1], thereby reducing the training effort (as necessary for long term missions and space station operation) and minimizing mistakes in the experiment performance. Secondly the crewmember should get a tool that facilitates access to support and reference information (e.g. malfunction procedures, photos, videos, etc.).
On the basis of the evaluation of the Payload Flight Data File of the German Spacelab missions D1 and parts of the D-2 mission, crew activites were analized. The categories of the typical crew activites are displayed in figure 1 below.

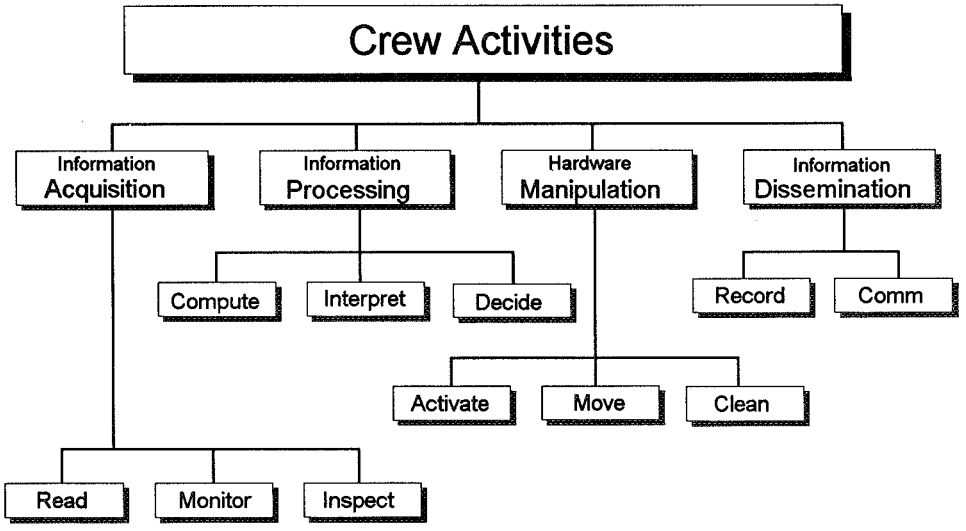


Figure 1: Classification of Crew Activites

These investigations were used to develop a new format for procedure instructions that is better suited for the use on a PC than the checklist format which is used at present. The format is build on procedure elements which describe the single task. A procedure element consists of seven defined positions as shown in figure 2. The last position leads to additional information concerning the performed step using a short code form. The OPIS standard was published in 1993 and presented to the German Space Agency (DARA) in the final presentation of the TOREX[2] study.

Step	Location	Object	Activity	Status	Info Code
1	CSK-1	display	✓ CHECK	address '66'	CO1

Figure 2: Example of a Procedure Element

The development of the OPIS software started in 1993. DLR provides the software requirements and WIB develops the software under contract by DLR. OPIS uses the WINDOWS environment and consists of two modules the one being the Editor for procedure generation, the other the so called Notepad-version is designed for the use by an astronaut.

The OPIS Editor

The OPIS Editor allows you to generate procedures in a standardised format by use of a structured language. This language has been constructed to describe tasks in a simple and unique manner. The editor would perform all tasks for the procedure layout automatically and offer all information for procedure generation on call, that has been by another experiment before. All procedure elements (locations, activties, objects, etc.) are stored in a database. All activity keywords are linked to appropriate icons. Complex procedure structures (for exeample "REPEAT...UNTIL" or "IF...THEN") can be generated in a simple way via implemented editor commands. A procedure syntax check via an syntax checker within the editor is foreseen for the future. Any sequence within a procedure can be defined as a standard module and can then be handled like a single activity (or command). You can have various standard modules in one procedure. In that way procedures that contain activities, that have to be repeated some times, can be simplified. The embedding of graphics, videosequences, offnominal procedures can be realised via linkage to an appropriate database. For the future the development of an language modul for the automatic translation into Russian is foreseen.

The OPIS Notepad

The layout as shown in figure 3 is designed to give the astronaut a clear picture of the steps he has to perform and the ones he already has performed. In the left icon bar the main file functions can be quickly accessed (the numbers 1 to 8 can be used to quickly open specified files). In the procedure window a highlighted bar shows the current step the astronaut is working on. When work on the procedure element is finished it can be tagged with the 'Enter'-key. In this way the system time and the line number will be entered into the 'Report File' wich is an ASCII-File containing all the information of the timely execution of the experiment. There is also a possibility for the crew to write notes and enter data into the procedure, which will also be transfered into the Report File. In that sense the original procedure can be used for different runs and the Report File will include all experiment specific infos for evaluation on ground.
Additional useful information is displayed in the status line at the bottom. The actual page and line number can be seen as well as the current time, the elapsed time since the procedure was called up, and a countdown that can be started if waiting periods are included in the procedure.

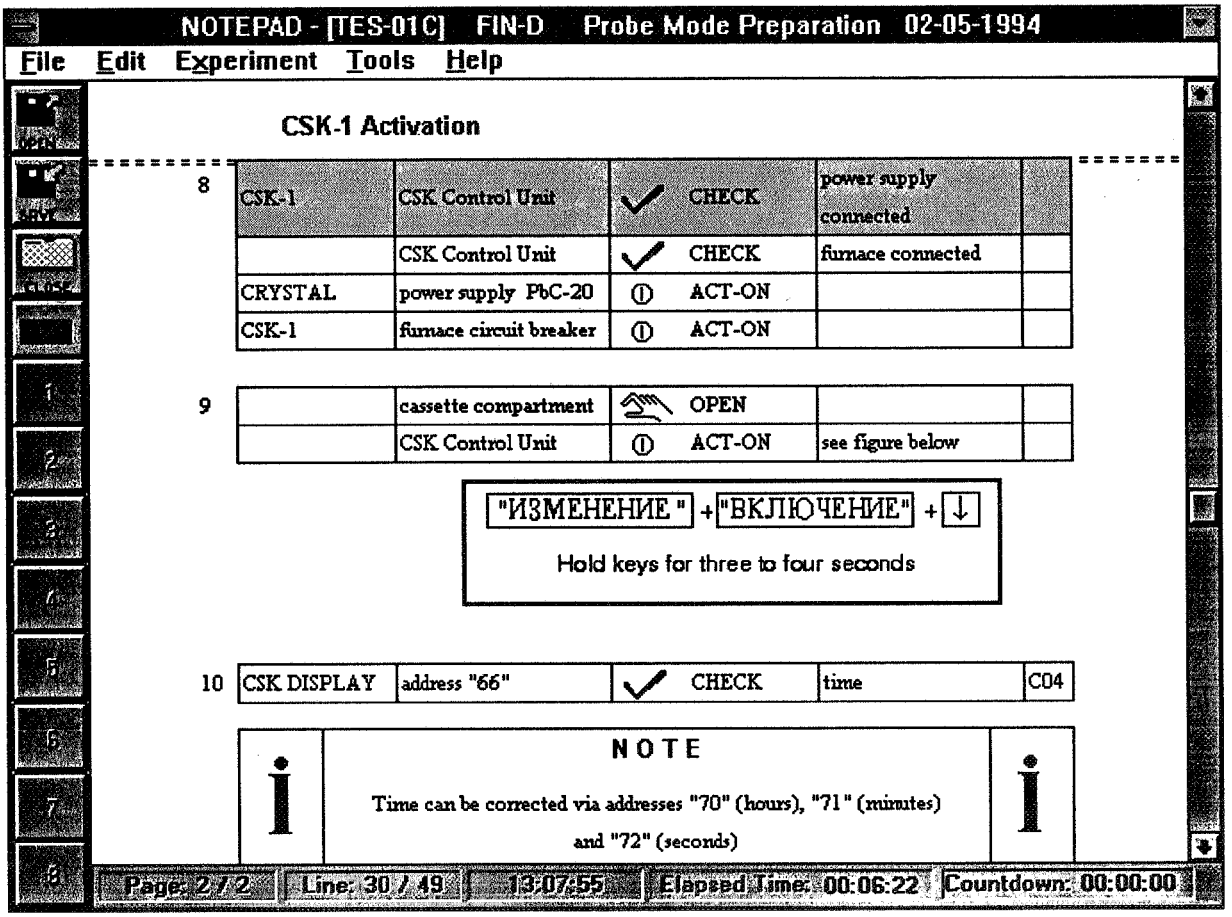


Figure 3: The OPIS Notepad environment

There are off-nominal situations and very complex procedures requiring additional information to safely perform the task. OPIS approaches this problem by establishing an ***interface to a data base*** containing photos, video clips and instructions to solve the problem. The data base currently in use for the TES-Experiment (material science) on Euromir '94 was developed by BSO under contract from ESA/ESTEC. The data base information can be accessed via a mouse doubleclick into the info-code box of OPIS.
As a paper backup or for selfstudying etc. the procedure can be printed from the editor or the notepad with an layout identically to the layout on screen. But the computer related topics (and that means most of the advantages of OPIS) will be lost.

Outlook and Conclusion

Main topics under consideration at present are:

- Needs of individual crewmembers for information vary by a wide margin (e.g. for medical experiments) -> ***'personalized' procedure desirable***

- Long-term missions require procedure systems that are capable of frequently providing updates of information without producing huge piles of paper
  -> file uplink (and downlink)

- Cooperation with Russia requires translation of procedures
  -> language module

With the development of an operational procedure information system we try to take into account the advantages of modern PCs. Our hope is that the ideas behind our system can help to improve the operations on board a manned space station even if OPIS is not the tool to be used then. We appreciate every comment to our paper and would be glad to demonstrate the software to interested parties.

References

[1] Crew Procedure Management Plan (JSC-08969), NASA Flight Activity Branch, Operations Division, 1991

[2] Abschlußpräsentation Technische und Operationelle Standard-Rahmenbedingungen für Experimente unter Schwerelosigkeit (TOREX) WIB GmbH, 23.September 1993. (Technical and Operational Standard-boundary conditions for Experiments under Microgravity)

111122

N95-17537

354002

P.7

SCOSII: ESA'S NEW GENERATION OF MISSION CONTROL SYSTEMS THE USER'S PERSPECTIVE

P.Kaufeler and M.Pecchioli, Mission Operations Department (MOD), European Space Operations Centre (ESOC), Darmstadt, Germany
I.Shurmer,VEGA Group PLC, Harpenden, UK, (MOD, ESOC, Darmstadt, Germany)

Abstract - In 1974 ESOC decided to develop a reusable Mission Control System infrastructure for ESA's missions operated under its responsibility. This triggered a long and successful product development line, which started with the Multi Mission Support System (MSSS) which entered in service in 1977 and is still being used today by the MARECS and ECS missions; it was followed in 1989 by a second generation of systems known as SCOS-I, which was/is used by the Hipparcos, ERS-1 and EURECA missions and will continue to support all future ESOC controlled missions until approximately 1995. In the meantime the increasing complexity of future missions together with the emergence of new hardware and software technologies have led ESOC to go for the development of a third generation of control systems, SCOSII, which will support their future missions up to at least the middle of the next decade. The objective of the paper is to present the characteristics of the SCOSII system from the perspective of the mission control team; i.e. it will concentrate on the improvements and advances in the performance, functionality and work efficiency of the system.

1. INTRODUCTION

The concepts and functionality of the Mission Control Systems (MCS) which are currently in use in ESOC, i.e. MSSS and SCOS-I, are mainly originating from the mission control requirements of the 1970's which were based on the hardwired spacecraft technology which was the standard at this time. The arrival of a new generation of more complex spacecraft with significant amount of on-board software and increased on-board autonomy, such as EURECA or ERS1, placed much more demanding requirements in terms of functionality and performance on the MCS which, although they could be accommodated (sometimes requiring development of mission specific adds-on), revealed the limits of these systems. Therefore the decision for the development of a new generation
of MCS, SCOSII, was taken, with the following main objectives:

- reduce mission adaptation/maintenance costs,
- improve efficiency of mission preparation, execution and evaluation tasks,
- increase operational quality and reliability,
- have a life time of at least 10 years,
- cope with a wide range of different mission type/size/complexity.

which led to the following major design requirements:

- SCOSII must be a full scope generic system.
- It must be a modular and open system, being adaptable and expandable in size, performance and functionality.
- It must operate in a basic hardware and software environment that is vendor independent.
- It must be based on state-of-the-art software technology.
- It must be compatible with the new standards in the space domain such as in particular the CCSDS and related ESA standards for telemetry and telecommand packets, and the standards and guidelines of the ESA Committee for Operations and EGSE Standards (COES).

2. SYSTEM CHARACTERISTICS AND CONCEPTS

The SCOSII system has been conceived as a generic infrastructure platform, providing an exhaustive set of standard functionality constituting the basis for the development of mission dedicated MCSs. As such, a particular instance of a SCOSII based MCS will not offer multi-mission support, but will be able to cope with multi-satellite missions, thus supporting simultaneous control of several satellites of the same family.

2.1 Architecture

The high flexibility and performance requirements placed on SCOSII led to the choice of a decentralized architecture, consisting of a network of Unix workstations in a *'Client-Server'* configuration. Each operational user will interface to the system through a dedicated workstation providing local processing power to cope with the user-interface processing load, and local storage for e.g. hosting of the most recent historical data, while a set of system level services (e.g. interfacing to the ground stations) ensuring overall coordination will be provided by central server processors. The use of such a distributed system will allow the computing power to be tuned to the demand of a particular mission and will also offer advantages in terms of system availability and failure tolerance. A more detailed description of the architecture of SCOSII can be found in References [2] and [5].

2.2 Overview of Functionality & Utilisation

SCOSII is intended to cover the following functions and services:

- ***Mission Planning***, including acceptance, checking and pre-processing of various types of planning requests, generation of a conflict-free *'Plan'*, and derivation of an executable *'Schedule'*.
- ***Monitoring & Commanding (M&C)***, of the spacecraft, the mission support services provided by the ground network (e.g. telemetry and telecommand services of the TT&C stations) and SCOSII itself (i.e. control of user configurable functions). This means that e.g. the same generic M&C functions (e.g. status monitoring, commanding, procedures execution) can be used to handle the spacecraft, ground station services and on-line SCOSII configuration.
- ***Historical Data & Performance Evaluation***, consisting of the storage of all mission data in an on-line manner, the ability to access these data for direct visualisation and/or subsequent processing using powerful data analysis and presentation tools, and the production of corresponding reports.
- ***Mission Database Handling***, consisting of the generation and

PRECEDING PAGE BLANK NOT FILMED

PAGE 718 INTENTIONALLY BLANK

maintenance of all the static mission data used, to configure the system for a given mission (e.g. user privileges, display lay-out), and to define the characteristics of the mission (e.g. TMTC processing data, operations procedures, etc...).
-*On-board Software Maintenance*, consisting of the tools to monitor, and modify the content of the on-board memories (i.e. memory images).
- *System Level Tools & Services*, such as state-of-the-art Human Computer Interface (HCI) techniques, user access control mechanism, advanced help facility, etc...

The wide range of functionality provided by the system, and its flexibility and adaptability, will allow SCOSII to be tailored to cover, for a given mission, different *'Roles'*, each being carried out by a specific instance of a SCOSII system. This will of course include its main role of *'Prime'* MCS which will incorporate the full set of functionality required to support the mission, but SCOSII may also be used as *'Mini-backup'* spacecraft M&C system to be located at e.g. a TT&C ground station. Furthermore, the fact that SCOSII is being designed in accordance with the standards and guidelines of COES, will ensure not only its full compatibility with checkout systems, but would allow SCOSII to be used, with minor adaptations, as a checkout system as such.

Having outlined the functionality and roles of the system, we will now address the various user scenarios which SCOSII will have to support. Here again, SCOSII constitutes a major step forward with respect to its predecessors which were only providing very restrictive and rigid centralised user access, in that it will also support various types of remote access scenarios as described below and illustrated in figure 1.

- ***Office based users***, for mission preparation and/or evaluation activities.
- ***Home-based users***, for on-call contingency support.
- ***Engineering support users***, such as spacecraft manufacturer, for anomaly investigation, mission evaluation.
- ***User Operations Control Centres*** (USOC), for the control of given payload(s) on a spacecraft.

2.3 Configurability

Since SCOSII will constitute the basic MCS kernel for a wide range of missions of different type and complexity, the system will have to be highly configurable. One important aspect in this context, is the capability of SCOSII to be descoped, adapting its functionality and hardware to the needs of the mission. For a simple mission, a mini-system running on a single SUN workstation, could be used. Moreover, its portability will allow such a mini-system to run on a PC.

Another issue related to configurability is that the system must be, as much as possible, data-base driven, maximizing the tailoring capabilities and minimizing the need for software modifications. For predecessor systems, this approach was limited to the spacecraft TM/TC processing characteristics, which were fully defined in the spacecraft characteristics database. For SCOSII this concept has been expanded to all functional subsystems, including

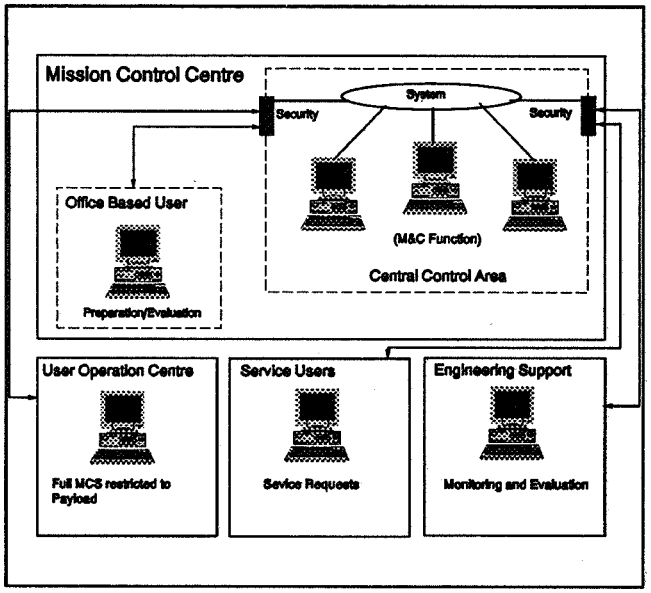


Figure 1: SCOSII User Scenario

data driving the system configuration, thereby providing the user with the capability of defining through the Mission Database the haracteristics of major elements of the system such as:

- HCI layout (e.g. layout of input forms or of displays templates),
- defaults for most of the functions (e.g. which packets are to undergo which types of checks),
- definition of standard named sets of user privileges.

The SCOSII system will therefore consist of a set of generic functions plus a generic default configuration, which can be modified by the user to suit the needs of his mission.

2.4 Performance

As SCOSII is intended to be the basis for MCSs for at least the next ten years, very ambitious performance goals have been adopted. These include concurrent real-time telemetry and telecommand rates of 2Mbps and 4Kbps respectively, display update rates exceeding 10 per second, very short response times to user requests - e.g. from 5 sec for retrieval of data not older than a few weeks to 30 sec for data being several years old -, the above requirements being applicable to utilisation scenario involving up to 50 workstations used simultaneously.

2.5 System Level Tools

In support of its main functions as described above, SCOSII will provide a set of very powerful system level services and tools, the most significant of which are presented below.

2.5.1 Modelling Tool

Previous control systems were based on a low-level view of the spacecraft in that they only considered its telemetry and telecommand components, and thus did not include any information about their link to the higher level components of the spacecraft such as the devices/units, subsystems, etc..., and their interrelationships. This approach was sufficient to handle relatively

simple missions, but was not adequate for introducing more advanced functionality and user interfaces which require a more structured and intuitive view of the mission/spacecraft.

A fundamentally different approach was followed for SCOSII. In the SCOSII database the mission will be described as a hierarchical structure of components of operational significance. This is achieved by defining a decomposition following the object-oriented *'whole-part'* relationship, starting with the mission as the highest level component, down to the devices/units hosting the individual measurements and command items at the lowest level.

In addition to this decomposition into what are called *'System Elements'* in the SCOSII jargon, it will be possible to associate with them synthetic information, called *'Operational Modes'* and *'Roles'*. The former represents particular states of operational significance as derived from the state of their constituted parts, while the latter corresponds to their function(s) within their respective mission domain. This will provide a first step towards an advanced modelling capability; initially modelling will be restricted to data routing, power control and redundancy but this will be further extended in future releases of the system to include the full set of standard functions and behaviours of the typical mission components.

Moreover, SCOSII will also provide a library of *'System Elements'* which could be used as building blocks. In order to define e.g. the battery 1 component of mission X, the user would chose the standard battery building block in the SCOSII library; he would, if required, modify it to correspond to the characteristics of the batteries of mission X by specifying its difference to the standard SCOSII battery, and instantiate it to become battery 1 by specifying the links to its constituent telemetry and telecommand items. These modelling capabilities which are illustrated in Figure 2 below and further expanded in Reference [5], will provide significant improvements in the following domains:

- ***Mission Database Definition:*** increased efficiency and quality/consistency, by reducing the information to be specified by the user to a strict minimum and by providing him with a more intuitive view of its mission.
- ***System Configurability and Controllability***: by allowing the user/system to exercise this at mission component level (for navigation through mimic display, to disable functional checks for only a particular mission component, to allocate/restrict functional privileges to e.g. a particular spacecraft subsystem, etc...).
- ***Mission Execution:*** by making use of the modelling data (in particular the *'Roles'*) to predict the status of the mission, thereby supporting the mission planning and commanding functions in assessing the effect of future commands (e.g. to ascertain their safety/feasibility), and the monitoring function by generating the expected mission status as reference for comparison against the status obtained from telemetry. This *'Prediction'* function is a new feature, making use of the *'Mission Model'* to obtain the best estimate of the mission status at any time in the future, based on an initial state and on the knowledge of any planned activities and any foreseen on-board events and actions.

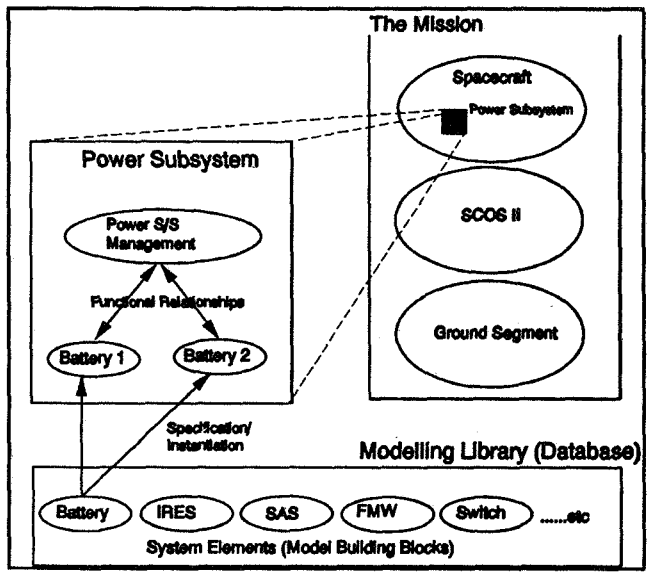


Figure 2: SCOSII Modelling

2.5.2 Operations Language

To allow an efficient definition and maintenance of the mission specific knowledge in the Mission Database, a dedicated SCOSII Operations Language (OL) is required. The OL has been designed to provide users without software design expertise, with a set of mini languages offering the necessary expressive capability to define the knowledge for the more advanced SCOSII functions, such as:

- ***Procedural knowledge***, for the presentation of, navigation through and automatic/semi-automatic execution of procedures to e.g. control the spacecraft, ground station services and SCOSII configuration,
- ***Events & Actions***, to identify from the incoming data, user defined events to be logged and the corresponding actions to be initiated by the system (e.g. an event could be a particular spacecraft anomaly which would initiate a specific set of recovery and diagnosis actions),
- ***Selection Strategies***, to provide the various data selection capabilities that will be required by the user and/or system in support of the different activities/applications (e.g. selection strategies could be applied to restrict a particular function to a subset of the data it would normally be applied to).

Further details about the SCOSII OL can be found in Reference [4].

2.5.3 Mission Database Test Function

This is another new functionality, which will provide an on-line mission database checking capability, using as data sources either real-time or historical telemetry, or data generated by the *'Mission Model'* being driven by a predefined sequence of commands. This local test function will allow to significantly reduce the turn around time for database changes, and to alleviate the need for the lengthy and resource-intensive validation using an external software simulator.

2.6 Human Computer Interface (HCI)

The SCOS II HCI will provide users of all levels of experience, with an intuitive, but reliable and robust interface. The SCOSII HCI will be based on WIMP (Windows, Icon, Mouse and Pull-down menus) technology. SCOS II will support all the traditional display types (e.g. alphanumeric, graphic and mimics displays), however, the users will be given tools which will allow them to combine these different data display techniques to display data in a more flexible and efficient manner. Due to the increase in the diversity and versatility of the HCI with respect to previous systems, particular attention has been paid to the specification of general guidelines concerning display and data representation techniques in order to provide the user with a consistent HCI across all applications.

3. MISSION DATABASE

The scope of the SCOS II Mission Database is much wider than that of the earlier systems, which generally concentrated upon the data required by the Monitoring and Control functions. In addition to the latter, a SCOS II database will contain, e.g. the mission model data, the mission planning/scheduling data, the on-board software memory images, the operations procedures and the Spacecraft Users Manual (SUM), and will also include the system set-up and configuration data (e.g. definition of user privileges).

3.1 Mission Database Structure

The Mission Database will consist of a hierarchical collection of database parts, each with a unique identifier and version number, arranged in a user defined structure (Figure 3). The higher level parts are used purely for organisational purposes, the lowest level parts contain the data and constitute the lowest level entities submitted to version control. For a given mission, the user will have some flexibility of configuring the structure to its particular needs.

3.2 Database Management

There will be three types of Mission Database.

- ***The Operational Database***: A database which is, or has been judged so in the past, capable of supporting real-time operations. SCOS II will support a number of Operational Database versions.
- ***The Active Database***: The Operational Database which currently supports real-time operations; any of the Operational Databases may be selected as the Active Database.
- ***The Draft Database***: A database used as an intermediate step to constructing a new Operational Database. There will be only one Draft Database.

It can be imagined that all the databases are kept within a *'Database Area'* and accessed via the users from a *'Working Area'*. The *'Working Area'* contains a number of user accounts, i.e. *'User orking Areas'*, which will allow multiple user database maintenance. Special mechanisms will be provided in order to ensure this multiple user maintenance is done in an orderly manner; e.g. each user working area will be completely isolated and the system will prevent several users from being able to work on the same database part simultaneously. The database manager will be able to select modified database parts and to integrate them

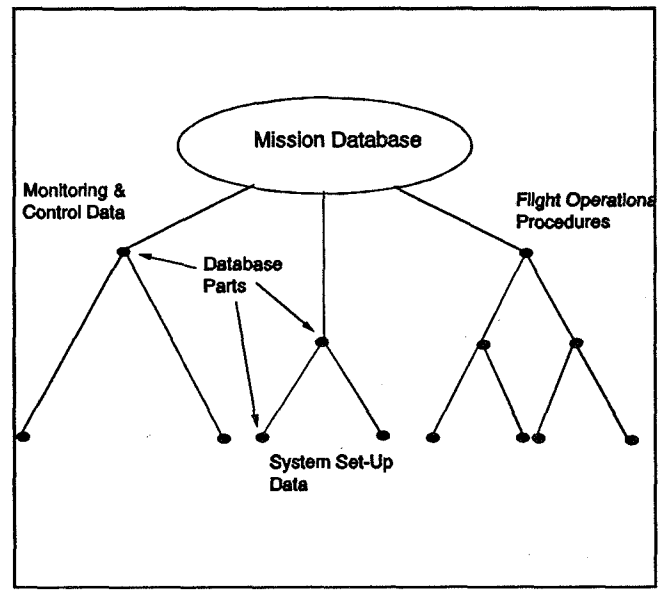


Figure 3: Mission Database Structure

back into an Operational Database, either directly (for on-line changes) or via the draft database (for changes of a higher magnitude). Subsequently, this database can be selected as the Active Database.

Version Control functionality will automatically maintain the version of the Operational or Draft Databases and their constituent parts. In addition Change Control functionality will permit exhaustive recording of all database changes at item level and at the higher levels of the database hierarchy. The SCOSII database management concept, as described above, is illustrated in Figure 4, below.

3.3 Database Maintenance

Mission Databases are mainly constructed from input data that are provided from spacecraft/payload(s) manufacturer(s) or from checkout. Since the data volume may be extremely high (typically several thousands of parameters, just for telemetry) these data are to be provided in an electronic form. SCOSII will be able to import these source databases in various electronic formats (e.g. ORACLE, ASCII), to integrate the contained data items into the SCOSII internal database, and to subsequently handle new versions of the source database (e.g. functions to compare a new source version with previous ones or SCOSII versions).

In addition to the acquisition of the source database, SCOSII will provide the editing capabilities required to handle the data that have been acquired from the source database (for this dedicated functions will be provided to facilitate large scale editing) and to subsequently maintain the data. The data maintenance functions will of course include exhaustive but flexible data consistency checking functionality. Consistency checking will be performed at all levels (e.g. data item, database part and database), however, the user will be able to switch the checking off, an essential feature for the preparation phase, when inconsistencies cannot be avoided.

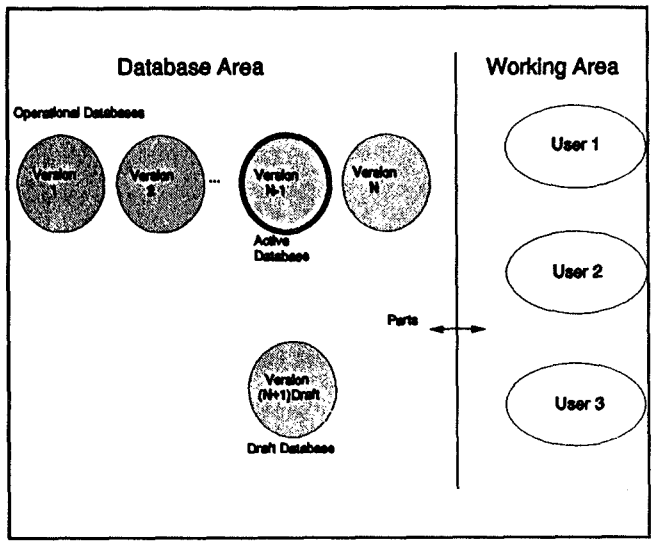


Figure 4: Database Management Environment

4. MONITORING

The following monitoring tools will be provided.

4.1 Monitoring Parameters

Unlike previous systems, there will be several potential sources of monitoring parameters in addition to those that come from the spacecraft, e.g. SCOS II parameters and Ground Segment parameters. Regardless of source, all parameters will be processed by SCOS II in the same manner.

SCOSII will be far more flexible and versatile than previous systems. The users will be given the facilities to view the monitoring parameters in a number of different ways called *'Representations'*. The user will be able to select, in real-time, which *'Representation'* is to be displayed. In particular, SCOS II will support:

- ***The Raw representation:*** the uncalibrated view of the parameter value.
- ***The Engineering representation:*** the calibrated view of the parameter value.
- ***The Functional representation:*** obtained by applying a function to the parameter eg. derivative, integral, mean, max.
- ***The Status representation:*** returning the state of the currently applicable *'Checking'*, eg. nominal (see section 4.3 below).

In addition, SCOS II will support the display of values in different formats (e.g. raw representation in hex, decimal or binary) and the on-line conversion between engineering units.

4.2 Parameter Validity

SCOS will support the concept of monitoring parameter validity, since for a number of possible reasons, the latest parameter value could be meaningless or unreliable. The following factors influencing the validity have been identified.

- ***Power:*** the status of the power supply the parameter is dependent upon.
- ***Data Unit:*** the quality of the data unit within which the parameter was transported.
- ***Data Routing:*** the status of the transmission route taken by the data.
- ***Age:*** the age of the parameter value.
- ***Stability:*** the parameter value may be in a transient state due to commanding activity.
- ***Status:*** any other explicit criteria the user wishes to specify.

SCOS II will check all these factors when assessing a parameter's validity. The resultant validity state of a parameter will be automatically propagated throughout the system affecting other processing where relevant (e.g. synthetic parameters will also be flagged as invalid if they use an invalid parameter) and affecting how the data is displayed to the user.

The user will be able to gain real-time access to the results of each validity component check. Hence, the SCOSII user will be provided with significantly improved validity checking facilities and, when a parameter is flagged as invalid, considerably more information about the reason why.

4.3 Parameter Checking

The objective of checking is twofold. On one hand the system must be able to check whether the operator has not or is not going to place the mission elements under its responsibility (e.g. the spacecraft) in a non-nominal or unsafe state, on the other hand, the system must be able to detect whether these elements are behaving as expected. This led to the following categories of parameter checking being provided by SCOSII.

- ***Operational Status Checks:*** Monitor the on-board status which is required regardless of any commanding activities, to ensure that the spacecraft is left in the correct state after a series of operations.
- ***Operational Constraints Checks:*** Are of the same nature as Operational Status Checks, but stronger. They are rules which should never be violated operationally and as such, should never be disabled. They will contribute to *'Activity' 'Pre-Execution Validation'* (see section 5.3 below).
- ***Behaviour Checks:*** Are based upon the prediction of the on-board status, taking into account the effects of commanding and of predicted events. The checking performed is to ensure that any behaviour exhibited (e.g. change of state after a command) is as expected.

5. COMMANDING

An overview of the envisaged full SCOSII Commanding functionality is given in Figure 5.

5.1 Activities

In order to control the mission, a SCOSII user will be able to initiate the execution of *'Activities'*, where an *'Activity'* is either a Procedure (highest level), a Command Sequence (simplified procedure syntax), or a Command (lowest level). SCOSII treats each of these in exactly the same manner. Each can have execution pre-requisites, each can be monitored through its execution phase and each can be verified. Activities will be initiated manually, or automatically by the system. The long term aim of SCOSII is to have a fully automated Procedure execution capability.

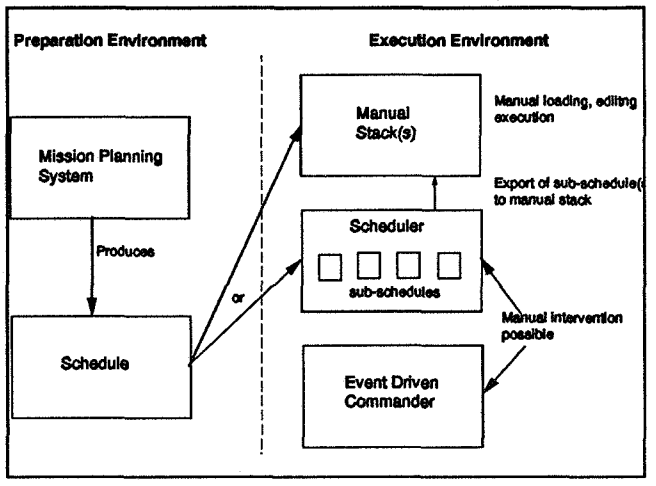


Figure 5: Commanding Overview

5.2 Preparation

This consists in the production of the schedule of activities corresponding to a given time increment, for later submission to the activity execution function. While this will be initially done manually, it will be carried-out, in later releases of the system, by a generic Mission Planning functionality, which will include:

- ***Processing of Planning Requests***: This covers the acceptance, checking and pre-processing of planning requests received from external entities e.g. experimenters, external control centres, flight dynamics.
- ***Planning/Scheduling Function***: This covers all activities required to generate a conflict-free *'Plan'* and its corresponding *'Schedule'* of activities from the pre-processed planning requests.

5.3 Activity Execution

It will be possible for the user to execute operational *'Activities'* by means of three facilities:

- ***The Scheduler***: Pre-prepared *'Schedules'* will be imported from the preparation environment into the *'Scheduler'*. If necessary, the user will also be able to split this imported *'Schedule'* into a number of logical partitions called *'Sub-Schedules'*. Each *'Sub-Schedules'* of executable activities could then be assigned to a different user and/or to a different type of operations (e.g. one sub-schedule could be dedicated to Payload-X), thus delegating execution control. Nominally the *'Scheduler'* will manage the execution of activities automatically, taking into account execution pre-requisites and links between activities, prompting for manual input when required. However the user will always retain the capability of regaining, if required, control over the *'Scheduler'*.
- ***The Manual Stack:*** The traditional commanding facility, allowing the user to directly control the release of Activities will, of course, also be supported by SCOSII in order to provide the user with fully manual execution capability in the eventuality of critical and/or unplanned operations.
- ***The Event Driven Commander:*** This is a new SCOSII concept that will give the users the capability of setting up event-action relationships as Event Driven Commanding Routines (EDCRs). EDCRs can periodically monitor for the occurrence of an event that will trigger the execution of a specified set of activities, e.g. can be used for automatic closed loop reaction to on-board anomalies.

All executable activities will have pre-requisites which must be satisfied before they can be released from the SCOSII system. In SCOSII, these are called *'Pre-Execution Validation'* (PEV) checks. These will have three components:

- ***Feasibility Checks***: Checking that all necessary resources are available, e.g a transmission route.
- ***Safety Checks:*** eg. Checking that Unit A if OFF before switching Unit B ON.
- ***Dynamic Checks:*** Checks which are not related to the activity in isolation, but to the external context of the execution of a specific instance of an activity, e.g. time window and interlock checks.

Activities will then be released by SCOSII when authorised by their respective PEV, based on a *'Release Strategy'* specified at preparation. SCOSII will support both manual and automated release strategy such as "initiate execution X minutes after event Y".

5.4 Activity Execution Monitoring

To enable the user to be aware of the transmission and execution status of any activity that has been released from SCOSII, dedicated verification checks will be performed. For command execution verification, the users will explicitly define verification criteria, using the Operations Language, in the Mission Database. Though, there will be the capability of doing the same for command sequences and procedures, the majority of their checks will be implicitly defined by the checks defined for each command they contain. SCOSII will support the explicit definition of simple or complex multi-staged verification criteria, the latter for those commands which are executed in a number of stages (eg. reception on-board, reception by application, execution stage 1, execution stage 2). For each identified verification stage, SCOSII will automatically compute a verification window based on expected execution times and/or user defined margins/delays.

6. CONTROL OF SCOSII SYSTEM

The M&C functionality will be controllable flexibly. This is particularly important in the case of contingency situation where the normal conditions of applicability of a function may not be valid any more; past systems have been rather rigid in this respect. During operations, the user will have the capability to control the way the functions are applied and to which data they are applied; e.g. one will be able to completely or partly disable parameter validity checks. Standard parameter checking, as described in 4.3 above, will be applicable to the status of the controlled functions, to ensure that they are not left in a non-nominal/undesirable state.

7. MISSION EVALUATION

Sophisticated tools will enable the users to access historical data and then view, analyze them and to produce reports. This functionality will be an integral part of SCOSII, and unlike on previous systems, will be available on-line. The following functions will be provided.

7.1 Historical Data Access

The user will be able to access data and if required to save them for later re-use, either for direct presentation using the standard displays used for real-time monitoring, or for submission to further processing (e.g. detailed analysis). Data access will be supported by a powerful syntax, allowing the user to define expressions, called *'Data Access Strategies'*, which he could save for later re-use, and capable of specifying:

- A time window or multiple time windows.
- References to events eg. 'entry into eclipse'
- Expressions eg.'when A123 > 35 degrees C'
- Data access criteria e.g. all AOCS telemetry

7.2 Viewing Historical Data

SCOSII will provide the user with two viewing modes, *'Replay as Live'* and *'Video Replay'*. *'Replay as Live'* will be dedicated to the technical analysis of the mission data, i.e. it will allow the user to replay historical data and to interact with them as if they were being generated in live, while *'Video Replay'* will be dedicated to operational investigation, i.e. it will allow a user to be confronted with the same data and workstation lay-out as at the time of reception of the data.

In both cases, the user will have complete control over the replay, controlling its start time, the number of workstations it appears on and its speed and direction (eg. fast forward, forward, pause, rewind, fast rewind etc...).

7.3 Historical Data Analysis

The users will be provided with a data analysis package which will have the following functionality at a minimum:

- ***Data Manipulation***, allowing the user to select a subset of the retrieved data for analysis.
- ***Mathematical functions***, e.g sin, cos, tan, log, differentiation, integration.
- ***Statistical Analysis functions***, e.g. mean, standard deviation.
- ***Graphical tools***, allowing the user to produce 2- and 3-D graphs, straight line fits, polynomial fits, bar charts, pie charts etc...

7.4 Report Generation

A great amount of effort is expended by Operational Teams producing reports, many of which are of a routine nature. Therefore SCOSII will, unlike on previous systems, include a report generation function allowing production of text documents in which mission history data can be incorporated. This function will also support an automatic report generation facilities; a user will be able to define report templates, e.g. definition of the contents and structure of a report, and use these to automatically produce reports of data for a user defined time period.

8. CONCLUSION

SCOSII is a major step forward with respect to its predecessor systems, which will put ESA at the forefront of the technology and meet its main goals of minimising mission costs and improving mission preparation efficiency and operational performance.

This is the first time that a systematic and thorough effort has been invested in defining user requirements for a generic infrastructure (as opposed to individual mission systems). This has involved a close cooperation between the users and the developers of the system, and has included exploratory prototyping (as well as technology prototyping).

Release 1 of SCOSII is at an advanced stage of implementation, a preliminary delivery being expected in November 1994. Broadly speaking Release 1 is covering the same range of functionality as the previous infrastructure, with inclusion of the Commanding function (not available in SCOS-I) and with enhanced functionality and more modern human computer interfaces. More advanced functionality will be added in Release 2 (1995-6) and Release 3 (1996-7), including Modelling, Mission Planning, Data Distribution and certain of the more advanced database features. Consolidation of Release 1 functionality will also take place in the later releases. Such an incremental implementation has been chosen in order to minimise technical and schedule risks to the first client missions of the system, HUYGENS, ARTEMIS,and ENVISAT to be launched in 1997-1998.

References

1 Packet Utilisation Standard, ref. ESA PSS-07-101, issue 1, May 1994, to be published.

2 N.Head & J.F.Kaufeler, Evolution of the Agency's Software Infrastructure for Spacecraft and Mission Control, ESA Bulletin no.67, August 1991.

3 M.Jones et al., SCOSII: ESA's New Generation of Mission Control System, ESA Bulletin no.75, August 1993.

4 A.Baldi et al., SCOSII OL: A Dedicated Language for Mission Operations, SpaceOps 94.

5 M.Jones, N.Head et al., SCOSII: ESA's New Generation Control System, SpaceOps 94.

AN OBJECT MODEL FOR MULTI-MISSION COMMAND MANAGEMENT SYSTEM

Jon Kuntz
Loral

Paper Not Available

PRECEDING PAGE BLANK NOT FILMED

111123

N95-17538

354010

P-6

A SOFTWARE ARCHITECTURE FOR AUTOMATING OPERATIONS PROCESSES

KEVIN J. MILLER

Operation Engineering Lab
Jet Propulsion Laboratory, MS 301-345
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109-8099

ABSTRACT

The Operations Engineering Lab (OEL) at JPL has developed a software architecture based on an integrated toolkit approach for simplifying and automating mission operations tasks. The toolkit approach is based on building adaptable, reusable graphical tools that are integrated through a combination of libraries, scripts, and system-level user interface shells. The graphical interface shells are designed to integrate and visually guide a user through the complex steps in an operations process. They provide a user with an integrated system-level picture of an overall process, defining the required inputs and possible outputs through interactive on-screen graphics.

The OEL has developed the software for building these process-oriented graphical user interface (GUI) shells. The OEL Shell development system (OEL Shell) is an extension of JPL's Widget Creation Library (WCL). The OEL Shell system can be used to easily build user interfaces for running complex processes, applications with extensive command-line interfaces, and tool-integration tasks. The interface shells display a logical process flow using arrows and box graphics. They also allow a user to select which output products are desired and which input sources are needed, eliminating the need to know which program and its associated command-line parameters must be executed in each case. The shells have also proved valuable for use as operations training tools because of the OEL Shell hypertext help environment.

The OEL toolkit approach is guided by several principles, including the use of ASCII text file interfaces with a multimission format, Perl scripts for mission-specific adaptation code, and programs that include a simple command-line interface for batch mode processing. Projects can adapt the interface shells by simple changes to the resource configuration file. This approach has allowed the development of sophisticated, automated software systems that are easy, cheap, and fast to build.

This paper will discuss our toolkit approach and the OEL Shell interface builder in the context of a real operations process example. The paper will discuss the design and implementation of a Ulysses toolkit for generating the mission sequence of events. The Sequence of Events Generation (SEG) system provides an adaptable multimission toolkit for producing a time-ordered listing and timeline display of spacecraft commands, state changes, and required ground activities. The multimission SEG software is easily adapted and OEL Shell templates are built to meet different mission requirements. The SEG system was adapted in a unique way for the Ulysses mission since the spacecraft does all commanding in real time. The Ulysses SEG toolkit allows a user to interactively build commands on a timeline display in spacecraft event time and then the system automatically derives required ground events, builds a mission sequence of events listing, and outputs a space flight operations schedule.

INTRODUCTION

The Operations Engineering Lab (OEL) at JPL has developed a generic set of tools for Sequence of Events Generation (SEG) that have been adapted to many of the current

PRECEDING PAGE BLANK NOT FILMED

PAGE 728 INTENTIONALLY BLANK

flight projects. The toolkit includes what-you-see-is-what-you-get (WYSIWYG) editors for the Sequence of Events (SOE), Space Flight Operations Schedule (SFOS), and Deep Space Net Schedule (DSNS), a set of servers to enhance the Perl language which is used to generate the SEG products, and a user-configurable graphical user interface (GUI) to control the SEG process. All of the SEG interfaces are text files.

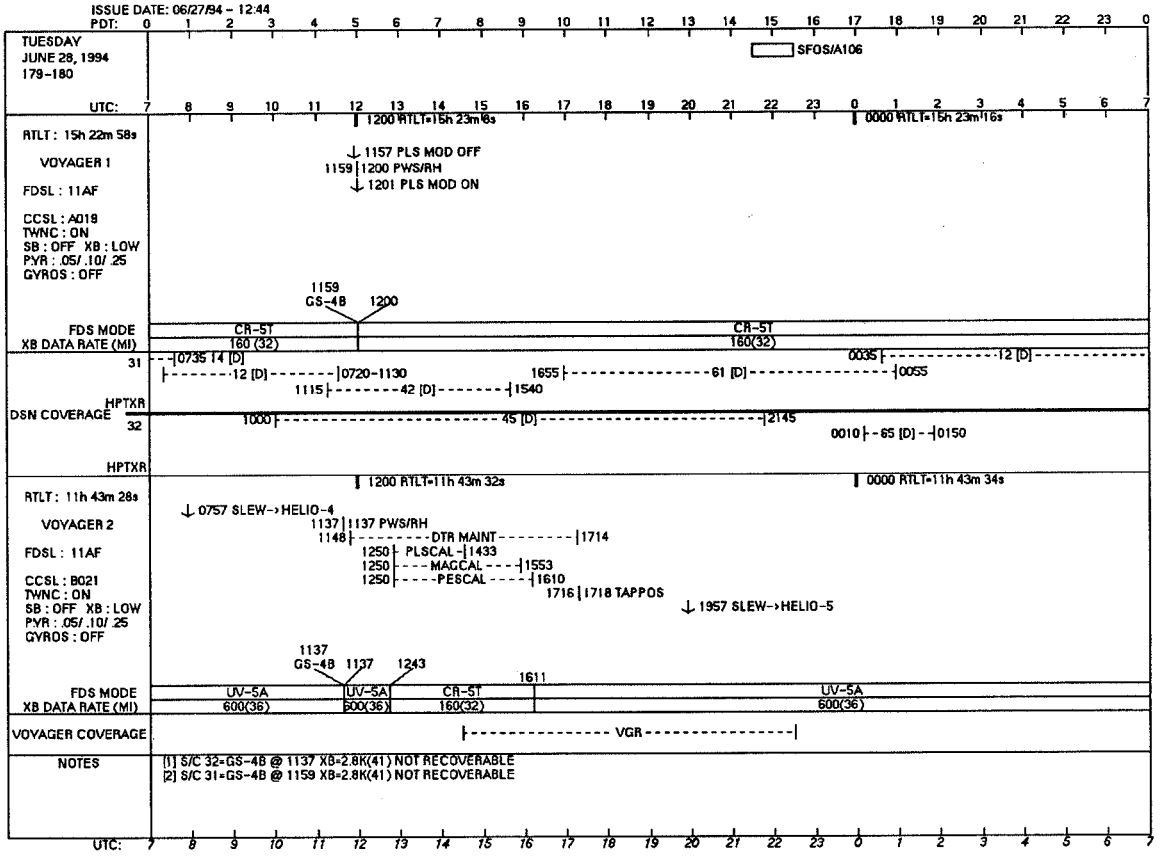


Figure 1. Voyager SFOS

The editors are generic object-oriented programs that display, edit, filter, and reformat the SEG products, but do not interpret the data. The editors are X Windows / UNIX programs written in C. The same editors are used by all projects. Rather than writing MSDOS or Macintosh versions of the editors, we export files that may be read with most MSDOS or Macintosh tools.

The SEG process for most missions is to take the spacecraft sequence file, the Deep Space Network (DSN) allocations and view periods files, and the light time file, and generate the SOE, SFOS, and DSN keyword files. Simply, SEG integrates the spacecraft and ground schedules in to a unit. The spacecraft sequence file is generated far in advance, does not include real time commands, and is often based on out-of-date DSN allocations. The SOE, SFOS, and DSN keywords files will contain more accurate ground information, and are used by the Mission Control Team and the Spacecraft Teams to schedule ground activities. In addition all SEG products use ground times for both ground and spacecraft events.

We chose to write our generating software in Perl since it is a very powerful interpreted language designed for processing text files. We also did not want to write a new language. Since the delivered executable is

also the source code, it is reasonably easy for the Mission Control Team to maintain the SEG adaptation. Perl has only two elementary data types: strings and floating point numbers, so additional servers were written in C to manipulate triggers, time-dependent state variables, time conversions, and spacecraft command string processing. The parent Perl script includes a Perl library that automatically starts up the server process and sets up a communications channel between the parent Perl script and the server similar to the Remote Procedure Call (RPC) mechanism. The server functions are then invoked with simple Perl function calls. It is possible to compile new functions directly into the Perl language, but the server model was chosen since it simplifies configuration management on the operations workstations, a new version of Perl may be installed without having to link in any SEG code, and in fact, the servers are not even tied to Perl.

The final component of the SEG toolkit is the OEL Shell. This is a user configurable GUI that lets the user gather input files, specify output files, and selectively run portions of the generating process and the SEG editors. OEL Shells have been built for several projects' SEG processes

THE OEL SHELL

The OEL Shell is a compiled program based on the X11 release 5 windowing system, the X toolkit (Xt), the Motif Widget set, and David Smyth's Widget Creation Library (Wcl) [1]. The intent was to provide a shell that would allow the user to enter UNIX commands with parameters from a simple Motif interface. The interface is configurable by the user by modifying the resource file. Several copies (which should in fact be links) of the compiled program may be available on the system. The appearance of these shells is determined by the program's name and its corresponding resource file. Since the user is encouraged to modify the resource file, and create one's own shells or enhancements to existing shells, some knowledge of Motif widgets and the resource database is prerequisite.

From a user's perspective, the OEL Shell consists of a series of push buttons, text entry areas, and toggle buttons arranged on a work area or control panel (one or more Motif drawing areas). Pressing one of the push buttons causes a UNIX program to execute. This program may be another Motif application or a script without a graphical user interface. The work area provides text entry areas for the user to enter command line arguments for the program. Toggle buttons correspond to UNIX command line options.

Below the panel is a scrolling message area which displays any output messages from the executing program or script. In addition, the actual UNIX command created from the push button, text, and toggle buttons may optionally be displayed here. If a text widget is used for file input, it will generally have a Select and an Edit push-button located nearby. The OEL Shell does not need to open any user files, however the user may wish to browse through the file hierarchy with the Motif File Selection Dialog.

To use the File Selection Dialog, choose the Select button near the file text that you want, and the File Selection Dialog will appear. The OK button will cause the selected file name to be copied to the text entry area in the control panel that last had focus. You can focus on a text widget (move the mouse cursor over the text widget, and press the left mouse button) and then hit the OK button. Unlike other Motif programs written in the OEL, this File Selection Dialog is non-modal. You may leave it up while you work with the main window. The OK button does not unmanage the dialog, so you can use it to fill filenames into several text widgets. The Cancel button will remove the dialog. The Help button will display help text for this dialog.

The Edit buttons will bring up an editor, which the user may choose in the resource file, to view the file specified by the contents of the currently selected text widget. The Exit button will cause the shell to terminate.

The shell also includes a Help button which is user configurable. This will pop up a single pane of help text. It is intended that

the designer of an OEL Shell also attach help to each widget in the work area. You may obtain help on any button or text field in the work area by selecting that object and then pressing the Help key. The default Help key for Motif application is the F1 key. To obtain help on a push button without activating the button, move the mouse cursor off the button until the button no longer appears to be pressed. You may then release the mouse button without any activation.

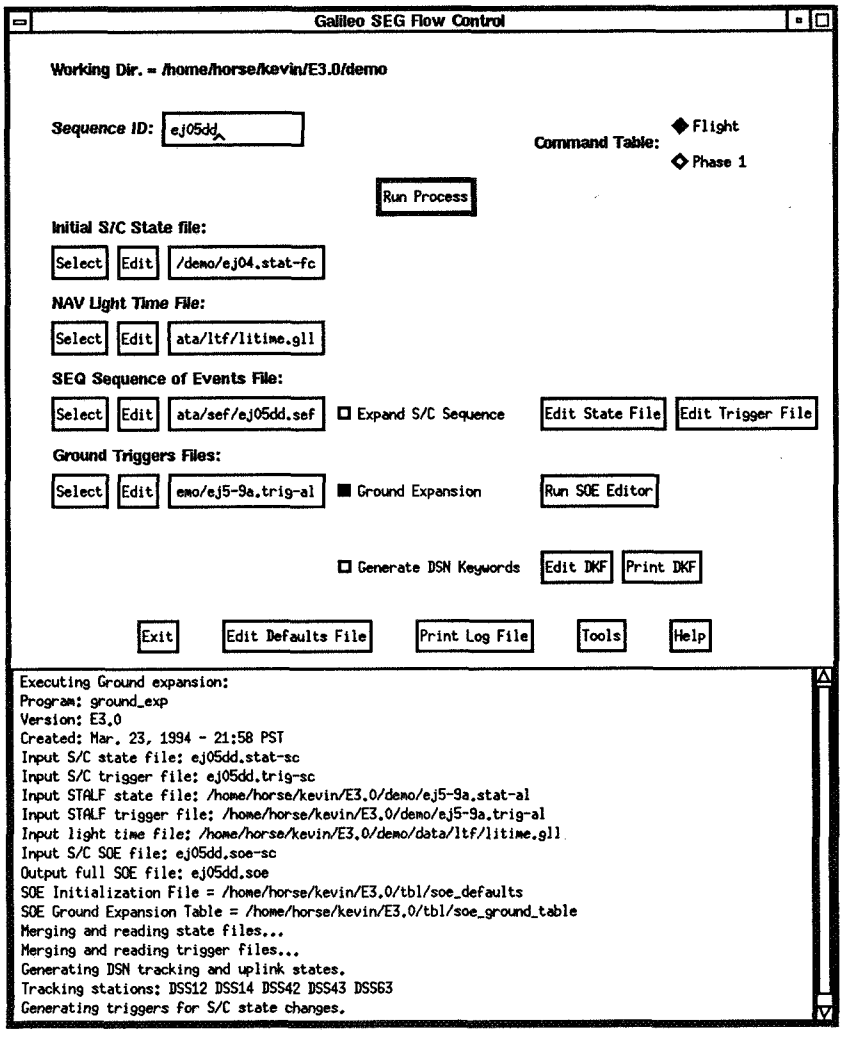


Figure 2. Galileo STALF Shell

WCL allows you to define the widget tree for an application in the resource file using new resource names such as wcCreate and wcChildren. In addition, callbacks are provided that set resources, manage and unmanage widgets, run an external program and exit.

The OEL Shell is a very simple application built on WCL [2]. It is basically about ten callback procedures which may be used in the resource file. The most important of these is CmdCB (the command callback). This callback executes its text string argument. For example, you could create a push button to execute the UNIX **ls** command as follows:

```
demo*lsPb.wcCreate: XmPushButton
demo*lsPb.labelString: Show Files
demo*lsPb.activateCallback: CmdCB(ls)
```

A simple command like this could be executed with WCL alone. The OEL Shell permits one to access text widgets, toggle widgets, and option menus, and pass these in CmdCB. For example, if demoTog is a toggle button, then $demoTog[-r] has the value in the brackets if the toggle button is true, and is the empty string if false. Likewise, the value of a text widget is just the text that the user entered.

Another very useful callback is the FocusCB which is used to specify the directory filter string used with the File Selection Box. A FocusCB is used with each text widget that is used to contain input file names.

Besides the resource file, two other files are used by the OEL Shell. These are a drawing file, that places simple X11 primitives (not widgets) in the drawing area. This has been used to give the OEL Shell the appearance of a flow chart. The other file is the help file.

Output from the child processes is sent to the scrolling message area below the work area. In addition, there are some special text messages that the child can send back which set resources in the OEL Shell.

In addition to SEG, we have used the OEL Shell for many other functions in operations. These include training, generating database queries, and running a command compiler.

Some advantages of using the OEL Shell are:

- It is easily configurable by operations personnel.
- It separates the computing engine from the GUI, thus simplifying testing of the computing engine.
- All functions may be run with or without a GUI.

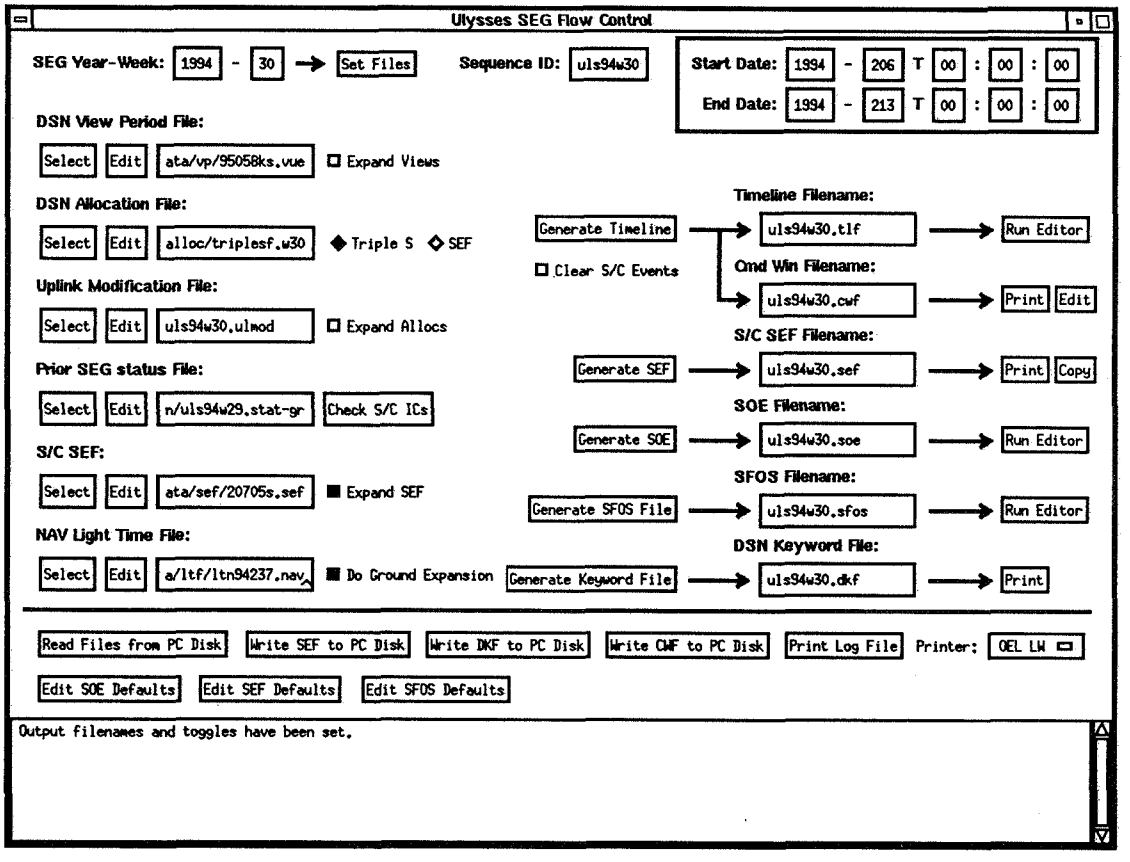


Figure 3. Ulysses SEG Shell

ULYSSES SEG

We have recently adapted the SEG software for Ulysses. The Ulysses mission is considerable different from the other missions in that the primary command mode is real time. Thus we do not have a spacecraft sequence file as an input to SEG. We introduced a new graphical document called the Timeline which contains the DSN allocation, view periods, and command windows translated into spacecraft time. This is generated from the DSN allocations and view periods files, and the light time file. The SEG operator then uses these times to schedule the spacecraft. Typical activities scheduled include: records and playbacks, telemetry mode changes and maneuvers. Since the SFOS editor is a general purpose timeline editor, it is also used to edit the Timeline document. The spacecraft information is then extracted and put into a file that roughly corresponds to the spacecraft sequence file for other missions.

From this point on, Ulysses SEG resembles SEG for the other JPL projects. The telemetry state of the spacecraft is extracted from the sequence file. The ground events are generated for the beginning and end of each track, the DSN configuration, spacecraft telemetry state changes, and other significant activities. This information is then used to create the SOE, SFOS, and DSN keyword files.

Ulysses SEG was the first project where the SFOS editor was used to input data that would then be passed on to other processes. The SFOS editor has functioned well, and it was easy to extract data from the SFOS records.

ACKNOWLEDGMENTS

This work was done at the Jet Propulsion Laboratory, California Institute of Technology, under a contract from the National Aeronautics and Space Administration. We would like to acknowledge the work of the technical staff in the OEL, the JPL Mission Operations Teams, and the Ulysses Spacecraft Team for their enthusiasm and support.

REFERENCES

1. The Widget Creation Library, David E. Smyth, September, 1991.

2. OEL Shell Programmers' and Users' Guide, Kevin J. Miller, October, 1993.

111124

N95-17539

354018

SPACECRAFT COMMAND AND CONTROL USING EXPERT SYSTEMS

July 22, 1994

Scott Norcross and William H. Grieser
Storm Integration, Inc.
CIT Tower - Suite 502
2214 Rock Hill Road
Herndon, VA 22070
(703) 478-6200 FAX (703) 478-6670

P-6

INTRODUCTION

We are in the midst of a revolution in the spacecraft command and control industry. This revolution is driven by several factors. Traditional customers of spacecraft command and control systems (like the government) are now trying to do more with less money. Where in the past the government would be inclined to design and build a system from scratch, today they are looking for an off-the-shelf solution. Another factor contributing to the changes in spacecraft command and control is the advancing technology of spacecraft. Several commercial ventures are underway to exploit large constellations of relatively cheap satellites. These new commercial space opportunities create a need for more economical command and control systems to satisfy these bottom-line oriented endeavors.

Some of the changing requirements in the market include:

- The skill level required to operate the system on a day-to-day basis is lower than required by traditional systems.
- The number of human operators required per satellite is smaller.
- The user interfaces are becoming graphical, as opposed to the text-based interfaces of traditional systems.
- The amount of time to prepare for a spacecraft mission is decreasing, making it harder for satellite users to develop their own system from scratch.

This paper describes a product called the Intelligent Mission Toolkit (IMT), which was created to meet the changing demands of the market. IMT is a command and control system built upon an expert system. Its primary functions are to send commands to the spacecraft and process telemetry data received from the spacecraft. It also controls the ground equipment used to support the system, such as encryption and decryption gear, and telemetry front-end equipment. Add-on modules allow IMT to control antennas and antenna interface equipment.

The design philosophy for IMT is to utilize available commercial products wherever possible. IMT utilizes Gensym's G2 Real-time Expert System as the core of the system. G2 is responsible for overall system control, spacecraft commanding control, and spacecraft telemetry analysis and display. Other commercial products incorporated into IMT include the SYBASE relational database management system and Loral Test and Integration Systems' System 500 for telemetry front-end processing.

Use of Expert Systems in IMT

Spacecraft command and control consists of a repetitive sequence of planning, contact and evaluation activities. During these phases, events occur and information is gathered that determine subsequent actions required to control the spacecraft. Traditional control systems require system operations personnel and spacecraft engineers to manually determine the appropriate responses to these events. In addition, to respond to recurring anomalous conditions that can be overcome via procedural solutions, operators often document detailed system conditions in a log book or operations manual. These references are examined by operations staff to determine how to resolve specific system conditions. If these conditions are not properly documented and accessible, the operations staff must consult with the operations "expert" to determine the appropriate course or action.

Using IMT's satellite support plan functions in combination with the embedded expert system, complex system conditions and responses are captured within a system knowledge base. IMT identifies specific events and conditions and invokes rules, procedures or specific satellite support plans to generate appropriate system responses. In this capacity, IMT stores, recalls and implements the knowledge of the system operations staff. The system can automatically respond to specific events or present suggested actions based on system conditions. The following are particular examples of how IMT implements these principles.

Telemetry Analysis and Display

The G2 expert system can be used to analyze telemetry data emitted by a spacecraft and determine the state of the spacecraft. G2's inherent ability to model real-world objects supports sophisticated analysis of complex data. The data can also be displayed to the user through G2 objects, presenting the data in a format that is easier to understand than traditional text-based displays.

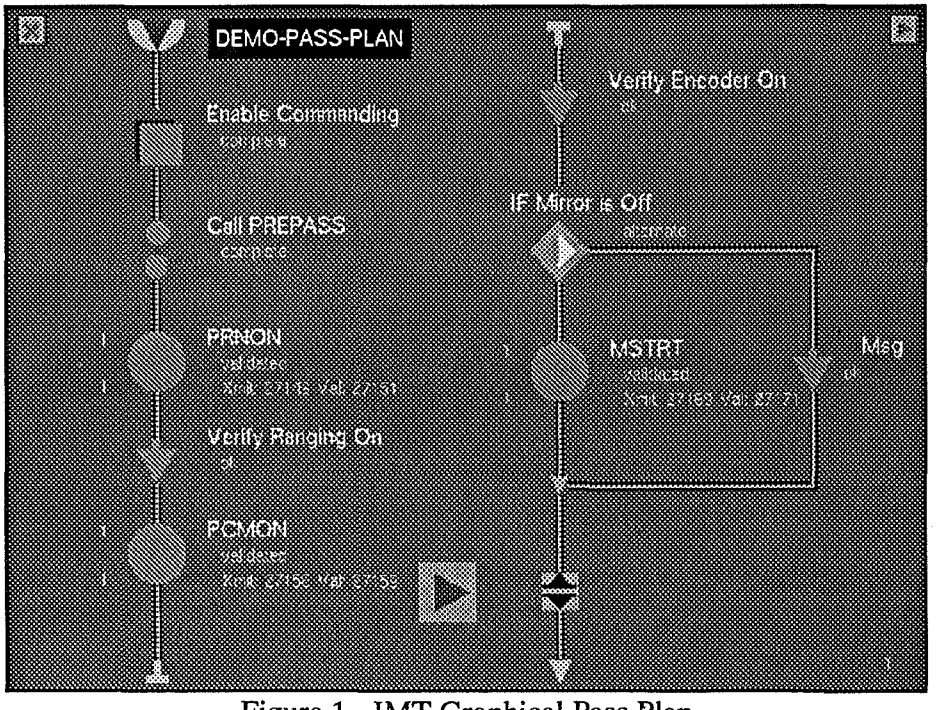


Figure 1 - IMT Graphical Pass Plan

Graphical Pass Plan

In IMT, a "pass plan" is a sequence of spacecraft commands and system configuration actions called "steps." IMT uses G2 to represent each step as a G2 object, and the flow of execution through the steps is indicated using the G2 connection facility. A graphical pass plan resembles a flow chart, which is more intuitive than the proprietary commanding languages used by other command and control systems. As the pass plan is executed, the current step is highlighted; status information about each step is presented along with the G2 icon for the object.

There are two ways to create pass plans. The first is to select commands from command palettes and connect them into graphical sequences to form pass plans. The second way to create a pass plan is to build an ASCII file using an off-line process (e.g. using an editor or the output from another tool). IMT's Pass Plan Import function is then used to convert the ASCII file into a graphical pass plan where it can be executed like any other pass plan.

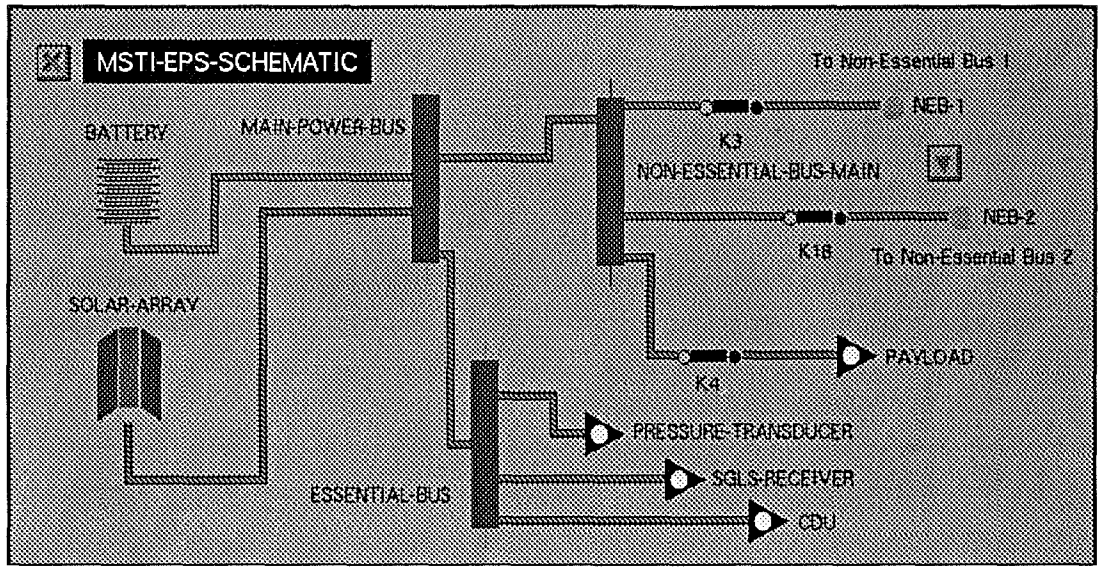
IMT supports two modes for pass plan execution: manual and automatic. Automatic execution provides the first step toward the complete automation of system operation. During automatic execution, command sequences are executed without operator intervention. Automatic execution continues until the sequence is completed successfully or until an anomaly is detected. Anomaly detection could be based on the inability to properly transmit the command from the ground system, a command rejection from the spacecraft, or the result of a complex set of rules developed to verify the command operation.

Logic in Pass Plans

Logic is provided through an "if step," which is analogous to an "if" statement in a high level computer programming language. When an "if step" is executed, G2 executes rules provided by the pass plan builder to determine which step should be executed next.

Interactive Telemetry Displays

IMT can be used to build "smart" interactive telemetry displays. These displays allow the operator to control the spacecraft by directly manipulating graphical representations of the system. For example, circuit diagrams representing portions of the electrical power subsystem can be created that contain graphical representations of subsystem components. The user could then click on the graphical representation of a switch to change the switch's position to allow (or prevent) current flow to the subsystem. This frees the operator from having to know the details of specific commands required to manipulate a system component (spacecraft or ground system) and creates a more "results oriented" user interface.



Command Verification

Traditional spacecraft command and control systems require manual examination of telemetry to determine the status of a spacecraft component or subsystem. Manual actions are initiated based on the examination of this data. For example, after transmission of a command, operators may continue to view telemetry data to determine if the spacecraft received the command and is responding as expected.

IMT uses expert system rules to automate the analysis of telemetry data, determine the status of the spacecraft, and identify necessary control actions. Specific control actions are captured in rules which are invoked after command transmission. Rules can be designed to examine specific data points and determine whether the desired reaction was achieved. Actions, as directed by the operations experts, can be initiated based on the results of the execution of these rules.

Commanding Constraints

Before a command is transmitted, IMT consults the knowledge base to determine whether it is acceptable to send the command. IMT allows the operator or engineer to specify command transmission constraints. To specify a constraint, a rule is written to which G2 backward chains during command transmission. These rules can refer to any available data to reach this conclusion. This includes telemetry data, system state, and even the person making the request to send the command. Using G2, it is easy to define constraints that can be turned on and off.

By defining a rich set of constraints, the end-users can customize their system to minimize the risk of using lower-skilled spacecraft operators.

Automatic Analysis of Pass Plans

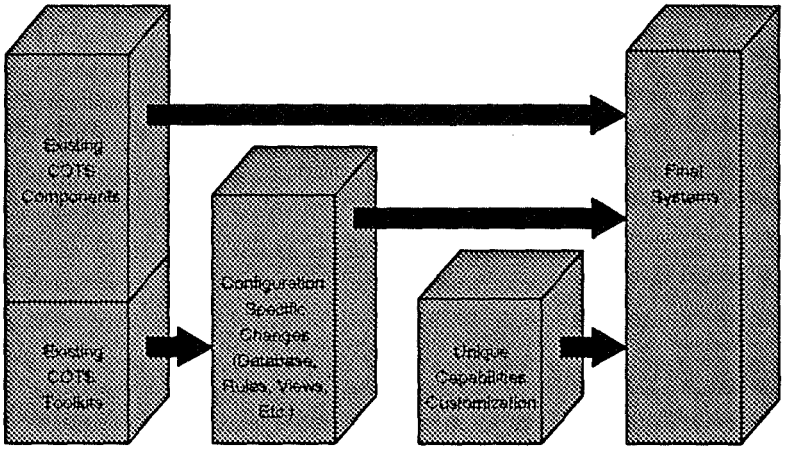
During mission planning, spacecraft operators determine the future activities of the spacecraft. The objectives of these activities are determined by vehicle maintenance requirements, overall mission objectives, and operations required to ensure the health of the spacecraft. Using IMT and the embedded expert system, mission objectives can be captured and applied during the planning process. For example, to ensure the health of a spacecraft, mission objectives might indicate that battery reconditioning must be performed at precise time intervals. These objectives can be stored as rules within the planning knowledge base.

As mission planners develop future contact support plans, this knowledge can be used to validate the proposed pass plans and command sequences. As system intelligence increases, this analysis can incorporate knowledge from previous spacecraft contacts. For example, suppose the last time the vehicle was contacted, a specific anomaly was detected. Using knowledge of this condition, along with the expert spacecraft knowledge captured by the system, the system could identify a proposed command sequence as ineffective or dangerous to the spacecraft.

THE TOOLKIT MODEL

IMT was designed specifically to support a dynamic system environment. The "Toolkit" model allows the product to be configured to satisfy a variety of mission unique requirements and ensures the system can evolve to meet changing system requirements.

The "Toolkit" Model emphasizes the use of COTS products as the foundation for final solutions. Rather than developing a complex system from scratch, the target system is developed by integrating commercial products - best suited for the target application - into a final solution. Mission unique requirements are implemented primarily through modifications to expert system knowledge bases and standard relational databases. In addition, many commercial products provide graphics rich tools that allow the system to be tailored to meet user specifications without extensive software development. This environment supports rapid system customization and reduces development, operations and maintenance costs. When development is required, the level of effort is significantly lower than that required by traditional system development approaches.



CONCLUSION

The Intelligent Mission Toolkit provides significant advantages to the implementation of a complex command and control system. The embedded expert system offers the ability to store and apply expert mission operations and planning knowledge using system knowledge bases. This information can be used to automate spacecraft command validation, control ground system equipment and apply intelligence to the entire mission planning process.

IMT's modular architecture and fully Object Oriented implementation addresses the complex requirements of modern command and control systems. The "Toolkit" model emphasis allows end-users to customize the product to satisfy unique mission objectives resulting in the most powerful and flexible commercial command and control system available.

111125

N95-17540

354028

p. 8

SEQ_GEN: A Comprehensive Multimission Sequencing System

Jose Salcedo, Thomas Starbird

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California

Abstract

SEQ_GEN is a user-interactive computer program used to plan and generate a sequence of commands for the spacecraft. Desired activities are specified by the user of SEQ_GEN; SEQ_GEN in turn expands these activities, deriving the spacecraft commands necessary to accomplish the desired activities. SEQ_GEN models the effects on the spacecraft of the commands, predicting the state as a function of time, flagging any conflicts and rule violations. These states, conflicts, and violations are viewable both graphically and textually at the user's request. SEQ_GEN also displays the entire sequence graphically, showing each requested activity as a bar on its graphical timeline. SEQ_GEN includes a full-screen editor, allowing the user to make changes to the requested activities. After a change has been made to the sequence, SEQ_GEN immediately revalidates the sequence, updating its models and calculations along with its displays based on these changes. Because SEQ_GEN is user-interactive and because it has the ability to recalculate spacecraft states immediately, the user is able to perform "what-if" sessions easily.

SEQ_GEN, a multimission tool, is adaptable to any flight project. A flight project writes its adaptation files containing project unique information including in its simplest form, only spacecraft commands. For more involved projects the adaptation files may also contain flight and mission rules, description of the spacecraft and ground models, and the definition of activities. SEQ_GEN operates at whatever level of detail the adaptation files imply. Simple adaptations are straight forward to do. There is, however, no limit to the complexity of activity definitions or of spacecraft models; both may involve unlimited logical decision points. Commands and activities may involve any number of parameters of a wide variety of data types, including integer, float, time, boolean, and character strings.

SEQ_GEN will be used by the Mars Pathfinder, Cassini, and VIM (Voyager Interstellar Mission) projects in an effort to speed up adaptation time and to keep sequence generation costs down.

SEQ_GEN is hosted on UNIX workstations. It uses MOTIF and X for windowing, and was designed and coded in an object-oriented style in the language C++.

Introduction

SEQ_GEN is a flexible software tool that can be used in several roles in the uplink planning and sequence generation process. In this paper, we address various tasks that are done during uplink planning and sequence generation, and show how SEQ_GEN supports each of them. We begin with comments that apply to all uses of SEQ_GEN.

Typically, SEQ_GEN is used interactively. The user sees results of SEQ_GEN computations on a graphical timeline (see Figure 1). If there are conflicts or rule violations, the user changes the sequence by using SEQ_GEN's editor to alter, add, or delete requests. SEQ_GEN then recomputes the state of state of the spacecraft, and reevaluates the rules. This process is repeated until the user is satisfied with the sequence, at which

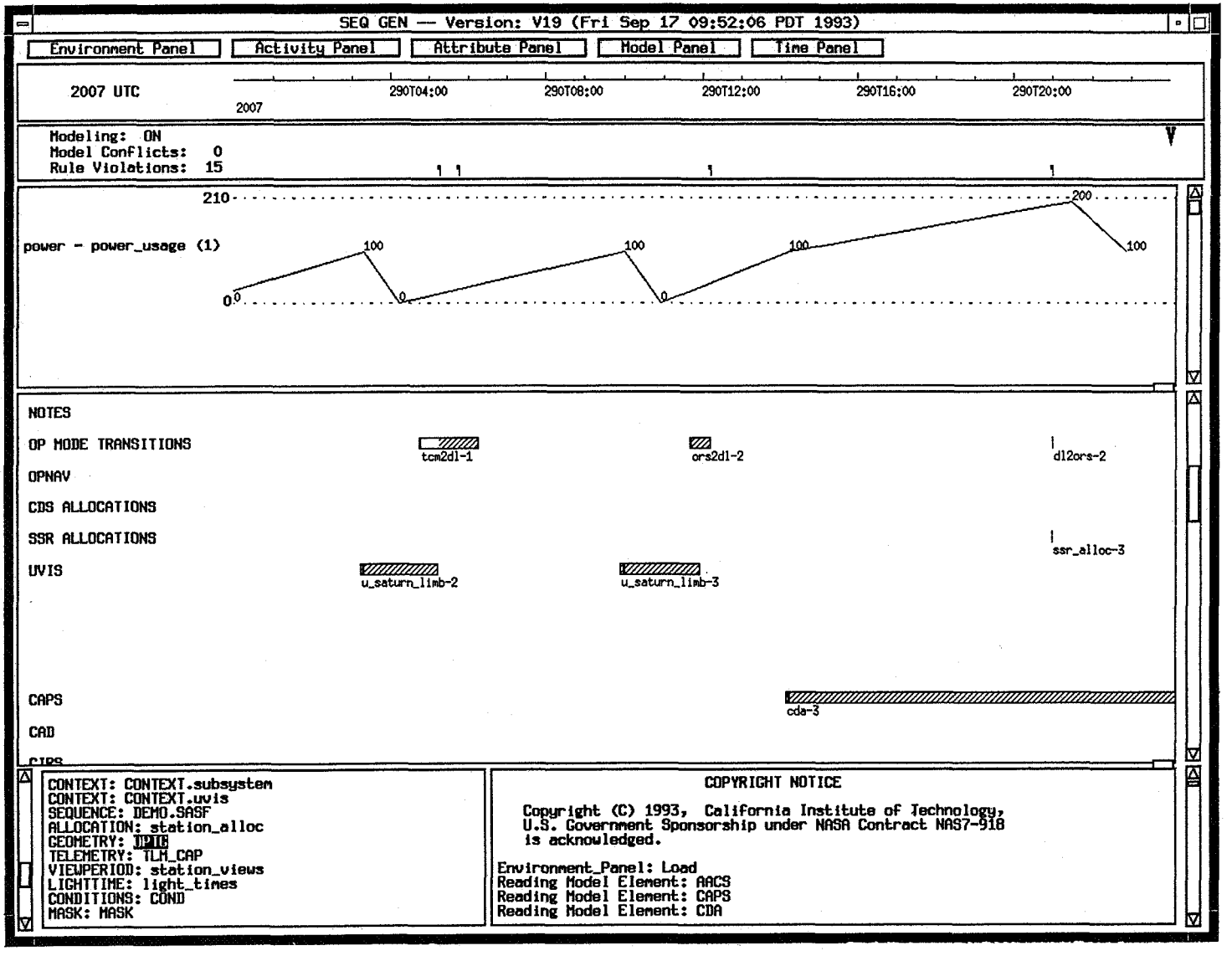


Figure 1. SEQ_GEN TIMELINE

time SEQ_GEN writes results into computer files.

A key feature of SEQ_GEN is that it is a multimission program. Adaptation of SEQ_GEN for use by a specific flight project in a specific role is done by supplying SEQ_GEN with files of data about the project. Only the information pertinent to the intended use of SEQ_GEN is required. In this paper we use the term "adapter" to denote the person or persons that supply the information about the flight project. The term "user" denotes a person who is using an adapted SEQ_GEN.

Now we discuss how SEQ_GEN supports various uplink planning and sequence generation tasks, showing SEQ_GEN's flexibility.

Generating Command-Level Sequences

Command-Level Editing

One simple use of SEQ_GEN is as an editor, enabling a person to write a sequence of spacecraft commands. The adapter provides the list of all spacecraft commands for the flight project. If the commands have parameters, those are named by the adapter. The adapter can specify what the allowable values are for each parameter of each command, and what type of value is appropriate (decimal integer, hexadecimal, octal, binary , floating point, duration, time, character string, boolean, or a one-dimensional array of any of the previous types). The adapter can also specify the default value of each parameter. In this way, SEQ_GEN "knows" a project's spacecraft commands.

When the user wants to add a command to the sequence, SEQ_GEN lists all the commands, letting the user choose one. SEQ_GEN displays the name and description of each parameter (as supplied by the adapter), to guide the user in specifying the requested command. SEQ_GEN will use whatever information the adapter has supplied concerning the parameters of the command. For example, if the adapter has supplied the allowable range, SEQ_GEN will warn the user when a value given by the user is not in range. SEQ_GEN's editor enables a person to form a request, consisting of one or more commands (and also "activities"; see below) and to add that request to the sequence.

Command-level Sequence

One output of SEQ_GEN in the simple adaptation described above is a file of all the commands (in mnemonic form), in time order, ready to be translated to bits and sent to the spacecraft.

In addition, SEQ_GEN produces a timeline (see Figure 1), both interactively on the screen, and on paper. The timeline shows visually the position in time of each request in the sequence.

Such an adapted SEQ_GEN is useful for building sequences for use before launch in the testing of the spacecraft. It is also useful for simple projects where command-level sequence planning is adequate, and where any constraints on interactions of commands can be checked by hand.

Spacecraft Clock

If precise timing of commands with respect to the spacecraft's clock is vital, the adapter can define the units of the clock and their nominal durations. The definition is then used in some of SEQ_GEN's calculations. For example, there is an option in SEQ_GEN to align all commands' times to the nearest whole unit in the spacecraft's clock. The relation between Universal Time and the values of the spacecraft's clock is given to SEQ_GEN at run-time, to account for differences in the clock's rate from its nominal rate.

Merging Sequences

Another feature of SEQ_GEN is the ability to merge sequence files. For example, SEQ_GEN could be used individually by different flight team members making their individual request files. Those files can then be merged to produce a single time-ordered file with all of the requests. Each request retains its requestor's name (or other identifying string), so that the individuals can check that their requests were properly handled.

Different requestors could include members of the engineering team (for example, an attitude control analyst requesting a calibration), or of the navigation team (requesting a maneuver), or of science teams (requesting scientific observations).

Predicting Events

It is often useful to predict the effects of the commands in a sequence. The adapter can supply models to SEQ_GEN that enable predictions of the state of the spacecraft based on the commands in the sequence.

Flexibility of Models

A nice feature of SEQ_GEN is the variability possible in the models. One possibility, of course, is to have no models at all. In this case, as discussed above, SEQ_GEN's output is the time-ordered list of commands in the sequence.

Models of varying complexity can be added. For example, if the amount of power being used at any time during the sequence is of interest, a power model could be added. The adapter defines a model element by specifying its attributes (i.e., state variables). An attribute can be of any type (same choices as for parameters of a command; see above), and the adapter can define the allowable range of each (in which case SEQ_GEN will give a warning to the user if the attribute's value ever becomes out of range). For each spacecraft command that affects an attribute, the adapter describes the effect, using a simple language provided by SEQ_GEN. The language includes the basic programming language constructs, such as IF statements and loops. In addition, the language C can be used by the adapter to specify calculations. No compiling or linking of SEQ_GEN is needed to incorporate the adapter's compiled C code; the linking of the adapter's code is dynamic, done at run-time.

The effect of a command can depend on the state of the model before the command. The most common effect of a command is to change the value of an attribute.

Simple models, such as ones that keep track of whether a switch is "on" or "off", are simple for the adapter to specify. Each project can model the details appropriate for its sequencing needs.

The modeling done in SEQ_GEN is a discrete event simulation, where the commands in the sequence are the triggering events. SEQ_GEN processes each command by interpreting the simple language in which the adapter has written the effect of the command, and by calling any C functions the adapter may have used. The adapter can use SEQ_GEN's "stimulus" concept to promote the effect of a command to future time or to several model elements.

SEQ_GEN has built-in the ability to read files of Deep Space Network view periods and allocations, a file that contains predictions of downlink data rate capability, and a file that contains trajectory events, such as occultations. The adapter can write effects of such events in the same way as writing effects of commands. For interplanetary missions, where the light time is non-negligible, SEQ_GEN has the capability of adjusting times between ground time

and spacecraft time using a file giving the light time.

Predicted Events File

SEQ_GEN produces a comprehensive file that contains the results of the modeling (see Figure 2). The file is a time-ordered list that contains an entry whenever an attribute of a model is set to a value. The entry consists of the time, the values of the attributes of the model element, and an indication of the causal command. The file also lists all commands in the sequence. (Activities and rule violations are also in the file; see below.) The file can be used to review a sequence.

Interactive Display of Models

The user of SEQ_GEN can turn the modeling on or off at will. The user can also have SEQ_GEN display a graph of the value of any one or more attributes above the timeline of requests (see Figure 1). The user can change what to display any time during the SEQ_GEN session. When the user changes the sequence, SEQ_GEN models the part of the sequence being viewed and updates all the displays.

Thus the adapter has great flexibility in what models to build and how detailed to make them, and the user has complete flexibility in choosing what model attributes to display on the screen during the session.

Different Users, Different Models

Even on a single flight project, different adaptations of SEQ_GEN could be used. For example, an attitude control expert may include more detailed models of attitude, but omit models of interest only to a scientist, and vice versa.

Checking Rules

The adapter can add "rules", which are stated in terms of the model attributes. SEQ_GEN has eight types of rules. A rule contains a boolean-valued expression of model attributes. During modeling of a sequence, if the expression becomes true (or remains true for too long, or for not long enough, or becomes true too many times, or not enough times, or becomes true before some other state has occurred for long enough), the rule is considered violated. An indication of the violation occurs above the timeline of requests (see Figure 1). The user can click on the indication to get details of the violation. Rule violations are also included in the Predicted Events File.

By defining rules, the adapter enables SEQ_GEN to perform some of the validation of a sequence.

For situations where none of the eight built-in types of rule adequately reflects the constraint desired to be checked, the adapter can use logic in the models themselves to declare a conflict. An indication of conflict appears above the timeline (see Figure 1), and appears in the Predicted Events File.

Thus SEQ_GEN is flexible in the rules it can check. Just as different users could use different models, so they could use rules tailored to their interest.

Making High-Level Requests; Activity Types

SEQ_GEN offers flexibility in the level at which a user requests commands for the sequence. The adapter can define "activity types" (also called "blocks"), which can then be used in users' requests.

A simple activity type is a list of spacecraft commands, with their relative timing specified. The activity type has a name. By requesting an activity of that name, the user is effectively adding all the commands in the activity type's definition to the sequence, timed relative to the time specified for the request.

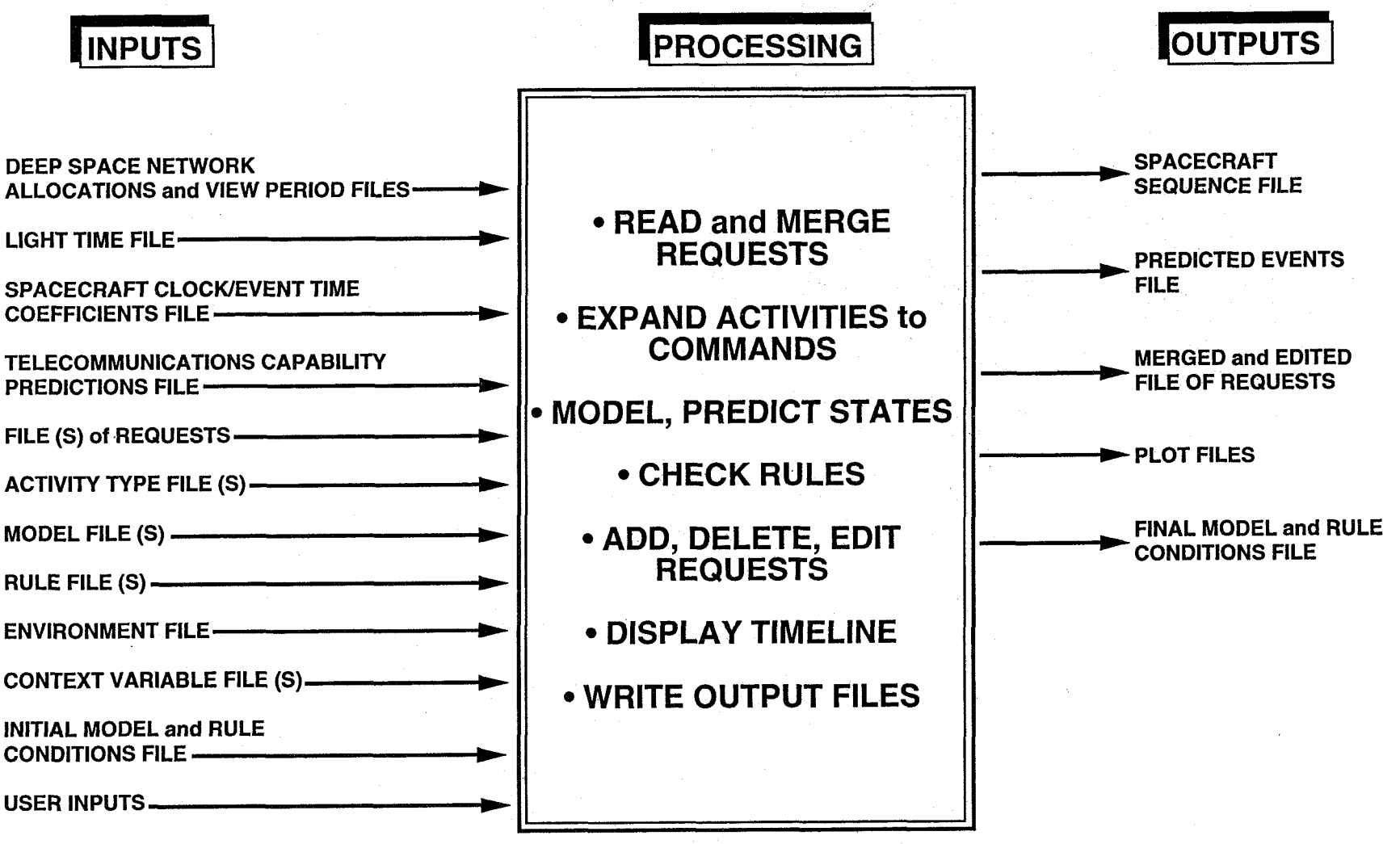


Figure 2. SEQ_GEN FUNCTIONS

SEQ_GEN is flexible in how complicated the definition of an activity type can be. An activity type can have parameters. The user, when requesting an activity of that type, is prompted by SEQ_GEN's editor for values of the parameters. The values can be used for parameters in commands that appear in the definition of the activity type. The values can also be used in logical constructs (such as IF statements) that govern what commands will be used in the activity. For example, in an activity type that represents a maneuver of the spacecraft, a parameter could be an option determining whether or not to turn on the gyroscopes.

The definition of an activity type can refer to other activity types (i.e., activities can be nested).

Activity types are "expanded" by SEQ_GEN to produce commands. The commands are modeled along with any commands requested explicitly by the user.

Using activities allows the user to think at a higher level than individual commands. Also, the definition of an activity type can be written or checked by experts, and tested before use. A person who is not an expert can then safely use the activity.

Some activity types represent on-board programs that can be invoked in a sequence to yield several commands. Such an activity type, called an on-board block, is expanded by SEQ_GEN for modeling, but is not expanded on the Spacecraft Sequence File (see below).

Writing the Spacecraft Sequence File

Another output of SEQ_GEN is a computer file called the Spacecraft Sequence File. This file contains (a mnemonic representation of) the information that must actually go to the spacecraft, i.e., spacecraft commands and calls to on-board blocks. Conversion of this file to binary in a form packaged for transmission to the spacecraft is not a function of SEQ_GEN.

Planning without Commands; d_commands

Activity types can actually be defined even if spacecraft commands have not been defined. SEQ_GEN has the concept of d_commands (dummy commands), which are requested by the user and modeled by SEQ_GEN as commands are, but which are not placed in the Spacecraft Sequence File. In this way, an adaptation of SEQ_GEN can be made wherein activity types are defined in terms of d_commands, which can trigger abstract or approximate models. An example of an abstract model is one telling whether a maneuver is in progress. Such an adaptation is useful for planning sequences early in the planning stage, or early in the life of the project.

Both actual commands and d_commands can be used in the same activity type and in a single sequence. Thus modeling and rule checking involving actual commands can be supplemented by modeling and rule checking of abstractions.

Changing Adaptation

The adaptation information is given in ASCII files (plus optional C code in the model or activity definitions). The adaptation can be changed as the mission progresses. Another program, called SEQ_ADAPT, is being developed to aid the adapter in producing syntactically correct and consistent adaptation files.

History and Use of SEQ_GEN

SEQ_GEN (under different names) has its historical roots in the Mariner Mars 1971 project, a Mars orbiter. Most major later projects at the Jet Propulsion Laboratory, including Voyager and Galileo, wrote new versions specific to the project. In the last few years, the

current version, which is a multimission version, was developed.

Its activity features were used on Mars Observer. It will be used on Mars Pathfinder, VIM (Voyager Interstellar Mission), and Cassini. SEQ_GEN is hosted on Sun SPARC and Hewlitt-Packard workstations.

Development of SEQ_GEN

SEQ_GEN has about 55,000 lines of code, written in C++ in an object-oriented style (Wirfs-Brock et al., 1990). SEQ_GEN is Category A; it was developed with full rigor and testing.

Summary

SEQ_GEN is a comprehensive and flexible tool for use in uplink planning and sequence generation. SEQ_GEN is flexible in that

- it can be adapted for use in any flight project, or for different classes of user in a single project
- it can be adapted in several versions, with or without spacecraft commands, models, rules, and activity types
- models can be simple or detailed
- models can be of actual spacecraft parts and/or of abstract quantities
- models can be triggered by spacecraft commands or by d_commands
- adaptation does not require compiling or linking of SEQ_GEN

Acknowledgement

The work described in this paper was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration.

The SEQ_GEN program was developed by Russ Brill, Imin Lin, Win Lombard, Bob Oliphant, John Sisno, Jose Salcedo, and Tom Starbird.

References

McLaughlin, W.I. and Wolff, D.M., "Automating the Uplink Process for Planetary Missions", AIAA 89-0580, AIAA 27th Aerospace Sciences Meeting, Reno, January 9-12, 1989.

Salcedo, J., "Version 19 SEQ_GEN User Guide," D-11261, December 1, 1993 (JPL internal document)

Wirfs-Brock, R., Wilkerson, B., & Wiener, L.(1990). *Designing Object-Oriented Software.* Englewood Cliffs, New Jersey: Prentice Hall.

N95-17541

111126

354045

P-8

THE PACOR II EXPERT SYSTEM: A CASE-BASED REASONING APPROACH TO TROUBLESHOOTING

Charisse Sary
Computer Sciences Corporation
7700 Hubble Drive
Lanham-Seabrook, MD 20706

Abstract

The Packet Processor II (Pacor II) Data Capture Facility (DCF) acquires, captures, and performs level-zero processing of packet telemetry for spaceflight missions that adhere to communication services recommendations established by the Consultative Committee for Space Data Systems (CCSDS). A major goal of this project is to reduce life-cycle costs. One way to achieve this goal is to increase automation. Through automation, using expert systems and other technologies, staffing requirements will remain static, which will enable the same number of analysts to support more missions.

Analysts provide packet telemetry data evaluation and analysis services for all data received. Data that passes this evaluation is forwarded to the Data Distribution Facility (DDF) and released to scientists. Through troubleshooting, data that fails this evaluation is dumped and analyzed to determine if its quality can be improved before it is released. This paper describes a proof-of-concept prototype that troubleshoots data quality problems.

The Pacor II expert system prototype uses the case-based reasoning (CBR) approach to development, an alternative to a rule-based approach. Because Pacor II is not operational, the prototype has been developed using cases that describe existing troubleshooting experience from currently operating missions. Through CBR, this experience will be available to analysts when Pacor II becomes operational.

As Pacor II unique experience is gained, analysts will update the case base. In essence, analysts are *training* the system as they learn. Once the system has learned the cases most likely to recur, it can serve as an aide to inexperienced analysts, a refresher to experienced analysts for infrequently occurring problems, or a training tool for new analysts.

The Expert System Development Methodology (ESDM) is being used to guide development.

Pacor II Overview

The Pacor II DCF acquires, captures, and performs level-zero processing of packet telemetry for spaceflight missions that adhere to communications services recommendations established by CCSDS. Pacor II provides three forms of service for packet processing: real time, routine production, and quicklook. It strips packets from telemetry frames, reassembles packets, sorts packets by selected fields, merges packets from different sessions, and delivers scientific data sets and other related products to the user.

Analysts provide packet telemetry data evaluation and analysis services for all data received. Data passing this evaluation is forwarded to the DDF and released to scientists. Through troubleshooting, data failing

this evaluation is dumped and analyzed to determine if its quality can be improved before it is released.

A major goal of the Pacor II project is to reduce life-cycle costs. One way to achieve this goal is to increase automation. Through automation, using expert systems and other technologies, staffing requirements will remain static, which will enable the same number of analysts to support more missions.

Problem Identification

Through discussions with Network and Mission Operations Support analysts, additional candidate areas for automation were identified. We focused on areas where the human reasoning processes of experts could be automated. Analysts provided a study that showed where they spent their time in the Hubble Space Telescope (HST) DCF for a 1-week period. Fifteen tasks were identified. The study described the percentage of staff-hours expended in each task for current operations and for projected future operations as workloads are expected to increase. The troubleshooting/dump analysis task had the highest potential benefit and was also suitable for implementation as an expert system.

Benefits

Through additional discussions with analysts, the troubleshooting problem was further evaluated for implementation as an expert system. Several potential benefits appeared to be possible.

Capture and store experience: Analysts felt that it would be useful to have a system that would enable them to more readily access prior troubleshooting problems and solutions. Currently, when problems recur, analysts must remember how they were fixed. If it is a problem that another analyst handled, analysts have to discuss it with each other or look up the problem and solution in a log book. Log books are available for analysts to record how they fix problems; however, specific requirements for the information stored there does not exist. The information may be sketchy, inconsistent, and difficult to find.

Analysts felt that a record of their prior troubleshooting knowledge, with an easy way to access the information, would help them in solving new or recurring problems. They also felt that troubleshooting experience from prior missions, including Pacor I, would be beneficial for Pacor II analysts at the start of the Pacor II mission, even though some problems may be new.

Expertise available during off hours: Shift analysts are the first analysts who fix problems that occur. If these analysts cannot fix a problem, troubleshooting analysts fix the problem. However, troubleshooting analysts only work during the day shift. An expert system could be an assistant to shift analysts on other shifts who do not have access to troubleshooting analysts and who are not as proficient in fixing problems.

Retain expertise with high turnover rate: Due to the nature of operations, analysts are required to work rotating shifts. Because this is demanding on the individuals involved, analyst turnover is high, which results in a high demand for training of new analysts. Analysts felt that it would be useful to have a system that would help in training and assisting inexperienced or new analysts perform their jobs. Also, because the Pacor II lifetime is expected to be long, expertise can be retained during personnel turnover through the use of expert systems.

Increased workload for same number of staff: Facility personnel currently handle complex decision-making processes. Through the use

of expert systems, some of these processes can be automated, which frees the analyst to concentrate on exceptional situations and relieves the analyst from performing the more routine decision-making tasks. This automation would enable the same number of analysts to handle an increased workload.

Case-Based Reasoning Overview

CBR is a kind of expert system or another way besides rules to build an expert system. CBR uses past experience in solving new problems by storing previous experience or cases in a case base or database of cases. Cases are indexed so that they can be easily retrieved from the case base, and retrieved cases can be adapted to solve new problems.

Figure 1 illustrates the CBR process. Application domain knowledge is stored as a set of cases that describes past experience. Each case is composed of a set of features with values associated with these features. Typical information that might be included as features of a case are a description of a problem, a solution for the problem, how the solution was reached, and the expected result following implementation of the solution. Most often, the case base is developed incrementally over time as users find and solve new problems.

When a new problem is encountered, an analyst enters the characteristics or symptoms of the new problem as a new case. The CBR system searches the existing case base for cases that match and then displays a set of closely matching cases. Cases are ranked to indicate the degree of match between an old case previously stored in the case base and the new case.

If there are no exact matches, adaptation is often performed where a closely matching case is adapted to fit the new situation. There are two types of adaptation: manual and automatic. In manual adaptation, a user modifies a closely matching case manually. The modified case is then stored so that it can be reused when the problem occurs again. In automatic adaptation, the system automatically adapts an existing case. This adaptation is typically performed using a set of rules that describe how an existing case should be adapted.

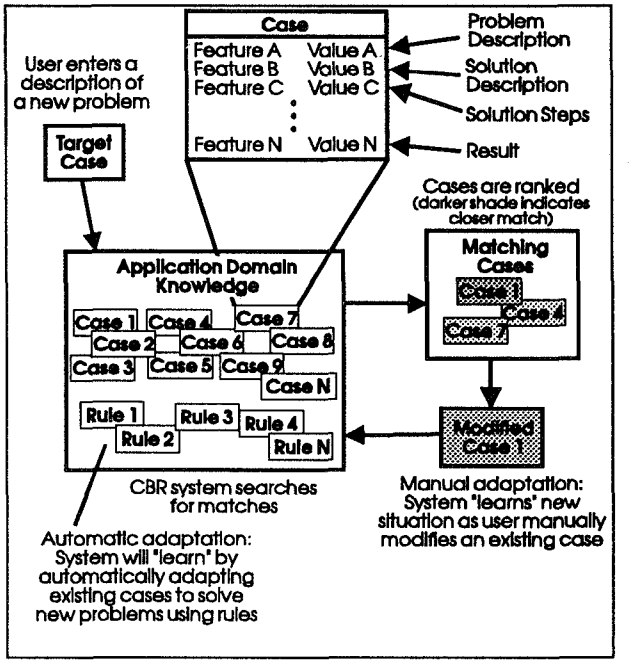


Figure 1. CBR Approach to Problem Solving

Advantages to CBR Approach

The CBR approach to problem solving has many advantages. Solutions to problems can be quickly derived because past experience is applied to the current problem. Previously obtained solutions can be reused rather than repeating the entire reasoning process each time the same problem recurs. Novices can use a CBR system to quickly obtain solutions to problems without a deep understanding of the process involved in deriving the solution. Also, with CBR, novices are prompted for the important features and do not have to remem-

ber what is important, which makes CBR systems useful training tools. Finally, past correct solutions and solution paths, as well as past mistakes that may have been forgotten, can be reapplied to new problems, eliminating "reinventing the wheel." The system becomes more robust as more cases are added or existing cases are modified.

Rule-based expert systems have been widely used to handle problems dealing with automating the human reasoning processes of experts. The CBR approach to problem solving has many advantages over the rule-based approach. It is often easier to add new cases to a case base as compared to adding new rules to a rule base. For example, it is not always clear what the effect of adding one rule to a rule base will have on other rules in the rule base. In CBR, each case is an independent entity and does not interact with other cases as a rule does when it fires other rules.

CBR solves problems more similarly to the way humans solve problems. Humans most often use what they already know in solving a new problem, reapplying a previous solution path and solution, rather than generating a new solution every time. They adapt what they already know to solve a current problem. Because cases are more understandable to the end user or expert, CBR systems are easier for a human to understand, build, use, and maintain, which also makes knowledge acquisition easier. However, as with any intelligent system, users must be cautioned not to blindly apply the recommended solution without thoroughly evaluating it to ensure that it is indeed the correct one.

Two types of problems are most suited to the CBR approach: (1) those where a significant number of past experiences or cases are available that are applicable to new problems and (2) problems where all solutions or expertise are not known in advance or where the domain is not well understood.

Rationale for Choosing CBR

Based on the characteristics of the troubleshooting problem, we felt that the CBR approach was a suitable approach for troubleshooting for several reasons. Pacor II conventional software is under development. Therefore, the necessary troubleshooting expertise for Pacor II does not currently exist. However, a troubleshooting assistant could be developed for Pacor II analysts from existing mission experience and, subsequently, for logging Pacor II troubleshooting sessions after Pacor II becomes operational. A Pacor II troubleshooting system could be developed incrementally as knowledge is gained. Also, analysts could take a major part in populating an initial case base during development, after case base design is stable, and they can perform their own maintenance during operations.

Methodology

ESDM describes a standard methodology to follow when developing an expert system. Because requirements are unknown at the beginning of an expert system project, by developing a series of progressively more complex prototypes, requirements will be identified and validated. ESDM is based on an iterative life-cycle model or spiral model. Each iteration adds knowledge about what the human expert does and what the requirements should be for the system. Each iteration also reduces the risks and uncertainties about the feasibility and practicality of using expert system technology for a given system.

ESDM is composed of five stages. The product of each stage is an executable prototype. We are using ESDM for this project and

have developed the first-stage prototype or a Feasibility Stage prototype.

The prototype produced during the Feasibility Stage automates one or a few key functions of the human expert and concentrates on feasibility issues.

Prototype Implementation

We have developed a proof-of-concept prototype that assists analysts in troubleshooting data quality problems. If the quality of the data received in the DCF is below a certain level, the analyst must determine the cause of the problem and decide if the quality of the data can be improved before it is forwarded to the DDF and to scientists.

The initial prototype is composed of a set of 12 cases. We expect the final system to contain about 100 cases. The cases range in level of detail from very broad, network-type anomalies to very specific, spacecraft-related anomalies. Categories of cases were classified into four general types:

- Spacecraft problem or spacecraft to ground station link problem
- Ground station to NASA Communications (Nascom) (GSFC) link problem
- Nascom to GSFC Building 23 interbuilding data distribution resource/interbuilding data transmission system (IBDDR/IBDTS) link problem
- BDDR/IBDTS to Pacor II link/Pacor II internal problem

The initial case base contains cases from the first three categories. Six of the cases are from Pacor I and six are from the HST DCF.

Each case is composed of a title to identify a case, a set of symptoms or a description of the problem, a description of the cause of the anomaly (solution description), and an explanation of what an analyst should do to handle the anomaly (action). Figure 2 provides a sample case.

Title: Nascom to Sensor Data Processing Facility (SDPF) Link Problem
Problem Description:
Frame-level errors—*Cyclical redundancy code (CRC)*
Block-level errors—*Polynomial errors*
System results match—*Generic Block Recording System*
Packet errors—*Missing packets or gaps*
Percent recovery—*Greater than 100%*
Data Type—*Playback Recorder*
Data Inversion Performed—*No*
Gap characteristics—*No gap in block time*
100% recovery—*Yes*
Inversion flag changes and frame synch pattern is valid but inverted—*No*
Duration of gap—*Less than 4 minutes*
Number of missing packets—*Greater than 1*
Frame CRC corresponds to each packet gap location—*Yes*
Location of frame errors corresponds to location of block errors—*Yes*
Solution Description: Link problem between Nascom and SDPF
Action: Notify the Payload Operations Control Center and request a retransmission from the ground station. Request Nascom support for line checkout.

Figure 2. Sample Case

To match a new case with a case stored in the case base, a similarity assessment technique must be defined. In the prototype, the similarity between two cases is calculated by generating a score that indicates the normalized sum of the number of features that match between a new case and a case stored in the case base. Features that describe the symp-

toms leading to a problem are used in generating this score.

Figure 3 illustrates a sample prototype screen. At the top of the figure, an analyst has entered the characteristics of a current acquisition session. All of the closely matching cases retrieved from the case base are displayed at the bottom. Each line contains a score that indicates the degree of match between the current case and a stored case, the name of the matching case, and a brief description of the problem causing the anomaly. An analyst may retrieve a stored case from the case base and compare it to the case describing the current situation.

We currently use manual adaptation. If no exact matches are found, an analyst reviews the cases provided to see what other analysts have done in the past and decides if any of the proposed solutions are applicable to the current situation. If this is a new problem, an analyst may build a new case by entering the characteristics of the new problem, including the proposed solution. Later the solution may be verified or changed to a better solution, other incorrect solutions that were tried and discarded may be added, or alternate suitable solutions may be added.

Tool Chosen

The prototype was developed using the ESTEEM CBR tool, developed by Esteem Software Incorporated. ESTEEM is a

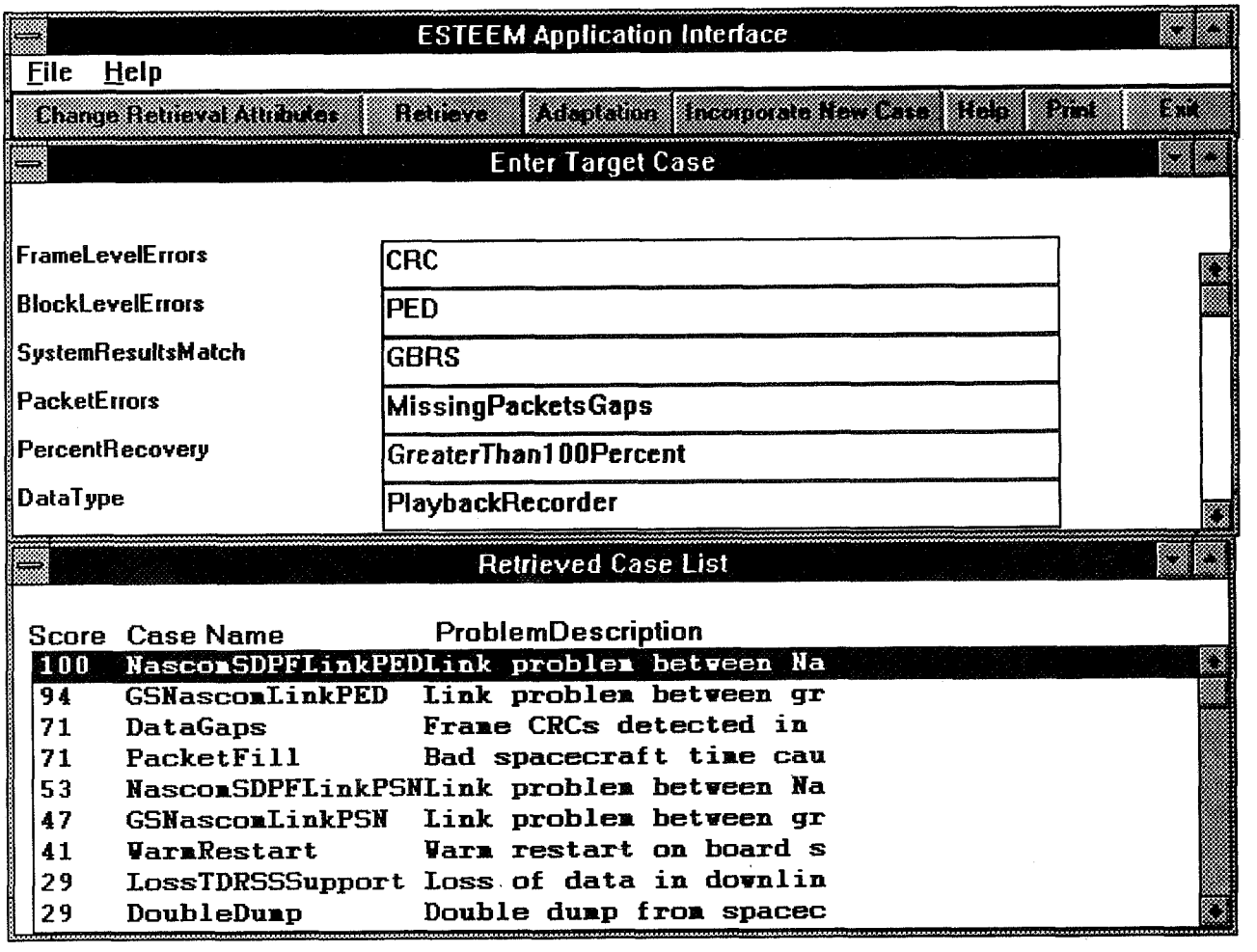


Figure 3. Sample Screen

standalone tool that runs on an 80486 IBM-compatible PC with 16 megabytes (optimal, 4-megabyte minimum) of memory, 5 megabytes of hard disk space, and a VGA monitor.

Future Issues

A major result of prototyping was to uncover issues that must be addressed in subsequent work. During maintenance in the operational environment, many analysts will have access to the case base. It needs to be determined if all analysts or if only the most experienced analysts will be permitted to add new cases to the case base. Also, it is very likely that analysts will have differences of opinion concerning the correct problem resolution. It needs to be determined whether all possible solutions or the most popular solutions will be added. Having alternatives could prove to be useful for situations where a close match is not found and an alternative solution is more suitable.

It is expected that in the operational environment, cases will evolve over time. A solution that an analyst initially thinks to be good could turn out to be in error, or an alternative solution may be better. The CBR system must be capable of evolving through this process.

For the prototype, we defined a set of features that describe the characteristics of the problem, the recommended solution, and the actions for handling the problem. For subsequent prototyping efforts, we need to determine if this set of features is suitable for all types of problems that analysts typically handle and for new, not-yet-encountered Pacor II problems. We need to determine if other information might be useful, such as other solutions tried that proved inadequate, additional background information or definitions for the inexperienced analyst, diagrams on how to fix a problem, and steps to follow to uncover the problem. A small analyst team has provided the expertise to build our initial prototype. The prototype must be evaluated by other analysts.

Because the Pacor II environment is UNIX based, we plan to port the prototype to the UNIX environment. The operational system will run as a tool for analysts who will extract feature values directly from the Pacor II database to minimize operator input. The final system will generate trouble reports automatically following an evaluation. Subsequent efforts will also include extending the case base and upgrading the computer-human interface.

Conclusion

This prototyping effort represents a novel approach to solving the troubleshooting problem using CBR. With advanced technologies such as expert systems, more automation can be introduced into operations, thus reducing life-cycle costs. Expert systems have been developed to handle troubleshooting using the rule-based approach. However, due to some of the unique characteristics of the Pacor II environment, the requirements of operations analysts, and the shortcomings of rule-based systems, an alternative approach was tried. This paper describes an initial proof of concept for the troubleshooting problem using CBR. A significant result of prototyping has been to confirm our hypothesis—we feel that this approach is a viable one for the troubleshooting problem.

Acknowledgments

The author would like to thank analysts Tony Walsh, Wende Peters, and Mark Hilliard for their contributions as experts during development.

References

Barletta, Ralph (August 1991). An Introduction to Case-Based Reasoning. *AI Expert*, Volume 6, Number 8.

Kolodner, Janet (1993). *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann Publishers, Inc.

N95- 17542

111127

354047 P. 7

AN INTELLIGENT AUTOMATED COMMAND AND CONTROL SYSTEM FOR SPACECRAFT MISSION OPERATIONS

A. William Stoffel
Human Performance Studies
NASA, Goddard Space Flight Center,
Mission Operations Division

ABSTRACT

The Intelligent Command and Control (ICC) System research project is intended to provide the technology base necessary for producing an intelligent automated command and control (C&C) system capable of performing all the ground control C&C functions currently performed by Mission Operations Center (MOC) project Flight Operations Team (FOT). The ICC research accomplishments to date, details of the ICC and the planned outcome of the ICC research, mentioned above, are discussed in detail.

INTRODUCTION

Beginning this year and extending into the foreseeable future mission operations personnel are being required to operate more complex ground systems with less flight operations team (FOT) personnel and lower budgets than in the past. The Intelligent Command and Control (ICC) system research is intended to provide the technology base necessary to solve these problems through automation and intelligent machine Case-Based reasoning and decision making. The need for the ICC is due in some cases to the fact that FOTs will be asked to command and control (C&C) more complex missions such as those of the Earth Observing System (EOS) and in others to the fact that FOTs will be required to operate several spacecraft concurrently from the same Mission Operations Center (MOC), such as in the case of the Small Explorer (SMEX) and the International Solar and Terrestrial Physics (ISTP) missions. These facts require that we develop an intelligent C&C system which is capable of acting as a cooperative assistant to the FOT, reduce the workload of existing FOTs, and reduce the cost burden of creating ever larger FOTs.

DEFINITION

The Intelligent Command and Control (ICC) System is designed to ultimately produce the technology necessary for development of a highly intelligent automated machine based C&C for Spacecraft mission operations which is capable of performing all the C&C functions currently performed by FOTs. While that is the ultimate goal, it should be noted that many very valuable interim products are being produced and will be produced which are and can be used to improve, automate, and reduce the cost of MOC operations.

This project was originally planned as a five year research project but, while interest in the ICC research is very high in the Space Ops and process control communities, funding has been halved and therefore the end-point of the ICC project is now 8-9 years out from the original start point of April, 1993.

A detailed description of the technology involved is provided later in this paper.

Program Objectives

The following are the objectives of the ICC research and development program:

1. To demonstrate that we can improve and simplify spacecraft MOC command and control by building and operating a real time Intelligent Command and Control (ICC) system utilizing AI, object oriented techniques, & animated graphical user interfaces.

2. To create a command and control system that can act as a cooperative member of an FOT.

3. To demonstrate that Mission Operations Center (MOC) Command and Control functions can be fully automated and that such a system can perform intelligent machine based decision making.

4. To demonstrate that such a system would show tremendous savings in both development and operating costs by:

* Limiting or reducing the number of FOT personnel.

* Intelligently automating spacecraft MOC functions to the point where management by exception can become a reality.

* Reducing operator error through more intuitive user interfaces, automation, the use of true machine decision making, and the application of standardized commands.

Technical Approach

The technical approach we have chosen to accomplish these objectives is as follows:

1. Establish a collaborative activity among the Mission Operations Division's (MOD) technology and operations groups, academia, and private industry.

2. Survey and evaluate existing advanced technology products available for possible use in the ICC.

3. Select and use an existing command and control system as a baseline with which to compare the ICC.

4. Prototype & evaluate the ICC using reiterative validation and development techniques.

5. Perform a side by side evaluation of the ICC and the baseline C&C.

Completion of ICC Research

Successful completion of the ICC research project is defined as completing a successful side by side test of a working ICC prototype and the baseline C&C. The comparison of the baseline C&C with the ICC will involve five steps:

1. Collecting data on the current or baseline C&C.

2. Turning off the baseline C&C and taking over all C&C functions with the ICC prototype for at least one pass.

3. Collecting data on the ICC prototype.

4. Turning off the ICC prototype and returning command and control to the baseline C&C.

5. Steps 1 through 4 above will be repeated until sufficient data on the performance and reliability of the ICC prototype has been collected to establish the results and conclusions of the ICC research.

Significance and Benefits

The following benefits potentially apply to all future NASA missions. Specific and strong interest in the ICC research, its results and products have been received from the following projects and organizations: ISTP, SMEX, Hubble Space Telescope (HST), EOS, and the Network Management and Operations Support (NMOS) Flight Projects Support Division, and the European Space Agency (ESA).

Expected benefits of the ICC research are:

1. Reducing operator error through more intuitive user interfaces, automation, and selection of standardized commands.

2. Lowering system supervisory costs through the use of management by exception.

3. Limiting or reducing the number of FOT personnel.

4. Faster, more cost effective and robust spacecraft system status, and operations simulators and models.

5. Simplified and reduced cost of training through the use of a command and control system which is both more generic, or standardized, for all missions, and internally more flexible (i.e. easier to modify for specific missions).

Accomplishments

Accomplishments to date are as follows:

1. Completed technology survey.

2. Completed 1st Transportable Payload Operations Control Center (TPOCC) Task Analysis (SMEX).

3. Completed ICC Prototyping Plan .

4. Completed Operator Function Model.

5. Completed initial ICC MOC Simulator which accepts actual TPOCC data as input .

6. Completed Task Analysis of Anomaly Detection and Correction Processes.

Deliverables and Future Accomplishments

The deliverables and accomplishments expected for the remainder of the ICC research project are as follows:

FY '94

7. Develop Case-Based+ Reasoning Tools for ICC.

8. Develop Advanced Tutor-Aid Paradigm for use in ICC (described below).

9. Complete Automation Analysis for implementation of control center management by exception.

10. Complete Second Task Analysis (ISTP).

11. Complete initial, basic research ICC Prototype.

FY '95

12. Conduct reiterative redesign and reevaluation of basic ICC prototype.

13 Complete detailed architecture (both structural and functional) of the ICC Inference Engine.

FY '96

14. Construct robust ICC MOC Simulator.

15. Begin construction of ICC inference engine.

FY '97

16. Complete Construction of ICC inference engine.

17. Conduct reiterative Integration and Testing (I&T) of ICC components.

18. Assemble ICC components into robust ICC prototype.

FY '98

19. Conduct reiterative I&T, evaluation, and redesign of complete robust ICC prototype.

20. Conduct sided-by-side test of ICC and baseline C&C.

Item Twenty (20.) marks the end of the research phase of the ICC project.

Technology Description

Functional Description:

Downlink Telemetry Handling:

The completed operational ICC when fully integrated into MOD operations will reside in the TPOCC workstation accepting data from the Front End Processor (FEP) Data Server Task (DST) and consist of the following: An

intelligent object oriented command and control system capable of accepting downlink telemetry in real time, and passing the telemetry (or database) updates to the ICC Reasoning Machine (RM). The RM, or inference engine using case based, and most probably a combination of AI machine reasoning techniques, will match the input with robust spacecraft and ground control system models/simulators and then decide what actions should be taken based on that information. The ICC will decide whether these actions are to be taken by the ICC directly, sent to the human operators (FOT) for further action, sent to other ground control systems (e.g., small Generic Systems Analyst Aid [GenSAA] built expert systems), or other users (e.g., Primary Investigators, or subsystem engineers) . What actions the ICC takes can be preset by the FOT, have default settings or be based on previous cases or extrapolations from such cases.

Uplink Commanding:

The Command side of the ICC will be capable of acting cooperatively as another member of the FOT. It will be capable of accepting and sending commands in real time, from a number of sources: default routine commands set by the FOT prior to the mission, commands set by the FOT for a given pass, Reasoning Machine ordered commands sent in response to electronic input. Whatever the source of the commands, it is currently envisioned that they will be converted from either operator graphically generated commands or RM generated commands into the Systems Test and Operations Language (STOL) commands that will be processed by the existing STOL Processor. That is our current plan, although we may find that the ICC can bypass STOL and go directly from machine generated commands or human graphically generated commands to a lower level language.

User Interface Description:

The user interfaces (UI) will be, mostly, graphical animated user interfaces. The guiding principle behind any UI design and the first question which will be asked in designing each user interface will be "What type of user interface most enhances task (and thereby mission) performance?" Therefore some user interfaces will be two dimensional graphical animated interfaces (such as those currently used in the operational Visually Inspectable Tutor and Assistant [VITA] training system [Chu, 1991]). Others will be real time interactive 3D graphical animated interfaces (such as those being developed for the 1997 Hubble Space Telescope (HST) Servicing Mission), some will use voice interactive interfaces, and still others will be alpha-numeric command line interfaces. The idea is to apply the most effective type of interface for the task to be accomplished and this will be determined by a reiterative process of prototyping and prototype evaluation using FOT personnel to conduct the evaluations.

Detailed Description of Conceptual Deliverables:

The following descriptions and discussions are derived primarily from work conducted under a NASA grant by Dr. Christine M. Mitchell of the Georgia Institute of Technology (Mitchell, 1994).

SAMPEX Operator Function Model:

The Operator Function Model (OFM) is a hierarchical-heterarchical decomposition of the FOT functions required to carry out real-time operations involved in satellite ground control. The OFM provides a detailed normative model specifying how operations are intended to be carried out. The OFM is hierarchical. At the highest level it specifies the components that comprise the overall real-time operations: pre-pass, on-pass, and post-past. It decomposes each function into its component activities that may be mapped to lower levels including sub functions, and tasks. At the lowest level, the OFM specifies operator actions, both, manual (e.g., issue this command) and cognitive (e.g., check the current state of the power subsystem) needed to carryout individual tasks. The OFM is both heterarchic and dynamic. Its components depict the concurrent activities typical of satellite ground control (e.g., execute and monitor a command to ensure that it is properly carried out at the same time as running procedures to up-load another command). The dynamic component provides the context:

triggers represent how new operator activities manifest themselves as a result of system events and previously executed operator actions.

SAMPEX Task Analysis of Anomaly Detection and Correction Processes:

This analysis is intended to understand how often and what happens when unanticipated events and anomalies occur. The study addresses events that occur post launch and early orbit (L&EO), i.e., examination of those events that are considered to have occurred during the SAMPEX nominal operations phase. In particular, the study documents for each anomaly (other than those identified by one of the SAMPEX experts as a peculiarity of the L&EO) the process of 1) failure detection (i.e., when, how, by whom was the anomaly first noticed?); 2) failure management (i.e., how long, and what happened, between the time when the anomaly is first detected, and when corrective action is initiated); 3) fault compensation (i.e., what was done, who did it (with emphasis on the decision maker's qualifications, e.g., spacecraft analyst, command controller). The study will include identification of time required to resolve the anomaly and distribute information to the FOT. This study will be coordinated with the SAMPEX OFM, particularly with respect to the issue of non-preplanned activities. Recall, the OFM will include comments on what actions are pre-planned (always, usually, sometimes), opportunistic (i.e., planned and executed on the fly without inclusion in the pass plan). In the latter case we will attempt to document the types of opportunistic activities undertaken and the personnel who formulate and execute them (e.g., lead analyst, spacecraft engineer).

Case-Based Reasoning for Real-Time Ground Control Operation:

Building a Knowledge Base of Experience of Real-Time Decision Making:

This component of the ICC project will investigate the use of case-based reasoning technology to accumulate a knowledge base of actual operations experiences and, subsequently, to use that experience as aid or advice in an intelligent decision support system. Initially such a system monitors real-time operations forming a knowledge base that reflects the range of nominal operations. As unplanned and/or anomalous events occur the case base grows, in fact it automatically learns, broadening its knowledge base to include operations experience accrued in managing these unanticipated events. Such a system uses case-based reasoning technology to build an extensive repository of operations experience--i.e., cases, that over time, can function as the knowledge base for an autonomous system. This project represents one of the first applications of case-based reasoning to real-time decision making and system control. It provides an alternative, and potentially richer, knowledge base than such applications as rule-based systems. Given the extent of operational experience that comprises the foundation of FOT expertise, a case-based system that can learn from skilled operators is a promising way to encapsulate and capitalize on human experience and subsequently make it available to both other operators and intelligent systems.

The Tutor-Aid Paradigm

This project builds on the highly successful VITA intelligent tutoring system as the first component of an integrated tutor-aid architecture. The tutor-aid paradigm proposes that an effective approach to operator aiding and training is the integration of aiding and training into one comprehensive system that

differentially responds depending on the skill level of the operator. An integrated tutor-aid provides a great deal of assistance and guidance to unskilled operators, i.e., operators-in-training; as the operator skills increase the tutor becomes less active and transitions into a well-understood assistant. The tutor-aid paradigm promises to be very effective. An integrated tutor-aid system is cheaper to build and maintain. Functionally, a versatile and intelligent tutor is likely to evolve into a well-understood and trusted aid. The knowledge bases that support an intelligent tutor-aid system (e.g., system and task models of what to do, how, and when) are exactly those needed for more autonomous system operation and control.

ICC-TPOCC (A Real-Time Simulator of the Operator Interface to TPOCC-Based Ground Control Systems):

Another component of the ICC project is the development of a research/experimental testbed, the ICC-TPOCC testbed. In addition to research concepts exploring intelligent systems for operator aiding and training, the ICC project is concerned with proof-of-concept demonstrations and evaluations of these technologies. Long term, the intent is to provide a side-by-side demonstration comparing conventional operations with operations incorporating the proposed aiding systems. In the interim, the individual research efforts can be demonstrated and empirically evaluated in the context of the ICC-TPOCC testbed. The ICC-TPOCC testbed is a real-time simulation of the operator interface to the satellite ground control system. It is modeled after the SAMPEX TPOCC mission operations center operations. The testbed provides the ICC project with the ability to implement the proposed system, and using NASA operations personnel as subjects, conduct experiments that compare current and proposed systems.

Automation Analyses:

Two studies comprise the final component of the ICC: an in-depth analysis of the feasibility of a completely autonomous control center and a statement of working assumptions that underpin the belief that an autonomous control center is possible. The feasibility study will examine the existing facilities and procedures integral to satellite ground control, specifically focusing on impediments to a completely autonomous control center (why are operator's needed and what do they do). As impediments to intelligent automation are identified, the study will attempt to suggest technological alternatives to the impediments. The sets of impediments and technological alternatives define the basis of the second study. This study will articulate a set of working assumptions that define the operating practices (current or needed in the future) essential to moving to fully autonomous ground control operations.

State of the Technology

Current technology in operational use employs windows and some point and click interfaces but is still highly tied to alpha-numeric command line and telemetry display technology. Very little artificial intelligence (AI) and animated real time graphics is built into any of the current operational command and control systems.

The technical challenges to developing the ICC lie, first, in the area of developing the most intelligent inference engine possible, second, in determining the most intuitive and cost effective graphical animated user interfaces. The third area is that of developing robust spacecraft simulators/models. The fourth technical challenge is that of integrating the ICC with the TPOCC systems.

Research Team

ICC Project Manager:

A. William Stoffel, Human Performance Studies, Code 513.1, NASA, Goddard Space Flight Center, Greenbelt, MD

Team Members:

Dr. Christine M. Mitchell, Center for Human-Machine Systems, School of Industrial and

Systems Engineering, Georgia Institute of Technology, Atlanta, GA

Dr. Patricia M. Jones, Dept. of Mechanical and Industrial Engineering, University of Illinois at Urbana-Champaign, IL

References

Chu, R., (1991, September). Towards The Tutor/Aid Paradigm: Design of Intelligent Tutoring Systems for Operators of Supervisory Control Systems. Doctoral Dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

Jasek, C., & Jones, P. M., (1994, June). Modeling and Supporting Cooperative Work in Mission Operations: The Development of the ISAM System, Semi-Annual Progress Report, NASA Grant NAS-5-2244. Report ICC-UIUC-9406, Engineering Psychology Research Laboratory, Dept. of Mechanical and Industrial Engineering, University of Illinois at Urbana-Champaign, 1206 W. Green St., Urbana, IL.

Mitchell, C. M., (1994, April) Intelligent Command and Control Systems for Satellite Ground Operations, Semi-Annual Progress Report, NASA Grant NAG-5-2227. Center for Human-Machine Systems, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

Operations

5. Orbit Determination

** Presented in Poster Session*

PRECEDING PAGE BLANK NOT FILMED

PAGE 764 INTENTIONALLY BLANK

N95-17543

111128

354051

DSN CO-OBSERVING OPERATIONS TO SUPPORT SPACE VLBI MISSIONS

Valery I. Altunin
Thomas B. Kuiper
Jet Propulsion Laboratory

Pamela R. Wolken
Allied Signal Technical Services Corp.

ABSTRACT

Reliable radio astronomy support of Space Very-Long-Baseline-Interferometry missions by ground radio telescopes is mandatory in order to achieve a high scientific return from the missions. The 70m DSN antennas along with other ground radio telescopes will perform as the ground segment of the Earth-Space interferometer.

Improvements of radio astronomy VLBI operations at the DSN to achieve higher reliability, efficiency, flexibility and lower operations costs is a major goal in preparing for radio astronomy support of SVLBI. To help realize this goal, a remote control and monitoring mode for radio astronomy operations at the DSN is been developed.

1. INTRODUCTION

Two Space Very-Long-Baseline Interferometry (SVLBI) missions are to be operational during the second half of the 1990's. The spacecrafts and Space Radio Telescopes (SRT) will be designed, manufactured and launched by the Japanese (VSOP) and Russians (Radioastron).

In addition to the flight elements, the network of ground radio telescopes which will be performing co-observations with the SRTs are essential to the mission. Observatories in 39 locations around the world are expected to participate in the missions [1,2]. They should provide co-observing support with detection of signals from celestial sources in L,C, K-bands for VSOP and Radioastron, and additionally P-band for Radioastron, two circular polarizations at each channel and recording of signals in VLBA/MKIV compatible formats.

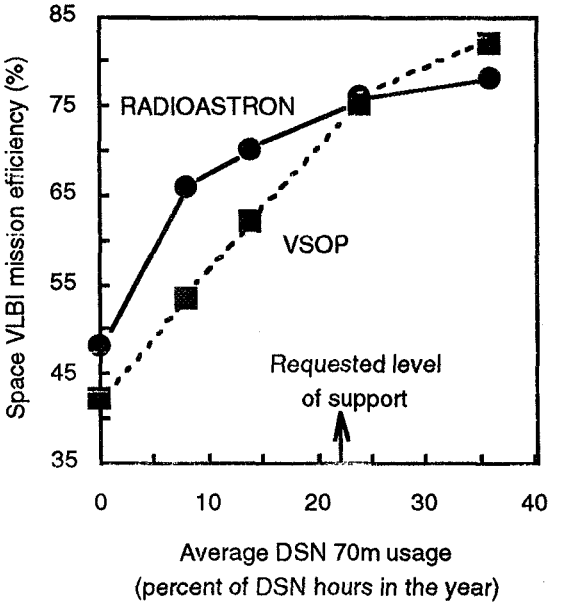
The 70m DSN antennas along with other ground radio telescopes will perform as the ground segment of the Earth-Space interferometer. DSN radio astronomy co-observations for future Space VLBI missions will play a special role due to the performance of the facilities (longest baselines, co-location with spacecraft data acquisition and phase link stations, 70m class of antennas with 22 GHz antenna efficiency up to 40-50%), and the inherent reliable operability of the DSN which is oriented to supporting routine operations (daily for 3-5 years).

The importance of DSN co-observing support for SVLBI missions is recognized by DSN management as evidenced by their preliminary allocation of DSN 70m

PRECEDING PAGE BLANK NOT FILMED

PAGE 766 INTENTIONALLY BLANK

time in their long-range resource allocation plan. The value of the DSN 70m network to SVLBI missions' efficiency is illustrated by Fig1. (Courtesy of D.Meier, JPL). This figure shows an estimation of SVLBI mission efficiency (percent of time per orbit actually used for observations by a SRT) vs average DSN 70m usage for co-observing with a SRT. The change of efficiency for a SVLBI mission can be significant due to a change in the level of the DSN co-observing support [3].



Preliminary consideration of the DSN 70m co-observing requirements and cost estimates for corresponding upgrades of the DSN systems did show that upgrading the existing DSN capabilities is the only way to keep the cost upgrades at a reasonable level and satisfy minimal requirements for SVLBI mission co-observing support. Another condition is to accept a lower than is usually used for s/c operations reliability of DSN operations to maintain radio astronomy support for SVLBI.

Three main areas of activities are under development to upgrade DSN VLBI radio astronomy performance and to provide adequate and reliable co-observing support:
- improvements of the current VLBI Radio Astronomy operations;
- renovation of radio astronomy receiving systems and upgrade of the MKIII to MKIV VLBI recording system;
- testing equipment and training operations personnel.

Some of these upgrades are part of an ongoing improvement of DSN radio astronomy capabilities. Others are specific to the SVLBI missions.

The purpose of this paper is to describe ongoing improvements of the current VLBI Radio Astronomy operations at the DSN in order to meet SVLBI co-observing requirements.

2. DSN OPERATIONS CONCEPT TO SUPPORT SVLBI RADIO ASTRONOMY CO-OBSERVATIONS

Improvements in VLBI Radio Astronomy operations at the DSN to achieve higher reliability, efficiency, flexibility and lower operations cost is one of the major goals in preparing for DSN co-observing support of SVLBI. These improvements will also resuly in major advancements in the DSN's support of other radio astronomy activities.

Radio astronomy co-observing support for SVLBI is very similar in structure and content of the

observing sessions to Radio Astronomy and Special Activities (RASA), but the volume of SVLBI co-observing activities is expected to be a few times more (yearly average) than the regular volume of RASA activities at the DSN.

Because of this, it is logical to improve the operations performance of existing DSN VLBI radio astronomy activities to meet requirements for SVLBI co-observing.

2.1. SVLBI co-observing concept

The required operations reliability for the DSN 70m antennas serving as radio telescopes in support of SVLBI is 90-95%. The SVLBI projects (VSOP and Radioastron) will provide the schedule for observations (DRUDG file) one month in advance, but in cases of "Targets of Opportunity," the telescope has to be able to change its configuration and support a new program for observations in three days.

Essential improvements in hardware to be used for co-observing are needed: use more reliable equipment, (e.g., instead of masers use HEMT LNA), provide spares, backup receivers and recorder, improve status of monitoring and calibration. Flexibility in operations can be provided through fast and simple ways to change operations configurations and modes, and through the standardization of operations procedures.

The goal of significantly improving operations performance without increasing the cost of operations can be achieved by reducing the amount of hands-on activity and automating routine activities as much as possible. Since the largest component of operations costs is the staff, by introducing automated and remote operations the costs can be lowered [4].

2.2. VLBI Radio astronomy operations functions and operations scenario

Existing VLBI Radio astronomy operations functions performed at the DSN, excluding the time allocation on the DSN, are listed in Table1.

The proposed improvements include:
(a) automatically processing DRUDG files (VLBI radio astronomy schedule files) received from the SVLBI project via Internet to DSN Predicts;
(b) remote monitoring and control of receivers (K, L, C-bands) by using dedicated Radio Astronomy computers connected with a computer at JPL via Internet at each DSN site;
(c) capability for remote monitoring of the antenna position and recorder status;

(d) station personnel will perform the initialization, calibrations (Antenna Gain Curve, Tsys) and tape logistics.

Radio Astronomy operations at the DSN are working toward an automated and remotely-controlled configuration such as is shown in Figure 1. As this capability develops, it may be an attractive resource for future SVLBI co-observing support possibilities.

VLBI radio astronomy operations functions at the DSN		
Functions		Staff
Predicts	DRUDG to Antenna predicts	NOA VLBI
	DRUDG to VLBI recorder predicts	NOA VLBI
	DRUDG to Briefing Message	Network Operations Project Engineer for RASA
Control	Antenna configuration*	Deep Space Complex operations staff
	Antenna pointing	Deep Space Complex operations staff
	VLBI Recorder	Deep Space Complex operations staff
	Receivers	Radio Astronomy engineer
Calibration	Boresighting	Deep Space Complex operations staff
	Tsys	Deep Space Complex operations staff
	Gain curve/nonlinearity	Radio Astronomy engineer
	System coherence test	Radio Astronomy engineer
Monitoring	Antenna status	Deep Space Complex operations staff
	VLBI Recorder	Deep Space Complex operations staff
	Receivers	Radio Astronomy engineer
Tape logistics	Tapes change	Deep Space Complex operations staff
	Log file	Deep Space Complex operations staff
	Shipment/Tapes label	Deep Space Complex operations staff

*Subreflector/waveguide

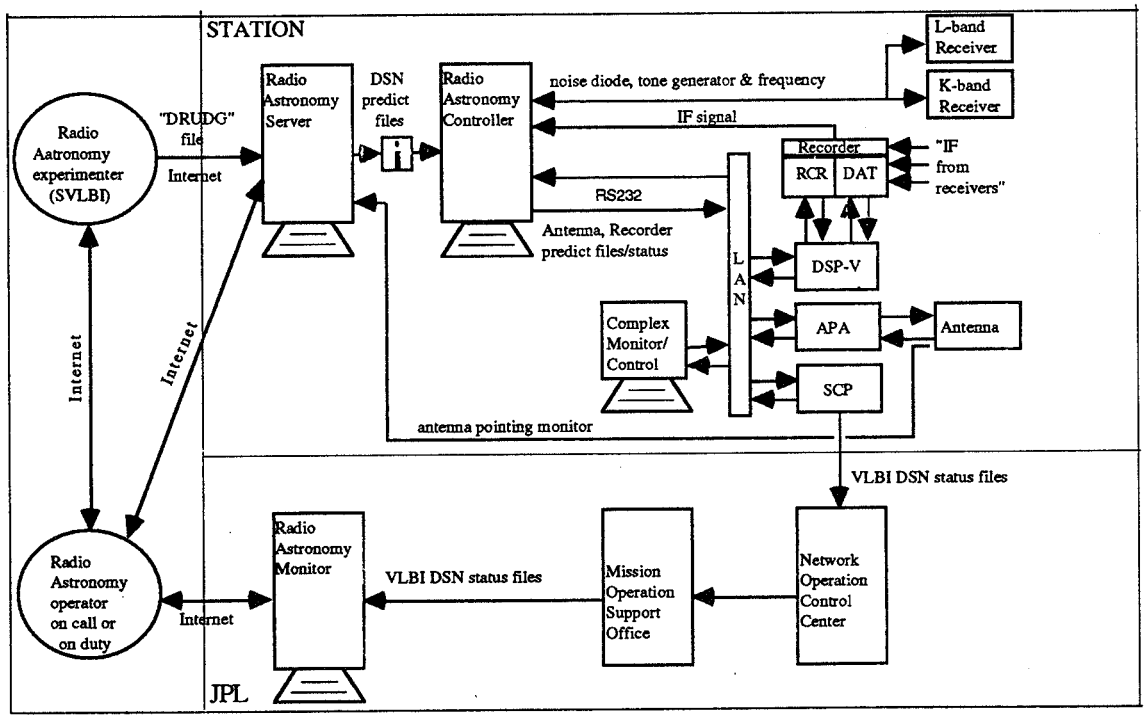


Figure 1. DSN VLBI RA operations configuration for 70m subnet

The majority of the radio astronomy community, including the SVLBI projects, in order to schedule VLBI co-observing, produce a generic scheduling file referred to as a DRUDG file. Because the DSN is used for a wider range of measurements than only VLBI radio astronomy (e.g. navigation, TM), the DSN uses its own scheduling format. DSN stations are incapable of reading DRUDG files. For this reason, someone must perform the conversion of DRUDG files to DSN predicts. SVLBI co-observing will require performing this activity much more intensively, basically every day. As result, this operation becomes very labor intensive. Automatically processing of the VLBI radio astronomy schedule files (DRUDG files), should eliminate or significantly decrease the workload to execute this function.

The Radio Astronomy Server (workstation) located at each station or in JPL, will automatically convert DRUDG files coming from the Space VLBI project to DSN predicts.

To provide security for DSN operations, it is required to have an "air gap" when information comes from outside the network it is transferred to inner network computers on diskette. The radio astronomy controller will serve as an additional filter to allow only commands which are permitted by the DSN complex. Finally, the observing program loaded on the Radio Astronomy Controller can be initiated only from the Complex Monitor and Control computer. In the future, the "air gap" may be eliminated with operations being remotely executed from the JPL control /monitor computer.

For planned SVLBI co-observations, a number of different DSN RA configurations are considered. The number of configurations is estimated to be 3 receivers x 2 polarization's x 4 recording modes = 24. An observing program may be different from day to day. An extensive automation of the control of the antenna, receivers and VLBI recorder configurations are necessary to provide reliable support without increasing of the workload of the stations personnel.

To monitor the VLBI DSN status during the observations, the necessary information will be taken from the regular flow of the DSN status information available in the Network Operation Control Center and displayed on the Radio Astronomy Monitor at JPL.

For Space VLBI co-observing, the Radio Astronomy Server and Controller may be considered as a Project resource for generating the required input files for the DSN Network Sypport System (NSS).

The station personnel will monitor activities on site during the observations for security reasons, but the automation and remote monitoring of many VLBI RA operations functions can significantly decrease the demands on the workforce thus enabling them to be shared by other projects.

Since by following the above recommendations the role of the DSN operations staff for co-observations will be minimized, more responsibility for successful observations must be assumed by the SVLBI Project. The Project

should be prepared to accept the higher probability of failures.

2.3 Implementation status

A new software for conversion of radio astronomy schedule files into DSN predicts is now under development (N.Vandenberg, Goddard SFC). The software will allow conversion of files which will arrive by the Internet to the Radio Astronomy Server atomatically and prepare the DSN predict files to use for DSN SVLBI co-observing operations.

The remote monitor control system development has been completed and its software has been successfully tested in Goldstone for 34m antenna operations (J.Leflang, JPL). The system is under development for the 70m antenna in Goldstone, and then will be implemented on other 70m DSN antennas

Dedicated RA computers (HP9000) exist at each complex. The automation of receiver control was demonstrated in Canberra DSCC. This needs to be implemented at the other complexes. It may be necessary to upgrade the computers at the other complexes to achieve full compatibility.

The monitor of data flow from DSCC via MOSO will be available on the RA computer at JPL in the near future. Software needs to be developed for the RA computer.

Antenna monitor data captured locally at each complex is available via the Radio Astronomy workstation at each complex. Software is being actively developed.

3.CONCLUSIONS
3. CONCLUSIONS

The new Radio Astronomy VLBI observations concept is under development at the DSN to provide co-observing support for future space VLBI missions. The concept is focused on a high degree of automated operations with elements of remote monitoring and control of the VLBI radio astronomy equipment.

The upgrades will benefit not only the SVLBI project but also VLBI radio astronomy and other related VLBI activities (s/c navigation, geodesy, astrometry) at the DSN.

4. REFERENCES

1. Altunin V.I., Technical Parameters of the Ground Segment and Data Management the Radioastron Project, The 2nd International Symposium on Space Information Systems, AIAA/NASA, Pasadena, CA, USA, September 17-19, 1990,
2. Altunin V.I., Robinett K.R., Mission Operations System for Russian SVLBI Mission, 43rd Congress of the International Astronautical Federation, August 28-September 5, Washington DC, 1992,
3. Meier, D.L., Contribution of Space VLBI Co-observations to the DSN's Overall Efficiency and the Importance of the DSN 70m Antennas to the Space VLBI Missions, 1994, JPL memo,
4. Wall S.D., Ledbetter K.W., Toward Lowering the Cost of Mission Operations, Proceedings of the Second International Symposium on Ground Data Systems for Space Mission Operations, Nov.16-20,1992, Pasadena, CA.

111129

N95-17544

354054

P-11

Implementation of a Low-Cost, Commercial Orbit Determination System

July 22, 1994

Jim Corrigan
Storm Integration, Inc.
2025 Gateway Place, Suite 118
San Jose, CA 95110
(408) 451-0620 FAX (408) 451-0622

Abstract

Traditional satellite and launch control systems have consisted of custom solutions requiring significant development and maintenance costs. These systems have typically been designed to support specific program requirements and are expensive to modify and augment after delivery. The expanding role of space in today's marketplace combined with the increased sophistication and capabilities of modern satellites has created a need for more efficient, lower cost solutions to complete command and control systems.

Recent technical advances have resulted in Commercial-Off-The-Shelf products which greatly reduce the complete life-cycle costs associated with satellite launch and control system procurements. System integrators and spacecraft operators have, however, been slow to integrate these commercial based solutions into a comprehensive command and control system. This is due, in part, to a resistance to change and the fact that many available products are unable to effectively communicate with other commercial products.

The United States Air Force, responsible for the health and safety of over 84 satellites via its Air Force Satellite Control Network (AFSCN), has embarked on an initiate to prove that commercial products can be used effectively to form a comprehensive command and control system. The initial version of this system is being installed at the Air Force's CEnter for Research Support (CERES) located at the National Test Facility in Colorado Springs, Colorado. The first stage of this initiative involved the identification of commercial products capable of satisfying each functional element of a command and control system. A significant requirement in this product selection criteria was flexibility and ability to integrate with other available commercial products.

This paper discusses the functions and capabilities of the product selected to provide orbit determination functions for this comprehensive command and control system.

Precision Orbit Determination System™ (PODS™)

Introduction

The Precision Orbit Determination System (PODS), developed by Storm Integration, Inc., is a workstation-based orbit determination system. PODS is layered on top of the commercially-available Satellite Tool Kit (STK)® produced by Analytical Graphics, Inc. PODS also incorporates the Workstation/Precision Orbit Determination (WS/POD)™ product offered by Van Martin Systems, Inc. The STK graphical user interface is used to access and invoke the PODS capabilities and to display the results. WS/POD is used to compute a best-fit orbit solution to user-supplied tracking data.

The Precision Orbit Determination System (PODS)™ grew out of a need to process antenna tracking data to determine a spacecraft orbit. The determined orbit can then be used to generate antenna pointing commands to control a ground antenna. Such a system is necessary for full "closed-loop" satellite command and control (i.e., from processing of telemetry and tracking data to the transmission of commands) and augments commercial command and control systems such as Storm's Intelligent Mission Toolkit (IMT)™.

PODS provides the capability to simultaneously estimate the orbits of up to 99 satellites based on a wide variety of measurement types including angles, range, range rate, and Global Positioning System (GPS) data. PODS can also estimate ground facility locations, Earth geopotential model coefficients, solar pressure and atmospheric drag parameters, and measurement data biases. All determined data is automatically incorporated into the STK data base, which allows storage, manipulation and export of the data to other applications.

PODS supports three levels of processing: Standard, Basic GPS and Extended GPS. Standard allows processing of non-GPS measurement types for any number of vehicles and facilities. Basic GPS adds processing of GPS pseudo-ranging data to the Standard capabilities. Extended GPS adds the ability to process GPS carrier phase data.

Requirements

A workstation-based capability is desired for compatibility with other workstation-based products (such as Storm Integration's IMT). The system should function stand-alone, but offer interfaces for integration with other products. A Commercial Off-the-Shelf (COTS) product approach is desirable for potential resale either alone or integrated with other command and control products. Finally, the development and certification costs must be kept low, which suggests incorporation of existing, proven COTS products in the implementation as much as possible.

Solution Approach

Storm chose two commercial products for incorporation into PODS: Satellite Tool Kit (STK)® by Analytical Graphics, Inc. (AGI), and Workstation/Precision Orbit Determination (WS/POD)™ by Van Martin Systems, Inc. (VMSI). PODS consists of these products as well as the additional code and data required to integrate the products, accept user inputs and provide output data in operationally useful formats.

Commercial Products

Satellite Tool Kit

STK is a workstation-based, interactive system for analyzing the relationships among satellites, Earth-bound vehicles, ground stations and targets. STK incorporates both text-based tables and graphics to display satellite orbits, periods of visibility, access times, and sensor coverage patterns for multiple satellites, ground stations and targets. The graphics allow animation of satellite constellations to see how sensor coverage and visibilities change over time and with orbital position.

STK allows the input of initial orbit conditions for satellites, facility and target coordinates, and Earth- and satellite-based sensor parameters via ASCII text file or Motif-based user interface panels. Output is displayed via graphical ground traces on a variety of map projections, and tables of access angles and ranges over windows of visibility. Both text and graphics output can be sent to files for printing and/or incorporation into other systems.

The STK user interface uses an object-oriented approach for defining and manipulating data. For example, a Scenario object consists of multiple Vehicle, Facility and/or Target objects. Each of these in turn may have one or more Sensor objects. Objects are created, saved, and restored separately. Data for objects are stored in individual ASCII files with pre-defined extensions (e.g., ".v" for vehicle files, etc.).

STK Programmer's Library

The Satellite Tool Kit/Programmer's Library (STK/PL)™ offers C application programmers access to the underlying functionality of the STK runtime version. The STK/PL includes header files and selected source code modules to allow programmers to develop add-on applications that are seamlessly integrated with the STK user interface, or stand-alone applications that use STK/PL as a library of functions. The STK/PL includes access to the object manager, user interface, and graphics, as well as astrodynamics libraries, time and coordinate conversion functions, and the orbit propagators. The STK/PL is written in an object-oriented manner which allows rapid modification and addition of new functionality. The PODS User Interface is being developed using the STK/PL.

Workstation/Precision Orbit Determination

WS/POD is a state-of-the-art precision orbit and geodetic parameter determination software system derived from the GEODYN II Version 8609 software used by NASA's Goddard Space Flight Center (GSFC). Van Martin Systems, Inc. has ported the GEODYN II software to numerous workstation

platforms, enhanced it in the area of GPS data processing, and packaged it as a commercially available and supported product.

WS/POD processes satellite tracking data using a Bayesian weighted least-squares data reduction algorithm and detailed environmental modeling using a Cowell-type numerical integration scheme to determine precisely various quantities related to the satellite orbit and tracking stations. Specific capabilities include the following:

Physical Models
- Atmospheric drag using the Jacchia 1971 atmospheric density model
- Solar radiation pressure
- Earth gravitation (up to 180 x 180 geopotential matrix)
- Polar motion
- Earth rotation
- Solid Earth tides
- Third body gravitation
- Earth precession and nutation
- Tropospheric refraction

Measurement Types
- Laser and radar range
- Radar range rates and dopplers (including single and double differences)
- Radar altimeter range
- Topocentric right ascension and declination
- East and north direction cosines
- X/Y angles relative to the tracking station
- Azimuth/elevation angles relative to the tracking station
- GPS pseudo-range and carrier phase, including single, double and triple differences

Parameters Estimated
- Orbit state vectors
- Parameters of atmospheric drag and solar radiation pressure
- Measurement and time tag biases
- Tropospheric refraction scale parameters
- Satellite and station clock polynomials
- Earth gravitational coefficients
- Tracking station coordinates

Algorithms and Capabilities
- Cowell-type numerical integration
- Bayesian weighted least-squares estimation algorithm
- Batch data processing
- Automatic data editing with criteria specified by the user
- Simultaneous estimation of up to 99 satellite orbits in a single run

WS/POD receives inputs and produces outputs exclusively through files. There is no user interface provided. Program control is provided by input files of 80-column card images with data in rigidly-defined column format. Data is provided and produced in ASCII text and binary files, with the file formats defined in the WS/POD documentation.

Summary

STK offers a state-of-the-art graphical user interface that has been perfected through many years of development, upgrades and customer feedback. WS/POD offers more algorithmic and data processing capabilities that any other commercially-available orbit estimation system. WS/POD also benefits from its NASA heritage, which assures that the algorithms have been tested using a wide range of operational scenarios over a span of decades

PODS Solution Approach and Features

PODS is separated into two components: PODS User Interface and PODS External Procedure (PODS/XP). PODS User Interface is implemented using STK/PL. PODS/XP is a stand-alone program independent from STK and provides a C-language interface to WS/POD. The PODS functional breakdown is shown in Figure 1: PODS Functional Breakdown and is described below.

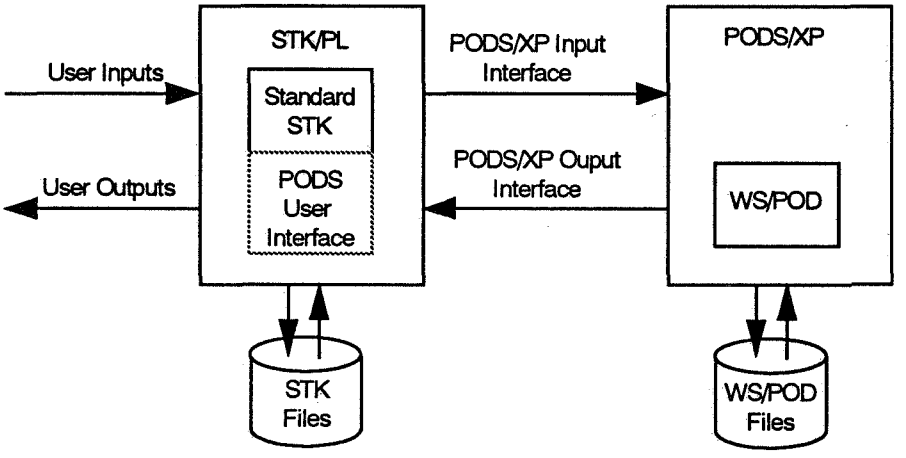


Figure 1: PODS Functional Breakdown

PODS User Interface

STK provides an object-oriented user interface in which the data applies to a selected object (either Vehicle, Facility or Scenario). PODS data is treated as an extension to the data for the existing STK object class. This allows STK to store the PODS user inputs in the STK object files and use previously-entered values as defaults for subsequent runs. This approach also allows PODS input data to be specified in the ASCII object files instead of through the user interface.

PODS operations are implemented as extensions to the existing STK operations and are invoked via the standard STK user interface. The PODS input panels are similar to existing STK panels, providing Motif pull-down menus, on-line help, and standardized range and data format checking.

Numerical outputs from PODS are displayed in standard STK output data windows, which allow scrolling through the output data, exporting to files, queuing to a system printer, and real-time units and time format conversions. Selected PODS data (e.g., ephemeris and facility locations) are entered into the existing STK data structures, allowing STK to display the data graphically and use it as the basis for accesses and other computations.

PODS External Procedure

The PODS External Procedure (PODS/XP) provides a C-language interface to the WS/POD product. It is designed to be independent from the specifics of the user interface, which allows the use of other user interfaces or calls from external applications. The interface data are consolidated in a series of structures in header files that are incorporated by the application providing the data (initially STK/PL). PODS/XP is designed such that calls to it can be made from any C program that makes use of the PODS structures.

Processing Levels

PODS provide three levels of support for users with a variety of mission requirements: Standard, Basic GPS, and Extended GPS All levels provide the STK-based graphical user interface and input/output capabilities. The different levels are licensed externally, allowing users to upgrade without re-installation of the PODS software. Each level is described in more detail below:

- Standard - Provides the capability to determine the parameters and process the measurement types listed in the section titled Workstation/Precision Orbit Determination, including processed GPS position/velocity data. Depending on the quality of the data and models used, sub-meter orbit positional accuracies are achievable.
- Basic GPS - Includes the Standard capabilities plus the ability to process GPS pseudo-range data from any number of GPS satellites and receivers. To achieve a more accurate solution using GPS data, PODS estimates the orbits of the GPS satellites based on tracking data from ground receivers rather than using the downlinked GPS navigation data.
- Extended GPS - Includes Basic GPS capabilities and the ability to process carrier phase data. Orbit position accuracies within 10 cm and ground station coordinate accuracies within 1 cm are achievable.

Inputs

This section summarizes the available inputs.

Inputs from User

PODS user inputs are provided per STK object (Scenario, Vehicle, or Facility). Scenario inputs apply to all vehicles and facilities in the Scenario. Inputs per object type are listed below.

Scenario Inputs

- Input tracking data file names and formats
- Selection criteria for tracking data by time span, measurement type, vehicle or facility, etc.
- Earth flattening coefficient
- Earth gravitational constant and sigma
- Maximum geopotential model degree and order for all vehicles

Vehicle Inputs

- Transponder delay
- Geopotential model degree and order to be used in the force model for this vehicle
- Vehicle area and mass
- Initial orbit state vector in a variety of coordinate systems and element forms (Cartesian, Keplerian, non-elliptical forms, etc.)
- Span for orbit estimation and/or propagation

- Earth gravitational model coefficients and sigma values
- Solar flux data and times
- Magnetic flux data and times
- Coordinate system reference date
- Data pass definitions
- Minimum and maximum number of iterations
- Convergence criteria
- Sigma editing criterion
- Initial RMS values
- Orbit integrator step size
- Selection of optional output reports as listed in the section titled Outputs to User
- Optional unmodeled acceleration and sigma values
- Solar pressure coefficient and sigma
- Atmospheric drag coefficient and sigma value
- Biases and sigma values for all measurement types
- Covariance matrix for initial orbit elements
- Selection of optional output files

Facility Inputs

- Minimum elevation angle before data is rejected
- Facility coordinates (in a variety of coordinate systems) and sigma values
- Coordinate system for station adjustments
- Facilities which are constrained in position relative to one another
- Earth semi-major axis and flattening overrides for geodetic conversion per station
- Antenna mounting type and displacement
- Nominal received wavelength
- Turn-around factor (ratio of wavelength transmitted to wavelength received)
- Biases and sigma values for all measurement types
- Override sigma values for normal equations and data editing
- Temperature, pressure and humidity at facility and time spans over which the data applies

Additional GPS Inputs (GPS options only)

- Names of RINEX files containing GPS tracking data
- Names of navigation files containing GPS navigation data
- Time span and/or measurement type criteria for selection/deletion of GPS data
- Radiation pressure model name for GPS orbit perturbations
- Identification of hub receivers used in construction of single differences
- Allowed tolerances between receiver times when forming differences
- Selection of optional output data

Inputs from Files

- Tracking Data Files - Files containing tracking data (formats described in PODS documentation.
- Environmental Files - Files containing Earth geopotential matrix; time system, polar motion and flux data; and planetary ephemeris.
- STK Object Files - ASCII files containing the STK and PODS data (user inputs, estimated parameters, orbit ephemeris, etc.) stored between runs.

Outputs

Outputs to User

All user outputs are displayed through the STK user interface. STK provides the ability to change display units and time systems, export data into a format suitable for use by a spreadsheet program, and send data directly to a system printer. The Mandatory Outputs are displayed during or after every PODS run, and the Optional Outputs can be displayed in addition to the Mandatory Outputs at the user's choice. The items in each output type are listed below.

Mandatory Outputs

- Tracking data summary, including:
  - Vehicles, facilities and measurements types for which tracking data exists in the selected files
  - Start and stop time of selected tracking data by vehicle, facility and measurement type
  - Number of passes
  - Time span for each pass
  - Vehicle, facility and measurement types per pass
- Convergence status (converged/diverged) for solutions
- Convergence criterion for solution
- Number of iterations performed
- List of parameters estimated
- For each estimated parameter:
  - A priori value
  - Estimated value before last iteration
  - Final estimated value
  - Difference between final and a priori values
  - Difference between final and last iteration values
  - Final sigma value
  - Final sigma value multiplied by the RMS value
  - Epoch times (for estimated orbits)
- List of STK objects updated
- Ephemeris data (including ground traces) for each estimated orbit
- New locations for each estimated facility

Optional Outputs

- Correlation and covariance matrices for solved-for parameters
- Last iteration residuals
- Number of measurements per type used in each iteration
- Summary per measurement type, including:
  - Name
  - Units
  - Total number of measurements in tracking data
  - Number used
  - RMS and mean value of both the residual and weighted residual
- RMS history per iteration
- GPS vehicle orbit elements (GPS options only)
- WS/POD TDF Run File
- WS/POD TDF Block Summary File
- WS/POD GDF Run File (for GPS options only)
- WS/POD FixClock Run File (for GPS options only)
- WS/POD CNTL Run File
- WS/POD EXEC Run File (132-column)
- WS/POD EXEC Terminal Output File (80-column)

Outputs to Files

- Solution Files - WS/POD output files saved after the PODS run. File formats are outlined in the PODS documentation.
- Environmental Files - Updates to the Environmental Files used by WS/POD.
- STK Object Files - Updates to the ASCII object files with the latest object data.

Applications

Single Satellite Maintenance

One potential application for PODS is the Air Force Satellite Control Network (AFSCN), which determines the orbit of individual satellites using azimuth, elevation and S-band range and range-rate from a world-wide network of Remote Tracking Stations (RTSs). Tracking data is generated by the stations and sent to a Mission Control Complex where an orbit estimation is performed. The new orbit is used to generate antenna pointing angles, which are in turn sent to the RTSs to drive the antenna for subsequent contacts with the vehicle.

A typical sequence of events using PODS is as follows:

- The analyst creates the vehicle in the STK database including the initial orbit estimate. This can either be the result of a previous PODS run propagated to the present time, or generated by STK using NORAD 2-Line Mean Element Set (2LMES) inputs.
- The tracking data from the RTSs are reformatted into a PODS data format. This can be accomplished using a database management system, custom program, or text formatting tool such as UNIX awk.
- The analyst produces a tracking data summary as necessary to display the types and spans of tracking data available.
- After approval of the tracking data contents, the analyst sets the estimation parameters and performs a PODS estimation run, resulting in a display of solution data and a ground trace for the new vehicle orbit.
- After examination of the output, the analyst can elect to accept the results by saving the vehicle object in STK, or can overwrite the results by reloading the original vehicle object from the data base.
- The analyst invokes the standard STK *Access* operation against the saved orbit ephemeris data to generate antenna pointing angles for the RTSs.
- After viewing the pointing angles, the analyst can export the data to a file for use in controlling an antenna in real-time.

The saved PODS results supply the input field defaults for the next PODS run for the same vehicle. The PODS-generated ephemeris data is used by other STK utilities and/or optional add-on STK products. The analyst can also at any time extend the ephemeris span of a PODS orbit by invoking the PODS orbit propagator from the STK *Vehicle/Orbiting* menu.

Automated Constellation Management

One of the powerful features of the PODS implementation is the ability to process the data for many satellites simultaneously. This allows management of entire constellations from a single workstation. The nature of the STK interface and object file storage capability allows inputs to be specified by an automatic process, eliminating the need for a user to manually enter data for each run.

As an example of such a process, consider a constellation of several dozen low-flying satellites at high inclination (as is proposed for several commercial global cellular communications networks). Tracking data for the satellites is collected by multiple ground stations around the world. A process utilizing PODS is as follows:

- Collect the tracking data for the different stations.
- Using a network management system (such as Storm Integration's IMT) perform the following:
  - Reformat into PODS tracking data types. Data from multiple stations and/or vehicles can be included in a single PODS tracking data file.
  - Automatically generate the PODS inputs and build the STK ASCII object files containing the PODS inputs per object.
  - Invoke PODS for the entire constellation. Graphical results for the entire constellation appear in STK.
  - Automatically save the estimated results for the entire constellation.
  - Use the Inter-process Communication (IPC) features of STK to automatically generate scheduling information, ground station access times and antenna pointing angles for the constellation.
- The analyst can perform periodic updates of the solar and magnetic flux information, Earth polar motion and UT1 coefficients using the PODS database management utilities, or these can also be automated.
- Manual overrides can be used at any time, entered either through the user interface or the object files.

Initial orbit estimations may require multiple passes of data in order to accurately estimate the effects of solar pressure, atmospheric drag, and the Earth gravitational field per vehicle. Longer data spans using multiple stations can also be used to precisely determine the location of the tracking stations as well as any biases associated with the measurements from the individual tracking stations. The best estimates of these parameters can be used in the automated scenario described above and can be updated at any time.

GPS Data Processing

PODS provides a variety of options for GPS data processing. The simplest option is supported by the Standard level and involves incorporation of GPS receiver point position vectors into an orbit solution. Vehicles with on-board GPS receivers generally telemeter the position vectors computed by the receiver. These position vectors can be combined with ground-based measurement types (e.g., range, range-rate, etc.) to form a single set of data for which PODS will compute the orbit that best fits the available data. The GPS receiver data can supplement ground-based measurement types, which can reduce the number and/or required coverage areas of ground stations while still achieving high accuracy. The GPS data can also be used as a reference to calibrate the ground-based receivers.

A more sophisticated approach can be supported when the on-board GPS receiver passes along the raw pseudo-range and carrier phase data. The GPS options of PODS can process these data types directly to obtain user satellite position solutions with 10 cm accuracy. Processing of pseudo-range and carrier phase data from ground-based receivers allows determination of ground receiver locations as well as orbit solutions for the entire GPS constellation with uncertainties below 1 m.

Summary

PODS combines two powerful COTS products, STK and WS/POD, into a single integrated system combining ease-of-use with high-fidelity algorithms. STK provides a modern graphical user interface and seamless integration of the estimated parameters with a wide range of existing mission planning and analysis tools. The integration with STK makes PODS a natural extension of existing STK capabilities. WS/POD provides powerful computational capabilities with demonstrated reliability due to the heritage from NASA programs. The system is designed so that it can be entirely configured by the end user with minimal assistance from the vendor.

Applications of PODS range from single satellite control to constellation management. The three different processing levels based on inclusion of different types of GPS data allow the user to choose the level of support appropriate for mission requirements. The open nature of the PODS/STK interfaces allow easy integration with existing command and control systems.

References

- Volume 1, MicroCosm® Systems Description, Version 9302.00, February 1993 published by Van Martin Systems, Inc., Rockville, MD.
- Volume 2, MicroCosm® File Description, Version 9302.00, February 1993 published by Van Martin Systems, Inc., Rockville, MD.
- Volume 3, MicroCosm® Operations Description, Version 9302.00, February 1993 published by Van Martin Systems, Inc., Rockville, MD.
- Volume 1, MicroCosm® Installation and Tutorial, Version 9302.00, February 1993 published by Van Martin Systems, Inc., Rockville, MD.
- Satellite Tool Kit® User's Manual, Version 2.0 for Engineering Workstations, published by Analytical Graphics, Inc., King of Prussia, PA.
- Satellite Tool Kit/Programmer's Library (STK/PL)™ Programmer's Guide, published by Analytical Graphics, Inc., King of Prussia, PA.
- Satellite Tool Kit/Programmer's Library (STK/PL)™ Programmer's Reference, published by Analytical Graphics, Inc., King of Prussia, PA.

111130

N95-17545

Development of a Prototype Real-Time Automated Filter for Operational Deep Space Navigation

354058

p. 5

W. C. Masters† and V. M. Pollmeier††

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

Abstract

Operational deep space navigation has in the past, and is currently, performed using systems whose architecture was originally designed to accommodate tape data transfers and computing environments with a tiny fraction of the current capability. Additionally, this architecture requires constant human supervision and intervention. A prototype for a system which allows relatively automated processing of radio metric data received in near real-time from NASA's Deep Space Network (DSN) without any redesign of the existing operational data flow has been developed. This system can allow for more rapid response as well as much reduced staffing to support mission navigation operations.

Introduction

In the past and current practice, deep space navigation operations have been relying on a system architecture that was designed for tape data transfers. The entire navigational procedure consists of processing batches of observations to correct spacecraft initial conditions and then using the corrected initial conditions to regenerate spacecraft trajectory. This practice not only requires constant human intervention but also makes it impossible to process data in an automated fashion.

In certain operational scenarios, it is desirable to recursively process data as they become available and to obtain the most current improvement on spacecraft trajectory (vice the correction on the initial conditions). Since the current software system can not serve this purpose, the development of the prototype system, which is dubbed the Real-Time Automated Filter (RTAF), is intended to fill this vacuum. The fundamental building block of RTAF is the Extended Kalman Filter [Ref. 1], which allows processing of data one at a time. The data driven feature of the system takes advantage of the architecture of the X-Windows Real-Time Display (XRTD) software [Ref. 2]. This system works recursively and each recursive step consists of the followings. A data point is first obtained and validated; then the spacecraft trajectory is propagated to the time corresponding to the data; and then the data point is used to correct the propagated spacecraft trajectory, which will be used for propagating the spacecraft trajectory when the next data point becomes available.

Interestingly, the Kalman Filter algorithm has been widely used and proven powerful in many data reduction applications including geo-satellite orbit determination. However, no utilization of any forms of the Kalman Filter has been documented in the

† Member Technical Staff, Navigation Systems Section, Jet Propulsion Laboratory
†† Technical Manager, Navigation Systems Section, Jet Propulsion Laboratory

PRECEDING PAGE BLANK NOT FILMED

PAGE 784 INTENTIONALLY BLANK

literature of deep space navigation operations. This prototype, once developed fully, may be the first such application using the Kalman Filter.

Approach

In the RTAF, the models for the spacecraft dynamics and measurements are a subset of that in the operational orbit determination software in JPL. The spacecraft dynamics include the n-body point mass gravitational accelerations, solar radiation pressure with an assumed cylindrical spacecraft geometry, a limited oblateness perturbation, and accelerations due to maneuver motor burns of finite time length. The measurement models are restricted to the coherent two-way Doppler with precision light time corrections for transmission and receiving times, as well as tropospheric delay of the radio signal. Filter parameters include the spacecraft state (position and velocity) and system parameters. Currently, the hydrostatic and wet zenith delays of the troposphere are treated as system parameters. Other examples of system parameters are solar radiation pressure and finite motor burn direction and duration.

Using the Extended Kalman Filter modeling definition, the spacecraft dynamics are modeled by first order nonlinear stochastic differential equations, the system parameters by first order Gauss-Markov process, and measurements by discrete nonlinear equations.

$$\dot{\vec{x}} = f(\vec{x}(t), \vec{q}(t), t) + \Gamma(t)\vec{\omega}(t) \quad (1)$$

$$\dot{\vec{q}}(t) = A\vec{q}(t) + \vec{u}(t) \quad (2)$$

$$\dot{\vec{y}}(t) = B\vec{y}(t) + \vec{\beta}(t) \quad (3)$$

$$z(t_k^R) = h(x(t_k^S), y(t_k^R), t_k^T, t_k^R) + \nu(t_k^R), \quad k = 1,2,... \quad (4)$$

In above equations, $\vec{x}$ is the spacecraft state vector; $\vec{q}$ is the dynamic system parameters, such as solar pressure and maneuver parameters; $\vec{y}$ is the ground system parameters, for example, the tropospheric zenith delays. For the measurement model, three times are involved, the station transmission time t_k^T, the station receiving time t_k^R, and the spacecraft transmission time t_k^S, all corresponding to the k-th data point. These times are related via precision time transformations between station time and ephemeris time as well as precision light time corrections. Statistical assumptions are the usual ones, such as the noise terms $\vec{w}$, $\vec{u}$, $\vec{\beta}$, and ν are uncorrelated and are of mean zero. Data validation is a simple minded approach currently, which is to check that each raw data point lies within a specified deviation limit. Data outside of this limit is discarded.

The data flow from NASA's DSN to the navigation workstation is accomplished via the same interface as is used with the XRTD software system (Ref. 2). This system taps into the already existing radio metric information stream. Data flows from each DSN antenna to the Ground Communications Facility (GCF) located at JPL. From this facility, the data flows to VAX computers which serve the Radio Metric Data Conditioning team, a part of the DSN's Multi Mission Navigation Team at JPL. At this point, an auxiliary data stream is created which allows the tracking data to flow from this DSN computer through a gateway machine also controlled by the DSN to the navigation operations workstation. This gateway is connected via DECNET to the DSN VAX and via TCP/IP to the navigation workstation. The direction of the data flow is exclusively controlled from the secure DSN machines and is restricted to a limited set of operations machines. Additionally no direct contact between the DSN operations computer and project computers occurs.

However, the result is that the navigation workstations receive the same Multi Mission Spacecraft Record (MMSPR) file that exists on the DSN computers with a time lag of no more than one minute. Figure 1 illustrates this data flow as well as highlights the software processes and file manipulations that occur on each machine. Initially a process named *DSNLISTEN* receives incoming data and generates individual files of data blocks (DBF's). The *SPRCREATE* process creates individual spacecraft record files (SPR's) and multi mission spacecraft record files (MMSPR's) at a predetermined schedule which is defined via the human controlled process *SPRCREATE*. The maximum frequency at which the MMSPR's are created is limited by the speed of the DSN Vax computer and is currently limited to once per two minutes. A process called *SPRNET* which runs on a DSN microVax monitors the MMSPR file on the primary machine and when it has been updated then transfers it to the navigation workstation via TCP/IP where a waiting process named *SPRD* receives the file and creates a copy of it on the navigation machine. The RTAF, then access the latest data from this file.

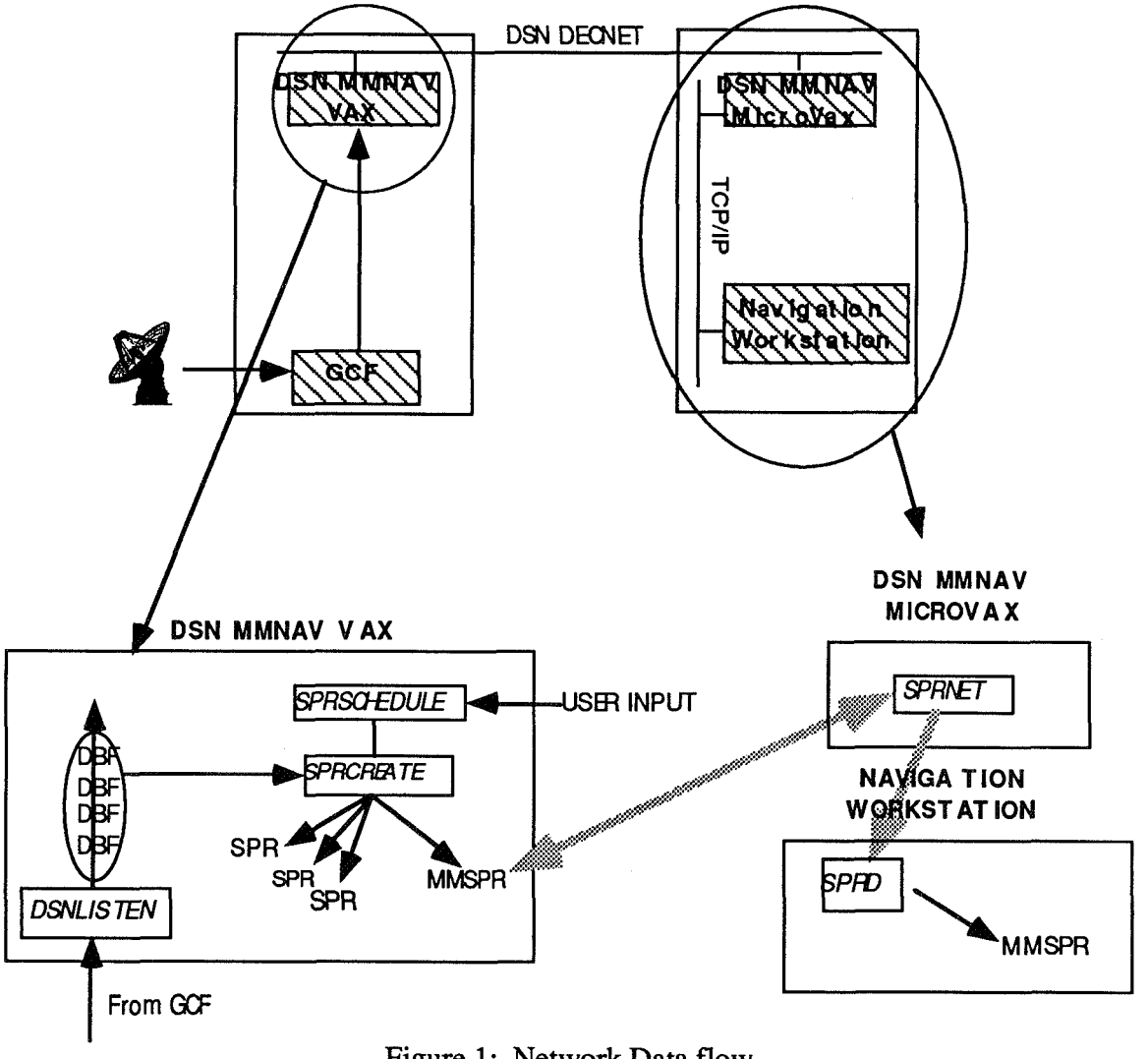


Figure 1: Network Data flow

As data flows in to the navigation workstation, the RTAF then recursively validates each data, extrapolates the spacecraft state and system parameters, computes predicted data using extrapolated state and system parameters, forms residual using the raw data and the

predicted data, and corrects the extrapolated state and system parameters. The following structure chart of the RTAF depicts this recursive process.

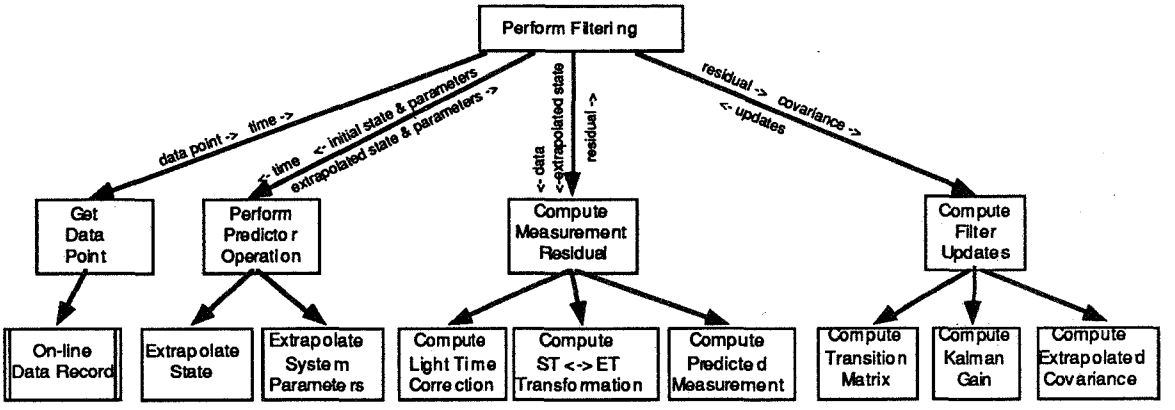


Figure 2: Filter Processing Algorithm

Conclusions and Future Plans

The RTAF represents a radically different way to perform deep space navigation operations. It has been shown to be well suited for real-time automated data processing, which would be impossible to accomplish using the traditional batch or batch sequential filter and it has high potential in autonomous navigation applications. In addition, it provides significant advantages over the traditional epoch state or pseudo epoch state formulations in its simplicity and extensibility as well as its natural way of modeling the temporal process.

Though this prototype has great promises, to be truly an operational tool, more work needs to be done. The future development will expand the spacecraft dynamic models and observable models. More sophisticated statistical methods will be incorporated in data and solution validation. Currently the system outputs a time history of changes in the estimated parameters. It is desired to have this system interface directly with one or more commercial numerical data analysis packages to allow greater data analysis capabilities. This prototype was developed in less than one year using parts of already existing systems. It is planned to develop a completely new operational tool based on this system design during the next eighteen months.

This new system will be similar in overall design to the one described here, but should provide much greater capabilities for autonomous operation as well as possible future application in on-board systems which do not use radio metric data types.

Acknowledgments

Special acknowledgment must be given to those who assisted in the development of this prototype including E. A. Rinderle, L. E. Bright, G. C. Rinker and T. P McElrath.

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States government or the Jet Propulsion Laboratory, California Institute of Technology.

References

Chui, C. K. and G. Chen, (1991). *Kalman Filtering with Real-Time Applications, 2nd ed.*, Springer-Verlag, Berlin Heidelberg.

Pollmeier, V. M., (November 1992). XRTD: An X-Windows Based Real-Time Radio Metric Display and Analysis System, *Proceedings of Second International Symposium on Ground Data Systems for Space Mission Operations*, Pasadena, California.

N95- 17546

111131

354099

Magnetometer-Only Attitude and Rate Determination for a Gyro-Less Spacecraft*

G. A. Natanson and M. S. Challa
Computer Sciences Corporation (CSC)
Lanham-Seabrook, Maryland, USA

J. Deutschmann and D. F. Baker
Goddard Space Flight Center (GSFC)
Greenbelt, Maryland, USA

ABSTRACT

Attitude determination algorithms that require only the Earth's magnetic field will be useful for contingency conditions. One way to determine attitude is to use the time derivative of the magnetic field as the second vector in the attitude determination process. When no gyros are available, however, attitude determination becomes difficult because the rates must be propagated via integration of Euler's equation, which in turn requires knowledge of the initial rates. The spacecraft state to be determined must then include not only the attitude but also the rates.

This paper describes a magnetometer-only attitude determination scheme with no a priori knowledge of the spacecraft state, which uses a deterministic algorithm to initialize an extended Kalman filter. The deterministic algorithm uses Euler's equation to relate the time derivatives of the magnetic field in the reference and body frames and solves the resultant transcendental equations for the coarse attitude and rates. An important feature of the filter is that its state vector also includes corrections to the propagated rates, thus enabling it to generate highly accurate solutions.

The method was tested using in-flight data from the Solar, Anomalous, and Magnetospheric Particles Explorer (SAMPEX), a Small Explorer spacecraft. SAMPEX data during several eclipse periods were used to simulate conditions that may exist during the failure of the on-board digital Sun sensor. The combined algorithm has been found effective, yielding accuracies of 1.5 deg in attitude (within even nominal mission requirements) and 0.01 degree per second (deg/sec) in the rates.

INTRODUCTION

The coarseness of the attitude information derived from the Earth's magnetic field, $\vec{B}$, limits the usefulness of magnetometers in accurate attitude determination systems. On the other hand, magnetic field measurements offer several advantages: (1) the sensors are inexpensive, (2) measurements can be made any time regardless of the spacecraft's orientation in space, and (3) $\vec{B}$ usually changes direction rapidly enough to make computation of its time derivative possible and these changes during the orbit are sufficiently large to enable determination of all three Euler angles using only a three-axis magnetometer (TAM).

The first and second advantages make a TAM attractive for Small Explorer missions that have modest attitude requirements. The third advantage prompts a closer look at contingency attitude algorithms that use only TAM measurements and are the subject of this paper. In fact, the third advantage allows the spacecraft rates to be computed, in principle, by examining time derivatives of $\vec{B}$.

Therefore, we address here the following nontrivial problem: Can we reliably estimate both attitude *and* rates of the spacecraft using only TAM measurements and *no* a priori information? If so, we can provide for sensor contingencies of a gyro-less

* This work was supported by the National Aeronautics and Space Administration (NASA)/Goddard Space Flight Center (GSFC), Greenbelt, Maryland, under Contract NAS 5-31500.

PRECEDING PAGE BLANK NOT FILMED

PAGE 790 INTENTIONALLY BLANK

spacecraft such as SAMPEX, as well as of a gyro-based spacecraft when the gyros are not functional.

Note that the second situation is not hypothetical. For example, the Earth Radiation Budget Satellite (ERBS) experienced a control anomaly (Kronenwetter and Phenneger, 1988, and Kronenwetter et al., 1988) during a hydrazine thruster-controlled yaw inversion maneuver that resulted in the spacecraft tumbling with rates of over 2 deg/sec. As a result, both Sun and Earth sensor readings became unreliable, and the gyro output was saturated. Similarly, control of the Relay Mirror Experiment (RME) satellite was lost after the failure of the Earth sensors (Natanson, 1992). In both cases, a TAM became the only functional attitude instrument.

We present here a combined scheme invoking two different algorithms—deterministic attitude determination from magnetometer-only data (DADMOD) and the Real-Time Sequential Filter (RTSF)—both of which have been tested successfully for SAMPEX in giving the positive answer to the above question. The DADMOD (Natanson et al., 1990; Natanson et al., 1991; and Natanson, 1992) is an algorithm that relates the time derivatives of $\vec{B}$ in inertial and spacecraft body coordinates to determine the attitude and the body rates. DADMOD has been successfully tested for ERBS under normal conditions as well as for RME after the aforementioned horizon sensor failure (Natanson, 1992).

The RTSF (Challa, 1993, and Challa et al., 1994) is a novel extended Kalman filter that estimates, in addition to the attitude, errors in rates propagated via Euler's equations. The RTSF is sensitive to rate errors as small as 0.0003 deg/sec (Natanson et al., 1993), and this feature makes it a very robust and accurate real-time algorithm. In particular, it has been shown (Challa, 1993, Challa et al., 1994) that the RTSF converges successfully in TAM-only situations using *inertial initial conditions*; i.e., the spacecraft is assumed at rest in the geocentric inertial coordinates (GCI) with its axes coinciding with the GCI axes. Note that the RTSF does not explicitly compute the time derivatives of $\vec{B}$, which are the main source of errors in the deterministic scheme.

The combined method suggested here uses the deterministic solution for initializing the RTSF to guarantee and speed up its convergence. In this scheme, the initial conditions for the RTSF are determined by the DADMOD using a 100-second batch of magnetometer measurements. The method is applied here to flight data for SAMPEX during eclipse periods. During these periods, the magnetic torquer is turned off, so that the spacecraft attitude is controlled only by the momentum wheel (Forden et al., 1990, and Frakes et al., 1992); this situation is similar to the aforementioned contingency conditions for RME. Remarkably, the accuracy of our attitude estimates is less than 2 degrees, which is within the SAMPEX requirements under normal conditions (Keating et al., 1990).

MAGNETOMETER-ONLY DETERMINISTIC ATTITUDE/RATE DETERMINATION

The deterministic scheme starts by constructing the second vector measurement from the first time derivatives of $\vec{B}$ resolved in the reference and body frames. This gives the usual transformation equations

$$A\vec{B}^R = \vec{B}^A, \tag{1a}$$

and

$$A\dot{\vec{B}}^R = \dot{\vec{B}}^A + \vec{\omega}^A \times \vec{B}^A \tag{1b}$$

where A is the attitude matrix, $\vec{\omega}$ is the angular velocity vector, and superscripts R and A imply that that the corresponding vectors are resolved in the reference and body frames, respectively. If the initial value of $\vec{\omega}^A$ is known, then $\vec{\omega}^A$ can be obtained by integrating Euler's equation, and the TRIAD algorithm (Wertz, 1984) can be used to compute the attitude matrix A from the vector pairs

$$(\vec{B}^R, \vec{B}^A)$$

and

$$(\dot{\vec{B}}^R, \dot{\vec{B}}^A + \vec{\omega}^A \times \vec{B}^A)$$

as has been done by Natanson et al. (1993). The nontrivial nature of the problem considered here arises from the unknown initial conditions for Euler's equation.

As shown by Natanson et al. (1990), the problem can be cast in the form of transcendental equations as follows. Taking into account that the vector lengths must be the same, regardless of the frame in which it is resolved, the projection $\vec{\omega}_\perp$ of $\vec{\omega}^A$ onto the plane perpendicular to $\vec{B}^A$ can be expressed as a function of an unknown angle Φ between the vectors $A\left[\vec{B}^R \times \dot{\vec{B}}^R\right]$ and $\left[\vec{B}^A \times \dot{\vec{B}}^A\right]$. The problem thus reduces to two unknown variables: the angle Φ and the projection ω_1 of $\vec{\omega}$ in the direction of $\vec{B}$, with the attitude matrix A dependent only on the angle Φ. To find Φ and ω_1, Equations (1a) and (1b) must be supplemented by Euler's equations, which can be written in the following schematic form:

$$\dot{\vec{\omega}}^A = \vec{\Omega}_0(\Phi) + \vec{\Omega}_1(\Phi)\omega_1 + \vec{\Omega}_2\omega_1^2 \quad (2)$$

where the vectors $\vec{\Omega}_0(\Phi)$, $\vec{\Omega}_1(\Phi)$, and $\vec{\Omega}_2$ are given by Equations (25a) through (25c) of Natanson (1992).* The kinematic equation relating the second derivatives $\ddot{\vec{B}}^A$ and $\ddot{\vec{B}}^R$ is then formally represented as

$$\vec{\Lambda}_0(\Phi) + \vec{\Lambda}_1(\Phi)\omega_1 + \vec{\Lambda}_2\omega_1^2 = \vec{0} \quad (3)$$

where the vectors $\vec{\Lambda}_0(\Phi)$, $\vec{\Lambda}_1(\Phi)$, and $\vec{\Lambda}_2$ are defined by Equations (23a) through (23c) of Natanson (1992).

Two nontrivial equations (transcendental in Φ) are obtained by projecting the vector equation (3) on two directions perpendicular to $\vec{B}$. One of the resultant equations is then analytically solved with respect to ω_1 at different values of Φ, and one of two roots, $\omega_1(\Phi)$ is substituted into the second equation. [The selected root $\omega_1(\Phi)$ must turn into the solution of the linear equation in ω_1, which arises in the limit $\dot{\vec{\omega}}^A \to \vec{0}$ (Natanson et al., 1990).] Finally, the resultant transcendental equation is numerically solved with respect to Φ.

REAL-TIME SEQUENTIAL FILTER

The RTSF's state vector $\vec{X}$ comprises the four components of the attitude quaternion, $\vec{q}$, and the three components of the rate correction, $\vec{b}$, to $\vec{\omega}^A$:

$$\vec{X} = \left[\vec{q}^T \;\; \vec{b}^T\right]^T \quad (4)$$

The RTSF uses sensor data to estimate $\vec{q}$ as well as $\vec{b}$, with $\vec{b}$ being estimated kinematically in the same manner as gyro biases for a gyro-based spacecraft, i.e., by attributing differences between the measured and propagated attitudes to errors in $\vec{\omega}^A$. The $\vec{b}$ estimates are then used to correct $\vec{\omega}^A$, and these corrected rates are used as initial conditions to propagate Euler's equation to the next measurement time. The propagation of $\vec{b}$ is modeled via a first-order Markov model:

$$\frac{d\vec{b}}{dt} = -\tau^{-1}\vec{b} + \vec{\eta}_b, \quad (5)$$

where $\vec{\eta}_b$ is a white noise vector, and τ is a finite time constant. The novel feature of the RTSF is that, since $\vec{b}$ represents rate errors accumulated between measurements, the optimum value for τ is the data period: 5 seconds for the SAMPEX data used here. (In contrast, the same model, when used for gyro bias estimation, requires τ of several hours.)

BRIEF DESCRIPTION OF SAMPEX

SAMPEX is the first of the Small Explorer satellites and is designed to study elemental and isotopic composition of energetic particles of solar and cosmic origin. It has a 550 × 675-km orbit with an 82-deg inclination. SAMPEX nominally is Sun-pointing and has a rate of one rotation per orbit (RPO) about the spacecraft-to-Sun vector. The

---

* Note that the cited equations erroneously used

$$I^{-1}[\vec{Y}\times\vec{Z}] = I^{-1}\vec{Y}\times I^{-1}\vec{Z}$$

instead of the correct expression

$$I^{-1}[\vec{Y}\times\vec{Z}] \equiv I\vec{Y}\times I\vec{Z}/\det I$$

where I is the inertia tensor of the spacecraft.

attitude accuracy requirement of 2 deg is achieved using a fine Sun sensor (FSS), and a TAM. The control hardware consists of a momentum wheel and a magnetic torquer assembly (MTA). During eclipse periods, the MTA is turned off, and attitude control is performed by only the momentum wheel under the assumption that the spin axis remains directed along the Sun vector.

The wheel momentum, $\vec{h}$, is directed along the body y axis, which is also the FSS boresight. The SAMPEX mass distribution is approximately symmetric about this axis. The body z axis is directed along the boresights of the science instruments.

ATTITUDE CONVENTIONS

In following Crouse (1991), the Sun-pointing orbital coordinate system (OCS) used here has its z axis directed along the target vector as it was initially defined by Flatley et al. (1990). Later McCullough et al. (1992) modified the control law, and as a result, the nominal direction of the body z axis in space differs slightly from the direction of the OCS z axis. The roll, pitch, and yaw angles are defined as the 1–2–3 decomposition of the matrix transformation from the OCS to the body frame. During the nominal 1-RPO mode, the roll and yaw angles are both close to 0, and $\vec{\omega}^A \sim (0, 0.06, 0)^T$ deg/sec, while the pitch angle may deviate from zero by a few degrees for the reason mentioned above.

The present work also uses the 2–3–2 Euler decomposition of the matrix transformation from GCI to the body frame. The advantage of this attitude parametrization during eclipse is that the third Euler angle directly reflects the 1-RPO rate of the spacecraft, while the other two angles are very nearly constant because no external control torque exists, and environmental torques acting on the SAMPEX are negligibly small.

The tests discussed below were performed using SAMPEX telemetry data for an eclipse on July 12, 1992. The truth model here is the attitude solutions from the single-frame TRIAD algorithm (Wertz, 1984), which are computed using the onboard algorithm; i.e., assuming that the Sun vector remains unchanged during eclipse.

RESULTS

Figures 1(a) and 1(b) present the first and third Euler angles for the 2–3–2 decomposition of the GCI-to-body attitude matrix, respectively. Except for the region between 400 and 700 seconds (discussed below), only two solutions are obtained, which significantly differ from each other. If attitude control is performed solely with the momentum wheel and enviromental torques are negligibly small, one can use conservation of the angular momentum to select the physical solution (Natanson, 1992). In the absense of spacecraft nutation, this implies that the first two Euler angles must remain unchanged. In fact, the first Euler angle depicted in Figure 1(a) remains unchanged for one of the two deterministic solutions and significantly varies for another. Except for the region of multiple solutions, the physical solution closely follows the straight lines of the TRIAD solution.

A similar conclusion can be drawn from an analysis of Figures 2(a) and 2(b) presenting the x and y body components of the angular velocity vector. Note that the DADMOD solutions presented here were obtained assuming constant wheel speed equal to the nominal value. Taking into account actual values from telemetry did not result in any noticeable gain in the accuracy.

More than two solutions appear when $\vec{B}$ becomes perpendicular to the pitch axis about 400 seconds after the beginning of the eclipse. Before this occured, the vector functions $\vec{\Lambda}_0(\Phi)$ and $\vec{\Lambda}_1(\Phi)$ in Equation (3) could be roughly approximated as:

$$\vec{\Lambda}_0(\Phi) \approx \vec{\Lambda}_0^0(\Phi) \equiv \vec{\Omega}_0^0(\Phi) \times \vec{B}^A = \frac{\left[I\vec{\omega}_\perp(\Phi)\left(\vec{h}\cdot I\vec{B}^A\right) - I\vec{h}\left(\vec{\omega}_\perp(\Phi)\cdot I\vec{B}^A\right)\right]}{\det I} \quad (6a)$$

$$\vec{\Lambda}_1(\Phi) \approx \vec{\Lambda}_1^0 \equiv \vec{\Omega}_1^0 \times \vec{B}^A = \frac{[I\dot{\vec{B}}^A(\vec{h}\cdot I\hat{B}^A) - I\vec{h}(\dot{\vec{B}}^A \cdot I\hat{B}^A)]}{\det I} \quad (6b)$$

where

$$\vec{\Omega}_0^0(\Phi) \equiv I^{-1}\left[\vec{h} \times \vec{\omega}_\perp(\Phi)\right],$$

$$\vec{\Omega}_1^0 \equiv I^{-1}\left[\vec{h} \times \hat{B}^A\right].$$

The approximation can be understood easily by taking into account that the magnitude of the vector $I\vec{\omega}^A$ is generally much smaller than wheel momentum. For the same reason, one can neglect the quadratic term in Equation (2). By projecting the resultant equation onto the vector $\vec{B}^A \times \vec{\Lambda}_1^0$, one then obtains the following quadratic equation:

$$a_0 + a_1 \tan\frac{\Phi}{2} + a_2 \tan^2\frac{\Phi}{2} = 0 \qquad (7)$$

which is analogous to that derived by Natanson et al. (1990) for the constant-angular-velocity limit. Obviously, this equation may not have more than two solutions. (Another advantage of this approximation is that one needs only the *first* derivatives of $\vec{B}$ with respect to time, which can be evaluated relatively accurately from a 30-second batch of magnetometer measurements.) However, the approximation made to derive Equation (7) fails if $\vec{h} \cdot I\vec{B}^A$ goes to zero, so that the vectors $\vec{\Lambda}_0^0(\Phi)$ and $\vec{\Lambda}_1^0$ become parallel regardless of the particular value of Φ. Because SAMPEX is very nearly symmetric about the pitch axis, the relation

$$\vec{h} \cdot I\vec{B}^A \approx 0$$

is satisfied in the region where $\vec{B}$ becomes perpendicular to the pitch axis. In addition, in this region

$$\vec{\omega}_\perp(\Phi) \cdot I\vec{B}^A \approx 0$$

regardless of the particular value of Φ. Consequently, $\vec{\Lambda}_0^0(\Phi)$ vanishes at any Φ, which implies that the coefficients of quadratic Equation (7) are all equal to zero. Therefore, when $\vec{B}$ is perpendicular to the pitch axis, one cannot disregard the contribution from the vector $I\vec{\omega}^A$ to $\vec{\Lambda}_0(\Phi)$. The coefficients of quadratic Equation (7) remain small for some time, making its solution completely unreliable.

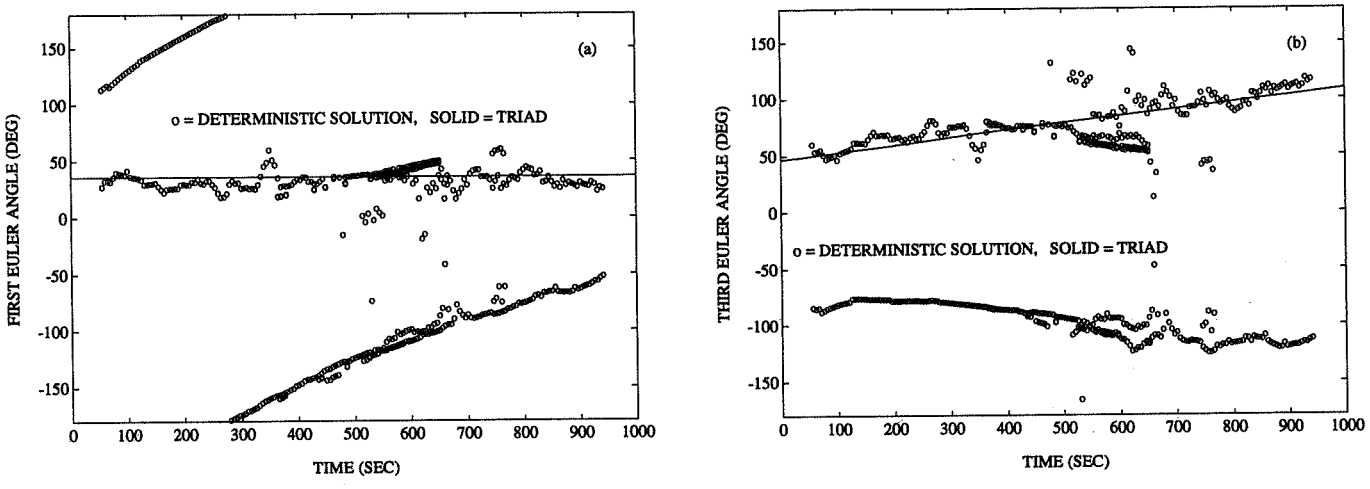
Figure 3 compares the RTSF roll and pitch angle results obtained after initializing the filter with two different schemes: the inertial initial conditions mentioned in the introduction to this paper, and the correct DADMOD solution from Figures 1 and 2. For both starting conditions, the roll angle results of Figure 3a reflect oscillations with the spacecraft's nutational period of 120 sec. The amplitude of the oscillations is a measure of the magnitude of the transverse component of $\vec{\omega}^A$ at $t = 0$. The true nutational amplitudes, however, are negligible for this data span (Natanson et al., 1993). Thus, the amplitude of the oscillations is RTSF errors and is a direct consequence of the initial rate errors.

Although the filter's rate-corrections feature enables it to converge (not shown here) after 2500 sec to within 0.01 deg/sec of the true rates even with the inertial initial conditions, it is clear that the DADMOD reduces the initial errors, as well as the convergence time, by an order of magnitude. More important, the correct DADMOD solution, by providing starting attitude and rates close to the true values, nearly eliminates the possibility of filter divergence.

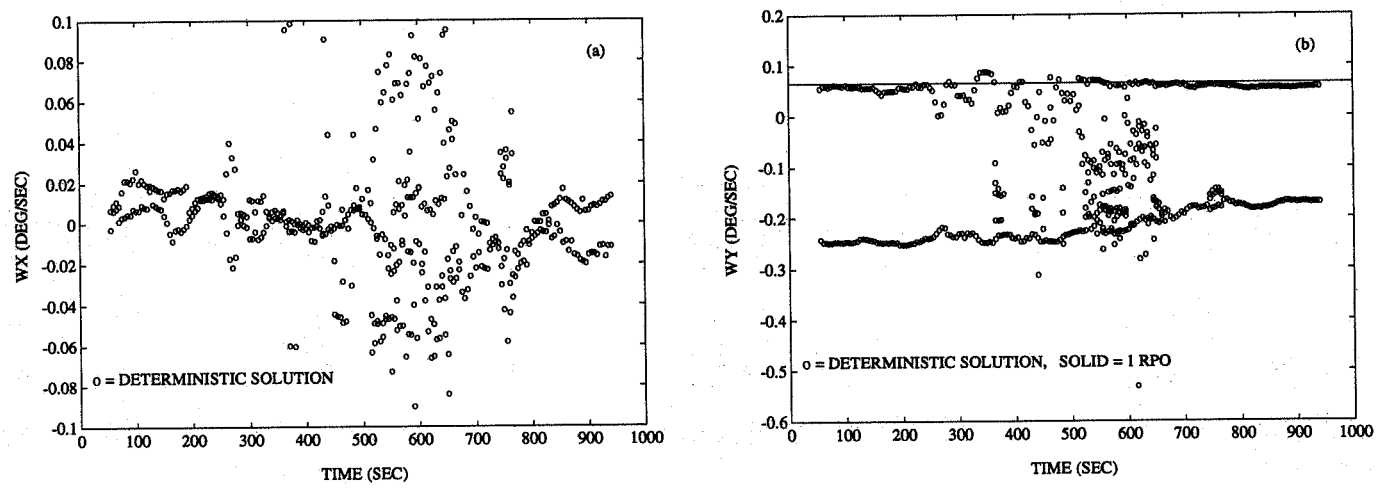
CONCLUSIONS

We find that, using only magnetic field data and no a priori information, the RTSF determines the attitude to within SAMPEX mission requirements of 2 deg and rates to within 0.01 deg/sec, respectively. Using the DADMOD to initialize the RTSF reduces the a priori errors and the RTSF's convergence time by an order of magnitude (to within a few hundred seconds) and also reduces the possibility of divergences.

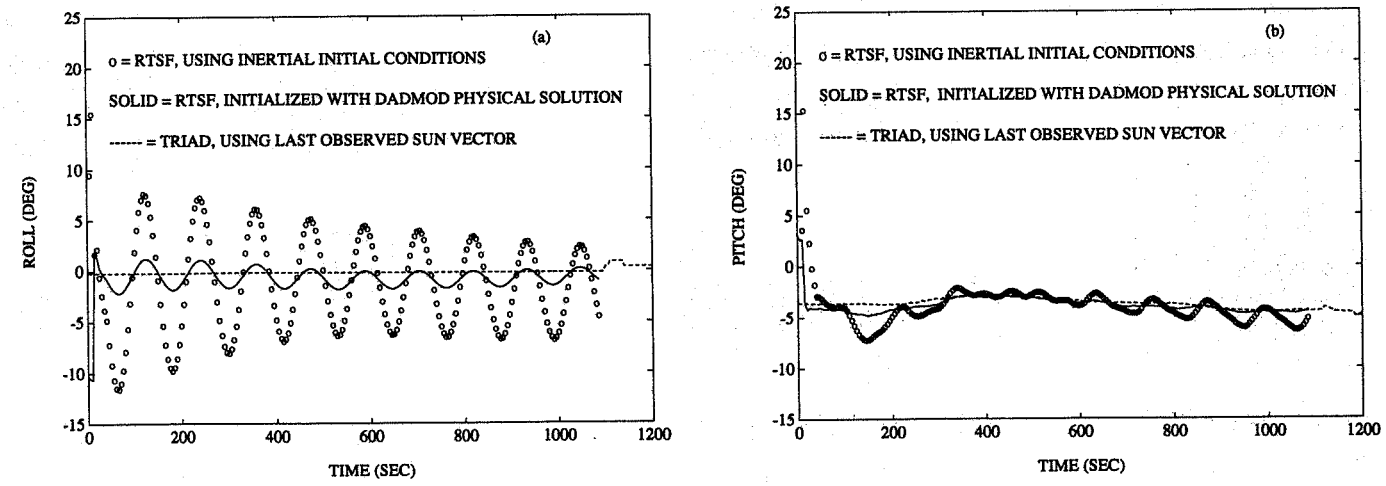
The DADMOD allows one to find the TAM-only attitude solution with an accuracy of 10–15 deg, unless the spacecraft passes through a region where $\vec{B}$ is perpendicular to the wheel momentum. The DADMOD results are consistent with those reported for the RME satellite (Natanson, 1992), where the onboard conditions after the failure of the Earth sensor are similar to those used here.



Figures-1(a) and (b). Attitude Solutions Generated by DADMOD



Figures-2(a) and (b). Rate Solutions Generated by DADMOD



Figures-3(a) and (b). RTSF Results Showing Faster Convergence Using the Correct Solutions for Figures 1 and 2

The current presentation has been deliberately limited to the case with no external torques so that the choice between physical and spurious deterministic solutions can be made by analyzing changes in the direction of the total angular momentum in space. It should, therefore, be noted that the inertial initial conditions enable the RTSF to converge in more severe conditions such as SAMPEX's Sun-acquisition mode, where the magnetic torquers are used to vary ω_x and ω_z rapidly, with amplitudes up to 0.6 deg/sec. This is shown in Figure 4 where the telemetered data span the transition (at about 2000 sec) from SAMPEX's Sun acquisition mode to the 1-RPO mode. Here, the TRIAD attitude solutions are obtained using both Sun and magnetic field data, and these are differenced to produce the TRIAD rate solutions. These TRIAD results serve as the truth model for evaluating the RTSF, which used only the magnetic field data. Despite a priori errors of up to 90 deg in attitude and 10 RPO in rates, the RTSF attitude and rate estimates converge to within 2 deg and 0.01 deg/sec, respectively, in about 1200 sec.

Therefore, the RTSF can also be used for TAM-only attitude determination in the magnetic despin mode using the magnetic field solely for the attitude control. This mode has been successfully used, for example, to despin ERBS during the control anomaly mentioned previously in this paper.

ACKNOWLEDGMENT

The authors thank J. Keat for his helpful comments on the manuscript.

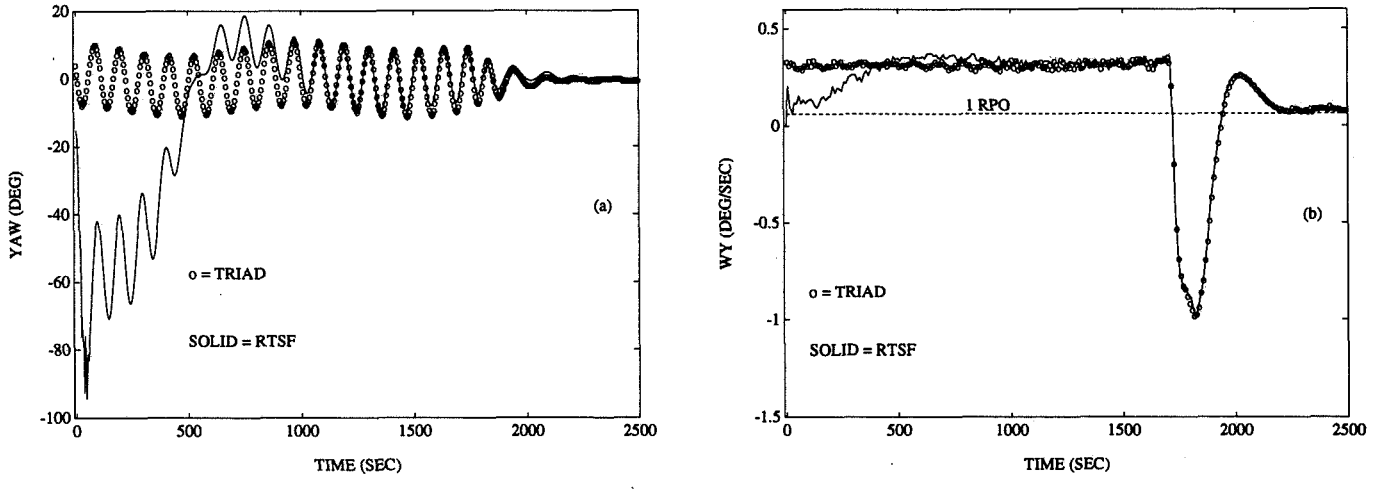
REFERENCES

Challa, M. (1993, April). *Solar, Anomalous, and Magnetospheric Particle Explorer (SAMPEX) Real Time Sequential Filter (RTSF): Evaluation Report* (553-FDD-93/024R0UD0). NASA/GSFC, Flight Dynamics Division, prepared by CSC.

Challa, M., Natanson G., Baker, D., and Deutschmann, J. (1994, May). *Advantages of Estimating Rate Corrections During Dynamic Propagation of Spacecraft Rates—Applications to Real-Time Attitude Determination of SAMPEX.* Greenbelt, MD: Flight Mechanics and Estimation Theory Symposium, NASA/GSFC.

Crouse, P. (1991, September). *Solar, Anomalous, and Magnetospheric Particle Explorer (SAMPEX) Real-Time Attitude Determination System for Personal Computer (SRTADS/PC) Specifications* (554-FDD-91/145). NASA/GFSC, Flight Dynamics Division, prepared by CSC.

Forden, J., Flatley, T., Henretty, D., and Lightsey, E. (1990, May). *On-board Attitude Determination and Control Algorithms for SAMPEX, Flight Mechanics and Estimation Theory Symposium.* NASA Conference Publication 3102.



Figures-4(a) and (b). RTSF TAM-Only Results Using Inertial Initial Conditions and SAMPEX Sun Acquisition Mode Data

Frakes, J., Henretty, D., Flatley, T., and Markley, F. (1992, May). *Sampex Science Pointing Modes With Velocity Avoidance* (Paper No. 17). Greenbelt, MD: Flight Mechanics/Estimation Theory Symposium, NASA/GSFC.

Keating, T. et al. (1990, July), *Small Explorer (SMEX)–1 Solar, Anomalous, and Magnetospheric Particle Explorer (SAMPEX) Flight Dynamics Support System (FDSS) Requirements* (CSC/TR-90/6002), CSC.

Kronenwetter, J., and Phenneger, M. (1988). *Attitude Analysis of the Earth Radiation Budget Satellite (ERBS) Control Anomaly* (CSC/TM-88/6154). CSC.

Kronenwetter, J., Phenneger, M., and Weaver, W. (1988, May) *Attitude Analysis of the Earth Radiation Budget Satellite (ERBS) Yaw Turn Anomaly* (Paper No. 18). Greenbelt, MD: Flight Mechanics/Estimation Theory Symposium, NASA/GSFC.

Natanson, G. (1992, September). A deterministic method for estimating attitude from magnetometer data only (Paper No. IAF-92-0036). *Proceedings of the World Space Congress.* Washington, DC.

Natanson, G., Challa, M., Kotaru S., and Woodruff, C., (1993, August). *Attitude Dynamics Task: Attitude Determination Using Only Magnetometer Data for the Solar, Anomalous, and Magnetospheric Particle Explorer (SAMPEX).*

Natanson, G., Keat, J., and McLaughlin, S. (1991, March). *Sensor and Advanced Attitude Studies: Deterministic Attitude Computation Using Only Magnetometer Data* (CSC/TM-91/6017). NASA/GSFC, Flight Dynamics Division, prepared by CSC.

Natanson, G., McLaughlin, S., and Nicklas, R. (1990, May). *A Method of Determining Attitude From Magnetometer Data Only.* Greenbelt, MD: Flight Mechanics and Estimation Theory Symposium, NASA/GSFC.

Wertz, J., ed. (1984). *Spacecraft Attitude Determination and Control.* Dordrecht, Holland: D. Reidel Publishing Co., 424-425.

111132
N95-17547
354101
p.8

TDRS ORBIT DETERMINATION BY RADIO INTERFEROMETRY

Michael S. Pavloff
The MITRE Corporation
Bedford, MA 01730

ABSTRACT

In support of a NASA study on the application of radio interferometry to satellite orbit determination, MITRE developed a simulation tool for assessing interferometric tracking accuracy. The Orbit Determination Accuracy Estimator (ODAE) models the general batch maximum likelihood orbit determination algorithms of the Goddard Trajectory Determination System (GTDS) with the group and phase delay measurements from radio interferometry. ODAE models the statistical properties of tracking error sources, including inherent observable imprecision, atmospheric delays, clock offsets, station location uncertainty, and measurement biases, and through Monte Carlo simulation, ODAE calculates the statistical properties of errors in the predicted satellite state vector. This paper presents results from ODAE application to orbit determination of the Tracking and Data Relay Satellite (TDRS) by radio interferometry. Conclusions about optimal ground station locations for interferometric tracking of TDRS are presented, along with a discussion of operational advantages of radio interferometry.

INTRODUCTION

As part of its effort to assess cost and performance benefits of various emerging technologies, NASA sponsored a series of studies on the application of radio interferometry to satellite tracking. Though astronomers had applied radio interferometry to astrometry for decades prior, it was not until the late 1960s that interferometry was proposed for use in satellite orbit determination. In an experiment devised by Irwin Shapiro, Alan Whitney, and others, very long baseline interferometric (VLBI) measurements were made on the TACSAT I communications satellite in geosynchronous orbit (GEO), and the semi-major axis of the orbit was measured with accuracy on the order of several hundred meters [1]. Subsequent experiments performed in the 1980s by Jim Ray, Curt Knight, and others to determine the position of the Tracking and Data Relay Satellite (TDRS) yielded accuracy on the order of 75 meters [2]. Such orbit determination accuracy, which derives from the extremely high precision of the group delay and phase delay observables, makes radio interferometry an attractive option for satellite tracking.

Operational considerations are also a benefit of radio interferometry in satellite orbit determination, because the group and phase delay measurements are made completely passively. Whereas the existing Bilateral Ranging Transponder System (BRTS) is taxing on TDRS communications resources, radio interferometry can derive its measurements from any signal, including the signal intended for the TDRS user community. Therefore, an interferometric orbit determination system for TDRS would eliminate traffic for tracking on the TDRS transponder. Because an interferometric tracking system would be passive, it would place no design constraints on the space segment, and it would therefore provide backward compatibility with all generations of TDRS. These potential operational and accuracy benefits led NASA to investigate radio interferometry for future TDRS tracking applications.

NASA sponsored a series of studies to investigate whether an operational radio interferometry system could provide TDRS orbit determination services (1) at lower cost, (2) at greater accuracy, and (3) across considerably smaller baselines than BRTS. Contributors to these studies included Interferometrics, Inc., where a Small Business Innovative Research (SBIR) contract was executed to demonstrate hardware and software that would provide group delay measurements on TDRS with VLBI. CSC performed an assessment for the Goddard Space Flight Center (GSFC) on a variety of TDRS tracking alternatives, including VLBI and Connected Element Interferometry (CEI) systems. The Jet Propulsion Laboratory (JPL) sponsored a series of experiments to determine CEI accuracy from its Goldstone facility. For its part of the effort, MITRE assessed optimal site locations and programmatic considerations of an operational interferometric TDRS orbit determination system.

For accuracy assessment purposes, MITRE developed a Monte Carlo simulation tool, the Orbit Determination Accuracy Estimator (ODAE), that models error sources in orbit determination with VLBI and CEI systems. In ODAE, the user can specify a satellite orbit, any set of ground stations between which group or phase delay measurements are to be made, and the statistical properties of the errors in those measurements. Upon each iteration of the Monte Carlo simulation, the orbit of the satellite is determined based on measurements with errors added, and the errors in the resulting satellite ephemerides are recorded. Thus, the user may study the statistical properties of the error in the batch orbit determination process resulting from the use of group or phase delay measurements.

We applied ODAE to study the effects of varying satellite and measuring station geometries on orbit determination accuracy. This paper presents an assessment of optimal siting for TDRS tracking by radio interferometry. A discussion of the operational and programmatic considerations of an interferometric tracking system are also presented.

THE ODAE MODEL

ODAE, which was implemented in Mathematica to allow maximum flexibility, models the batch maximum likelihood orbit determination process applied in the Goddard Trajectory Determination System (GTDS) [3]. The user specifies a reference true satellite orbit, a set of observing stations (earth-based or space-based), the observation types, and the times at which measurements are to be made. Given a set of observations on the satellite (e.g., radar measurements, group or phase delay measurements, or pseudorange measurements), ODAE determines the set of parameters (e.g., state vector, clock offsets, or atmospheric parameters) that best fit the observations. Upon each iteration of its Monte Carlo simulation, ODAE injects errors of user-specified statistical properties into various parts of the orbit determination process. ODAE computes the error of the measured parameters at each iteration, and at the end of the simulation, ODAE computes the statistical characteristics of the error.

Error sources that can be modeled by ODAE include inherent measurement imprecision, station location uncertainty, atmospheric delays, and clock offsets. The user must specify the statistical properties of the error sources. Trajectory propagation schemes available in ODAE for dynamic orbit determination range from the two-body approximation to numerical integration of the fully disturbed equations of motion. A detailed mathematical specification of the coordinate frame, force models, and numerical integration techniques used in ODAE are given in Reference 4. The only significant deviation from the GTDS approach to orbit determination is the use of Bulirsch-Stoer rational function extrapolation for numerical integration [5, 6]. For the numerical integration of the equations of satellite motion, the Bulirsch-Stoer technique has been shown to provide the same

precision as more traditional techniques, such as predictor-corrector integration or Runge-Kutta integration, but at reduced computational cost [4, 7].

For short-term dynamic orbit determination accuracy studies, it is often sufficient to apply simplified trajectory propagation schemes for the sake of reducing computation time. Absolute trajectory propagation accuracy is not of concern for the assessment of the relative effects of changes in geometry or measurement errors. For the study on TDRS tracking by radio interferometry, we were concerned only with the effect of ground station geometry on initial orbit determination accuracy, and so dynamics came to play only over the time of signal propagation from the satellite to the tracking stations. Therefore, we applied the two-body approximation for trajectory propagation and state transition matrix computation.

Since its initial application to the problem of optimal ground station siting for interferometric tracking of TDRS, MITRE has applied ODAE to a variety of problems, including an assessment of Space Surveillance Network Improvement Program (SSNIP) tracking accuracy on various classes of orbits, and an assessment of the accuracy of GPS for satellite telemetry, tracking, and command (TT&C).

INTERFEROMETRY OVERVIEW

Consider an interferometric orbit determination scenario in which O is the origin of an earth-centered inertial (ECI) coordinate system, $\mathbf{r}$ is the position vector of a satellite with respect to O, $\mathbf{b}_1$ and $\mathbf{b}_2$ are the position vectors of two ground stations from which measurements are to be made, and $\mathbf{d}_1$ and $\mathbf{d}_2$ are the position vectors of the satellite with respect to those ground stations, as pictured in Figure 1. The position vectors $\mathbf{r}$, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{d}_1$, and $\mathbf{d}_2$ are all functions of time. The sum of a station position vector, $\mathbf{b}_k$, and the satellite position vector measured from that station, $\mathbf{d}_k$, is simply the satellite position vector $\mathbf{r}$; therefore, $\mathbf{d}_k = \mathbf{r} - \mathbf{b}_k$. If the propagation rate, c, of the signal through the atmosphere is known, then the transit time, T_k, of the signal from the satellite at point P to ground station number k at point B_k will be given by

$$T_k = \frac{1}{c}|\mathbf{d}_k| = \frac{1}{c}\sqrt{(\mathbf{r}-\mathbf{b}_k)\cdot(\mathbf{r}-\mathbf{b}_k)} \quad (1)$$

Note that in equation (1), the vectors $\mathbf{r}$ and $\mathbf{b}_k$ are measured at slightly different times. Now, the true group delay, τ, between stations i and j is the differential transit time of the signal between these two sites:

$$\tau = T_j - T_i \quad (2)$$

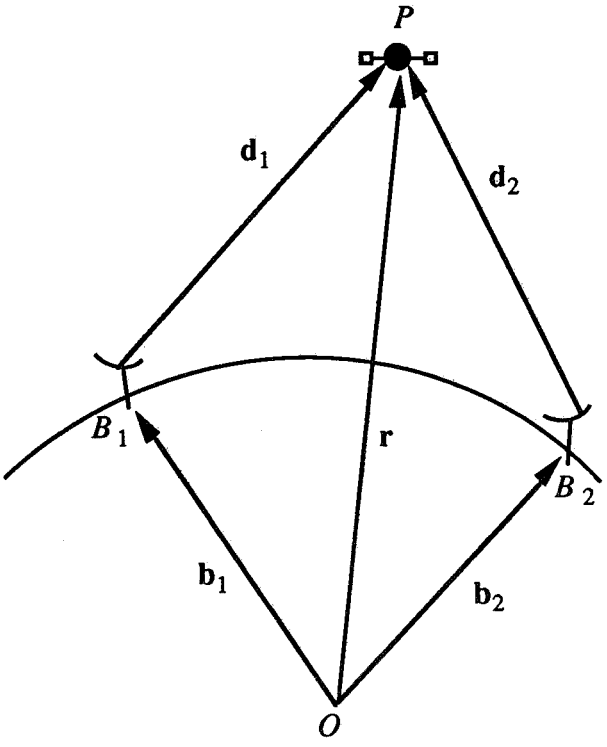


Figure 1. Illustration of the Interferometric Measurement Scenario

During the Monte Carlo simulation, ODAE computes measured group delay by adding measurement or atmospheric fluctuation errors to the true group delay as computed from equations (1) and (2). The solution of the orbit determination problem on each

iteration of the simulation, as described in Reference 7, follows the GTDS maximum likelihood estimation approach, one step of which is the computation of the Jacobian, or matrix of partial derivatives of equation (2) with respect to the state vector parameters at epoch.

For phase delay measurements, ODAE converts phase delay into equivalent group delay, as described in Reference 7. This computation can be accomplished so long as the cycle ambiguity can be determined from *a priori* information about the satellite's position vector. ODAE can model both the case where cycle ambiguity is unknown and the case where it is known. We assumed the latter in this study.

ODAE APPLICATION TO TDRS

In this section, we assess the level of orbit determination accuracy that can be attained for a geosynchronous satellite with radio interferometry, and we draw conclusions about optimal station-satellite geometry. The results are applied to recommend optimal ground station siting for orbit determination of TDRS by radio interferometry.

Radio interferometry with baselines the size of BRTS's, which are intercontinental, would translate the high level of observable group delay accuracy into greatly improved TDRS tracking accuracy. However, it was NASA's desire instead to accept only a modest improvement in accuracy while reducing system cost and ameliorating other operational considerations by greatly shortening the baselines. This led naturally to the study of a CEI-based system, where baselines are very short. Because of the requirement for a CEI system to have interferometer sites connected by fiberoptic cable in a temperature-controlled environment, the cost of lengthening baselines is very high. We constrained our baselines to 20 km maximum length for the purposes of this study.

We used ODAE to assess position determination accuracy on a GEO satellite for a sample interferometer siting scenario, and we determined the effects of varying the relative satellite to ground station geometry. Because the effect only of relative geometry was to be studied initially, it was not necessary to select true TDRS ephemerides or true potential ground station locations. The reference orbit chosen was geosynchronous with a 4° inclination and a subsatellite longitude of 18°W. To provide three independent baselines across which phase delay could be measured, we constrained four CEI sites to lie on the vertices of a square with a 20 km baseline, as shown in Figure 2. The site latitudes, longitudes, and altitudes for this reference scenario are given in Table 1. ODAE modeled simultaneous phase delay measurements across the baselines from station 2 to station 1, station 3 to station 1, and station 4 to station 1 (denoted 2-1, 3-1, and 4-1, respectively). These baselines are illustrated in bold in Figure 2.

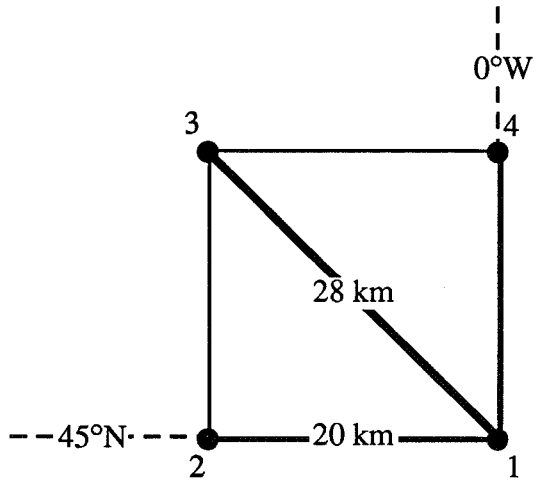


Figure 2. CEI Station Locations

An extension of Alan Whitney's work [8] shows that the theoretically achievable precision of the phase delay observable, σ_ϕ, is given by

$$\sigma_\phi = \frac{1}{2\pi(SNR)\nu} \qquad (3)$$

where ν is the center frequency, in Hz, sampled by the interferometer, and *SNR* is the signal-to-noise ratio. Since the TDRS downlink to White Sands is centered at 14 GHz and $SNR \approx 50$, the theoretically achievable precision of the phase delay observable is 0.23 picosec. While no TDRS tracking experiments were performed with JPL's CEI equipment at Goldstone, observations were made on natural radio sources at 8.4 GHz to assess the precision of the phase delay observable [9, 10]. JPL demonstrated the standard deviation in the phase delay observable to be approximately 1 picosec, which is 70% larger than the theoretically achievable value given by equation (3). Extrapolating this result to the theoretically achievable phase delay precision for TDRS, we estimated the practically achievable precision to be $0.23 \times 1.7 \approx 0.4$ picosec. We took this measurement error to be independently normally distributed across each baseline.

Table 1. CEI Station Locations for Reference Scenario

Station Number	Geodetic Lat. (°N)	Longitude (°E)	Altitude (km)
1	45.00000	0.0000	0.1
2	45.00000	-0.2545	0.1
3	45.17997	0.0000	0.1
4	45.17997	-0.2545	0.1

For the initial study, it was assumed that there were no equipment biases, that there were no atmospheric delay errors, that all station were connected by fiberoptic cable to one clock and frequency standard, that there were no local oscillator offsets between the four stations, and that station positions were known with perfect accuracy. Thus, the pure effect of measurement geometry and observable precision on orbit determination could be assessed.

ODAE Monte Carlo simulation of the orbit determination scenario described above with 200 iterations showed a 1σ root-mean-squared (RMS) position vector accuracy of 3.2 km. We also assessed the accuracy that can be attained with the use of other combinations of baselines. It is practical to have one site in common for all three measurements so that the common site can act as the correlation center at which the phase delay observables are generated. For the particular satellite and ground station locations in this scenario, selection of three measurements where one station is common to each pair (i.e., 2-1, 3-1, 4-1; or 1-2, 3-2, 4-2; or 1-3, 2-3, 4-3; or 1-4, 2-4, 3-4) results in a 1σ RMS position vector accuracy of 3.2 km. Thus, there is no geometrically-preferred common site for the measurements.

The orbit determination scenario described above was the starting point for the assessment of the effects of varying interferometric measurement geometry on orbit determination accuracy. Since only relative geometry matters, and since it would have been more cumbersome to vary the positions of four ground stations, we instead varied the satellite's initial position vector.

First, we studied the effect of relative interferometer baseline size on orbit determination accuracy. Satellite range from station 1 was varied while keeping the elevation angle and azimuth angle from that station constant. Because the baseline sizes are small relative to the range to GEO, the range, elevation angle, and azimuth angle from each of the other three stations are close to those of the first. For the sample orbit determination scenario described above, range from each site to the satellite is approximately 37,850 km, the elevation angle is approximately 39°, and the azimuth angle is approximately 155°. As shown in Figure 3, the smaller the range to the satellite for a constant baseline length (or, equivalently, the longer the baselines across which phase delay is measured relative to the range to the satellite), the greater the position vector accuracy.

Next, we assessed the effect of satellite azimuth angle on orbit determination accuracy. The azimuth angle of the satellite at station 1 in the original scenario was varied while keeping the range and elevation

angle from that station constant. The results indicate that for a configuration of four interferometric ground stations at the vertices of a square, position error is maximized when the satellite's azimuth angle is an integer multiple of 90°, and position error is minimized when the satellite's azimuth angle is an odd integer multiple of 45°.

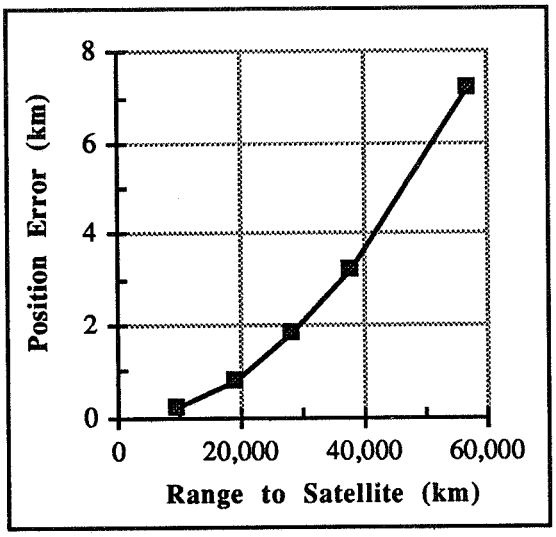


Figure 3. Position Error vs. Range to Satellite

Finally, we assessed the effect of satellite elevation angle on orbit determination accuracy in this scenario. The elevation angle of the satellite at station 1 was varied while keeping the range and azimuth angle from that station constant. As can be seen in Figure 4, for this particular orbit determination scenario, position error increases monotonically with elevation angle. Thus, based on the criterion of minimizing ephemeris error due only to error in the phase delay measurement, optimal viewing geometry is at the lowest possible elevation angle, and the scenario becomes degenerate when the satellite is at zenith.

A tradeoff is suggested by the geometrical result that greater orbit determination accuracy is attained at lower elevation angles. The tradeoff arises because statistical models of the variation in signal propagation rate through the troposphere show that, because a signal must pass through more of the troposphere as the elevation angle of the satellite decreases, errors in predicting signal propagation rate increase as elevation angle decreases [11]. Moreover, errors in predicting propagation rate due to tropospheric fluctuations tend to be the dominant error source in overall accuracy for CEI systems [12]. Thus, we sought to determine the optimal elevation angle for CEI measurements with consideration of both measurement error and tropospheric delay error.

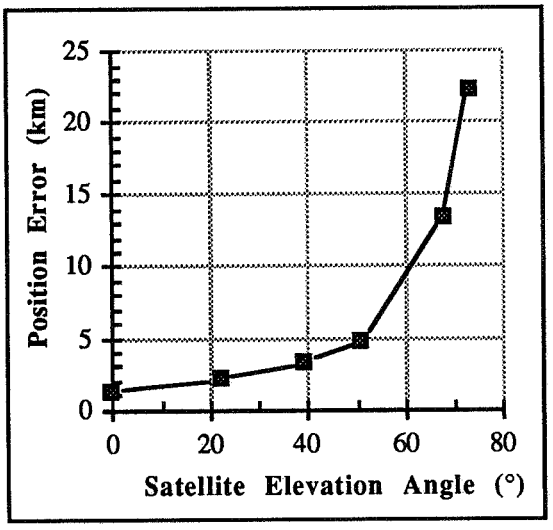


Figure 4. Position Error vs. Satellite Elevation Angle

We modeled tropospheric fluctuations between each interferometer site and the satellite as being independent and normally distributed. The assumption of independence is based on the fact that water vapor cells can be of several kilometers in diameter, and so tropospheric delay errors from each site can in fact be independent. From Reference 11, we computed the elevation angle dependence of the standard deviation in tropospheric delay error for 100 second measurement arcs of phase delay. The results are shown in Table 2.

Table 2. Tropospheric delay error as a function of elevation angle

Elevation Angle (°)	Tropospheric Delay Error (picosec)
10	7.5
20	5.7
30	4.6
40	3.9
50	3.3
60	3.0

For varying satellite elevation angles, we used ODAE to model error due to tropospheric fluctuations as well as inherent phase delay imprecision. The resulting 1σ position errors are shown in Figure 5. As can be seen, the optimal satellite elevation angle is approximately 30°. In the conclusions section of this paper, we show how these results can be applied to optimally siting a CEI system for TDRS orbit determination.

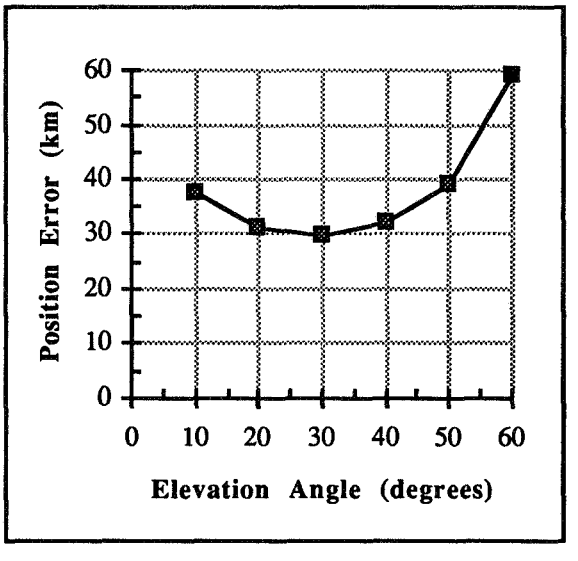


Figure 5. Position Error vs. Elevation Angle with Tropospheric Effects Included

CONCLUSIONS

We have derived conclusions about optimal geometry for orbit determination of a GEO satellite by radio interferometry. These results can be applied to the problem of optimally siting a CEI system to track TDRS. For a particular TDRS satellite, and for a configuration of four interferometer sites located at the vertices of a square, a geographical position should be chosen so that the satellite's elevation angle is as close to 30° as possible, and the square should be oriented so that the satellite's azimuth angle is an odd integer multiple of 45°. For TDRS-W at 171°W, the maximum elevation angle visible within the -20 dB contour of the White Sands downlink is in southern California at approximately 20° elevation. For TDRS-E at 41°W, an elevation angle near 30° can be attained within the -20 dB contour of the White Sands downlink by siting a CEI system in eastern Louisiana or western Mississippi.

DISCUSSION

Having determined optimal siting for a CEI TDRS tracking system, we return to a brief discussion of operational considerations. As stated previously, benefits include freedom from requirements placed on the space segment, the potential for excellent orbit determination accuracy, and the ability to locate the system entirely within the United States. It is expected that these benefits would ameliorate cost and operational constraints. Estimates have placed required staffing levels for an interferometric TDRS tracking system in the range from 10 to 20 full-time equivalent staff [13]. With respect to initial costs, Interferometrics demonstrated prototype hardware and correlation software for less than one million dollars [14]. Expected development and production costs for an operational system are expected to be an order of magnitude larger [13]. Finally, we note that interferometry offers low technological risk because it has been successfully applied in a number of related fields for several decades.

LIST OF REFERENCES

1. Preston, R. A., et al, 27 October 1972, "Interferometric Observations of an Artificial Satellite," *Science*, Vol. 178, pp. 407-409.

2. Ray, J., et al., October-December 1988, "VLBI Tracking of the TDRS," *The Journal of the Astronautical Sciences*, Vol. 36, No. 4, pp. 347-364.

3. Long, A. C., et al (ed.), July 1989, *Goddard Trajectory Determination System (GTDS) Mathematical Theory, Revision 1*, FDD/552-89/001, Greenbelt, MD: Goddard Space Flight Center.

4. Pavloff, M. S., September 1993, *A Monte Carlo Tool for Simulation of Satellite Orbit Determination by Radio Interferometry*, Cambridge, MA: The Massachusetts Institute of Technology, S.M. Thesis, Department of Aeronautics and Astronautics.

5. Bulirsch, R., and J. Stoer, 1966, "Numerical Treatment of Ordinary Differential Equations by Extrapolation Methods," *Numerische Mathematik*, Vol. 8, pp. 1–13.

6. Stoer, J., and R. Bulirsch, 1980, *Introduction to Numerical Analysis*, New York: Springer-Verlag Inc.

7. Pavloff, M. S., September 1992, *Analysis of Orbit Prediction Algorithms for the Universal Modem System*, WP-92B0000263V1, Bedford, MA: The MITRE Corporation.

8. Whitney, A. R., January 1974, *Precision Geodesy and Astrometry Via Very Long Baseline Interferometry*, Cambridge, MA: The Massachusetts Institute of Technology, Ph.D. Dissertation, Department of Electrical Engineering.

9. Edwards, C. D., April 1989, "Angular Navigation on Short Baselines Using Phase Delay Interferometry," *IEEE Transactions on Instrumentation and Measurement*, Vol. 38, No. 2.

10. Edwards, C. D., August 1990, "Development of Realtime Connected Element Interferometry at the Goldstone Deep Space Communications Complex," AIAA 90-2903, *AIAA/AAS Astrodynamics Conference.*

11. Treuhaft, R. N., and G. E. Lanyi, March-April 1987, "The Effect of Wet Troposphere on Radio Interferometric Measurements," *Radio Science*, Vol. 22, No. 2, pp. 251-265.

12. Edwards, C. D., January-March 1989, "The Effect of Spatial and Temporal Wet-Troposphere Fluctuations on Connected Element Interferometry," TDA Progress Report 42-97.

13. Potash, R., et al., September 1989, *Advanced Tracking Systems Design and Analysis*, CSC/TM-88/6060, Greenbelt, MD: Computer Sciences Corporation.

14. January 1991, *TDRS Interferometric Satellite Tracking System Field Station Hardware Manual and Guide to Operations*, Vienna, VA: Interferometrics Inc.

Operations

6. Small Explorers — Page 807

* *Presented in Poster Session*

111133

N95-17548

354104

COST EFFICIENT OPERATIONS FOR DISCOVERY CLASS MISSIONS

G. E. Cameron*, J. A. Landshof* and G. W. Whitworth*

The Johns Hopkins University
Applied Physics Laboratory
Laurel, Maryland 20723-6099

P 8

ABSTRACT

The Near Earth Asteroid Rendezvous (NEAR) program at The Johns Hopkins University Applied Physics Laboratory is scheduled to launch the first spacecraft in NASA's Discovery program. The Discovery program is to promote low cost spacecraft design, development, and mission operations for planetary space missions. In this paper, the authors describe the NEAR mission and discuss the design and development of the NEAR Mission Operations System and the NEAR Ground System with an emphasis on those aspects of the design that are conducive to low-cost operations.

INTRODUCTION

NEAR will launch in February 1996 and rendezvous with the asteroid Eros in January 1999. The spacecraft is to orbit Eros for up to a year, mapping the asteroid and collecting data on its gravitational and magnetic fields as well as its elemental composition. Significant challenges are anticipated in NEAR mission operations. NEAR will be the first spacecraft to conduct orbital operations around a small, irregularly shaped planetary body. Stringent orbital plane restrictions are required to simultaneously maintain instrument fields of view of the asteroid, communications antenna coverage of the Earth, and illumination on the solar panels. During certain portions of the year of asteroid operations, orbital maneuvers may be required every three days to maintain the orbital plane. Given the irregular shape and size of the asteroid, simple nadir pointing mapping strategies will not be sufficient for conducting operations at Eros; a flexible planning strategy must be implemented to coordinate scientific priorities given limited observation opportunities. These scientific observations must be combined with routine subsystem maintenance, orbital maintenance, and navigation requirements. A sophisticated sequence planning system with quick reaction capability is required (priorities and orbital dynamics can be expected to change on a continuous basis, requiring constant adaptation of operations to mission science needs).

These considerations generally increase the cost of mission operations in an era when Mission Operations and Data Analysis (MO & DA) costs are being scrutinized as never before. If NEAR and future Discovery class missions are to succeed, they must set new standards for cost efficiency. The goal of this paper is to show how mission operations costs can be controlled by the application of advanced technologies and operations concepts.

Organization of Paper

Following the Abstract and Introduction, this paper begins with a discussion of low cost mission operations. This is followed by a description of the NEAR Mission Operations System (MOS) which highlights those elements of the system design that contribute to low cost mission operations. Following the MOS description is a section detailing the design of the NEAR Ground System (NGS), again, with an emphasis on the low cost operations aspects of the design. Finally, we provide a summary of our recommendations for implementing low cost mission operations on Discovery class missions.

LOW COST MISSION OPERATIONS

The MOS is often the last element of the program to be developed; as such, the MOS frequently must make up for gaps and problems that have developed in the mission, spacecraft, and instrument designs. The MOS is generally custom developed for each mission, which is decidedly non-optimal from a cost-effectiveness viewpoint.

* Member of the Senior Professional Staff

PRECEDING PAGE BLANK NOT FILMED

Mission Operations costs can be divided into two major categories: development costs (mostly pre-launch) and operations costs (mostly post-launch). In the following discussion, potential cost saving measures are introduced in each category.

System Development

System development costs are primarily pre-launch and are generally incurred late in the pre-launch program. If a program gets into budget problems late in the spacecraft development phase (this is not uncommon), mission operations development costs frequently attract the attention of the budgetary ax-wielder. Saving money in development costs at the expense of repetitive costs in the post-launch mission operations phase is not cost efficient over the mission life cycle, yet this trade is frequently made. In the following, several approaches to saving costs in MOS development are discussed which do not compromise either mission capability or total life cycle cost.

Existing Infrastructure

Always take advantage of existing infrastructure where cost efficient. If an existing voice communications system or ground station network will work for your mission, why re-invent the wheel? It should be noted that existing infrastructure is not always cost efficient. Maintenance or personnel costs associated with outdated systems can negate their advantage. Each element must be individually evaluated on the basis of cost-efficiency.

Commercial-Off-The-Shelf Systems

Examine Commercial Off-The-Shelf (COTS) hardware and software systems for applicability to your program, again, on a cost efficiency basis. COTS systems have shown a tremendous growth in capability in recent years; low-cost programs can get a lot of bang for the buck compared to the development costs of custom systems. There are two major shortcomings of COTS systems. First, "COTS" elements for space mission applications are not the shrink-wrapped products we have come to expect in the truly commercial (i.e., PC) marketplace; they lack the smooth polish of a mass market product (e.g., documentation, on-line technical support) and must frequently be customized for each application. Make certain that the costs of these modifications are considered in the total cost of a COTS system. Second, many functions that are necessary to operate a complex space mission are not found in the COTS offerings. Straight-forward Telemetry, Tracking, and Control (TT&C) operations for a commercial satellite (such as a communications satellite) are significantly different from operations for a planetary exploration mission with complex planning tasks and command sequence development. COTS products tend to be stronger in meeting the needs of commercial users than scientific mission planners.

Concurrent Engineering

Use modern concurrent engineering development techniques. Traditional approaches to system development (requirement definition, specification development, preliminary and detailed design, fabrication, and test) are slow, cumbersome, and costly. Modern methods of system development such as concurrent engineering and rapid prototyping can be faster and cheaper. There are risks in this approach, however, the benefits generally outweigh these risks. For Discovery programs, higher risks must be tolerated to achieve the avowed goals of faster, better, and cheaper.

Design for Operability

Design the spacecraft and Mission Operations System for operability. Too often, flexibility and operability are relegated to the ground system and mission operations team to save development costs in the spacecraft. While this is an understandable approach (complexity vs. reliability tradeoffs in the spacecraft favor simplicity), this may not be the optimal approach. In some cases, relatively minor changes in spacecraft or instrument design can significantly save in operations costs (sometimes, over and over again). For example, thermal and power robustness may eliminate the need for complex analysis of every maneuver sequence, saving time and money in the development of sequence uploads. A mission level system engineer should have the authority and responsibility to perform such tradeoffs at a high level.

System Commonality

Build systems that achieve simplicity through the use of common architectures. Cost savings due to system commonality may not be apparent at the mission operations level, but are observable at the program level. Many Integration and Test (I&T) functions are duplicated in the Mission Operations System and vice versa. Why should these capabilities be developed twice? Using a common system design for Mission Operations (MO) and I&T saves money not only in design and development of the ground system, but in sparing, training of personnel, and staffing during test, launch, and mission ops.

Operations

The division of operations costs between pre- and post-launch is mission dependent. Pre-launch development of operations teams and processes, personnel training, and system testing can be significant cost items. If the mission is short, or if it can be staffed at a very low level, pre-launch costs can be a significant portion of overall operations costs to the program. If the mission is long, complex, or both, post-launch costs tend to be the driver of overall costs. In the sections that follow, we shall show how intelligent application of pre-launch funding can significantly reduce post-launch costs.

Low Staffing Levels

Minimize the number of personnel needed to operate the spacecraft during post-launch operations. The major post-launch cost item for most missions is personnel. In most programs, the key to lowering operations costs is to reduce the number of people required to operate the spacecraft.

Personnel reductions can be achieved merely by paying attention to the type and capabilities of personnel hired and the changes in skills needed during different phases of the mission. As teams become smaller, the competence and breadth of individual members becomes more important. Small teams can not afford to have members with specialized or limited skills; every team member must contribute significantly to the overall productivity of the team for operations to be cost efficient.

It is important to note that the skills required during design and development of the MOS are not the same as those required during post-launch operations. Personnel should be added as their skills are required and removed when their skills are no longer applicable to the needs of the program. This may conflict with the policies of some organizations, but is essential to controlling operations costs. Large institutions frequently utilize matrix management techniques that allow the program to draw from a broad mix of skilled personnel, paying only for the time charged to the program. Matrix techniques can be advantageous in the implementation of these practices.

Spacecraft Autonomy

Build spacecraft systems that require minimal operations support. Perhaps the most obvious way to reduce operations cost is to build a spacecraft that does not require operations! The more autonomy built into a spacecraft, the less the MOS needs to do. The prevailing view is frequently the inverse -- the more the ground does, the less the spacecraft needs to do. Mission system engineering of the spacecraft and MOS offers the capability to partition requirements between the ground and flight systems. If the optimization goal is to minimize overall program costs, operations costs will generally be lower. Even if cost is not an optimization parameter, the consideration of mission operations issues in the design of the spacecraft will generally result in cost savings (due to operability enhancements). Frequently, the spacecraft design team has options that have little impact on the spacecraft but significant advantage to mission operations.

Spacecraft autonomy features which simplify operations include: telemetry monitoring and alarming; processor memory management; anomaly detection, correction, and/or reporting; automated data handling; and multi-level autonomous safe modes. Each of these features are discussed below.

Autonomous telemetry monitoring and alarming reduces the work load on ground personnel, especially if the MOS is designed to communicate spacecraft generated alarms to operations personnel immediately. This

reduction in the need for ground system monitoring reduces the number of personnel and the frequency of contacts required. During missions with long cruise phases and infrequent contacts, onboard alarming, coupled with storing alarm status in memory, can enable operations personnel to instantaneously assess the state of spacecraft health since the last contact. This reduces the contact time required, the operations load, and thus, the total cost to the program.

Automation of memory management allows the MOS to use lower fidelity models of onboard processors, thereby reducing development costs. Additionally, fewer commands are required for processor memory management, reducing the costs of testing those commands as well as simplifying operations.

Autonomous anomaly detection, correction, and reporting is similar to onboard telemetry monitoring and alarming with respect to operations. The potential reduction in operations workload and the increase in intervals between contacts results in a reduction in operations personnel.

Autonomous data handling, in which the spacecraft processes, stores, and retrieves data by instrument or subsystem without detailed operator intervention, allows the operations team to use contact time more efficiently and send fewer commands, reducing the workload and cost of operations.

Multi-level safe modes allow the spacecraft to assume intermediate modes of operation between fully operational and "cocoon" mode (minimal activity, awaiting ground command). For example, a failure in the data handling system may cause the spacecraft to shut down the data handling system, point the antenna at Earth (assuming guidance, navigation and control functions are unaffected), and await instructions. Allowing the good subsystems to remain operational means that the anomaly will be addressed more quickly than would otherwise be the case. This allows for longer intervals between contacts, which reduces operations loads and costs. This also reduces the time spent and the assets utilized in recovering from a failure.

Ground System Automation

Build ground systems that minimize personnel requirements. The use of automation in the ground system can significantly reduce requirements on operations personnel. Most apparent is the application of automated telemetry display and command generation capabilities. The use of high level command languages reduces operations personnel requirements, as do integrated databases, graphical user interfaces, and automatic report generation and transmission capabilities.

The next logical step in ground system automation is ground systems that autonomously receive, process, interpret, and respond to spacecraft telemetry. While totally automated operations are not yet feasible for scientific missions, many functions can be automated. Automated monitoring of telemetry can not only alert an operator to an out-of-bounds condition, it can spawn a process to advise the operator what to do (i.e., retrieve a contingency plan from a database), or even take action itself (depending on the nature and severity of the anomaly). Spacecraft data trending and analysis can be highly automated, generating formatted reports and delivering them electronically to the correct parties at the appropriate times (e.g., at shift changes or on Monday mornings). Clearly, all of these capabilities can be used to reduce the personnel otherwise needed to perform these tasks.

Advanced Technology

Utilize advanced technologies, where applicable, to enhance productivity in operations. The application of advanced technology throughout Discovery class missions has been mandated by NASA (the NEAR mission design predates this mandate, and NEAR is specifically exempted from this requirement). Advanced technology can reduce operations costs by enhancing productivity, i.e., allowing fewer people to accomplish more work with fewer resources expended. Two ways in which advanced technology can be used to enhance productivity are: 1) advanced technology can enable the use of higher level interfaces to gain insight into data and processes, and; 2) advanced technology can be used to assist

in making decisions. The application of advanced graphical techniques to gain insight into complex data sets is called *visualization*; and the use of software to assist in decision making processes falls in the category of *expert systems*.

Visualization

Everyone has seen global maps with projected spacecraft ground traces, coverage circles of ground receiving sites, and perhaps time ticks indicating when a spacecraft will or did pass over a particular spot -- these types of displays were a staple of the highly publicized manned space missions of the 1960's. This type of display is a prime example of the use of visualization to provide insight into a complex data set -- in this case, the orbital ephemeris of the spacecraft, the locations and views of each of the ground network's tracking stations, and the time the spacecraft will be available for contact at each of the ground stations.

Humans excel at the assimilation of visual information. The recent trend in returning to traditional watches and clocks from the digital variety is evidence of this phenomenon. People easily interpret the time of day from the angles of clock hands, whereas a digital clock requires assimilation and interpretation to understand. Computer graphics are a powerful tool for taking advantage of this characteristic of the human brain to reduce operations costs. The trend in operations systems is away from alphanumeric screens with numbers and cryptic mnemonics towards graphical displays, including analog dials, graphs, and trees of color coded boxes representing spacecraft systems and subsystems, etc. Aircraft cockpits with modern CRT and flat-panel displays utilize representations of analog dials and "tape" gauges for the same reasons operations systems do; these displays rapidly and intuitively present more information to the user more quickly than alphanumeric displays, thus allowing fewer people to monitor a complex system more efficiently and completely -- and with fewer errors. Fewer people mean lower costs, and fewer errors mean greater spacecraft safety.

Expert Systems

More advanced than visualization (already in use in operations centers, albeit sparingly) is the use of expert systems to assist in decision making processes. Rule-based expert systems are currently in use in some operations systems to assist in telemetry monitoring and display functions. Rule-based systems may also be used in the near future to help diagnose spacecraft anomalies, again, based on interpreting spacecraft telemetry. In artificial intelligence circles, however, rule-based systems have fallen out of favor because of their inherent lack of robustness; these systems can only apply pre-programmed rules to a known data set, and can be very difficult to adapt rapidly to changing conditions. For complex systems, the rule sets can get very large and difficult to manage. Finally, rule-based systems require all rules to be programmed before the system is very useful.

Model-based systems are being investigated for spacecraft operations because they address these problems. Model-based reasoning (MBR) methods use models of systems and subsystems to make estimates of systems states. MBR allows incremental growth in capability as models are added, refined, or updated, and can provide answers that are both qualitative and quantitative. MBR can be used to diagnose problems based on spacecraft telemetry, but the models can also be used to support analysis in the sequence generation process.

Model-Based Reasoning appears likely to reduce MOS costs in two ways. First, it may allow the development of a single set of spacecraft models to perform planning, analysis, and assessment functions, thereby reducing system development costs over traditional MOS designs. Second, it may allow fewer analysts to generate very complex spacecraft sequences with greater confidence, thereby reducing personnel requirements while enhancing mission capability. MBR may be a suitable alternative to the building of costly hardware-based spacecraft simulators traditionally used for command sequence vetting.

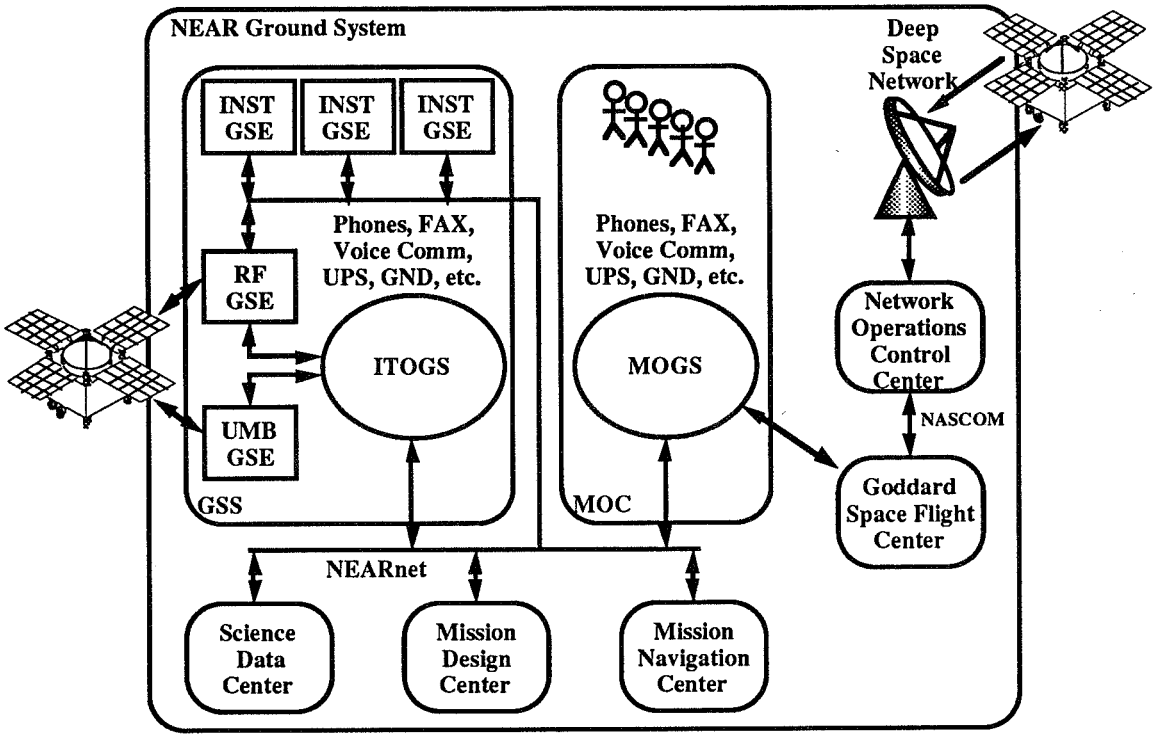


Figure 1. NEAR Ground System

MISSION OPERATIONS SYSTEM

Ground System

Figure 1 is a high level diagram of the NEAR Ground System (NGS). There are six major ground facilities: the Mission Operations Center (MOC); the Ground Support System (GSS); the Mission Design Center (MDC); the Science Data Center (SDC); the Mission Navigation Center; and the Deep Space Network (DSN), which is linked via NASA Communications (NASCOM) circuits at Goddard Space Flight Center (GSFC).

Mission operations will be conducted from APL. Therefore, the MOC and MDC are located at APL. The principal equipment in the MOC is a suite of interface equipment and high-end workstations, including software, known as the Mission Operations Ground Segment (MOGS).

The GSS includes a parallel construction called the Integration and Test Operations Ground Segment (ITOGS) as well as the Ground Support Equipment (GSE). The GSS is used to perform integration and test of the spacecraft at APL, environmental testing at GSFC, and prelaunch testing at the launch site. The ITOGS and MOGS are identical; by virtue of the interconnecting data network called NEARnet, each has controlled access to the spacecraft.

Science data received by the MOC is processed and passed on to the SDC, which further processes the data for dissemination to the science community. The Mission Navigation Center, located at the Jet Propulsion Laboratory (JPL), provides navigation data and products to the MOC, the SDC, and the MDC.

Existing Infrastructure

The NEAR Ground System maximizes the use of existing infrastructure, including the DSN and NASCOM. The DSN is used for all TT&C for NEAR. Operated by JPL, the DSN is a ground network primarily used for interplanetary missions, with ground station complexes in Barstow, California, Madrid, Spain, and Canberra, Australia.

Access to the DSN is provided via NASCOM. NASCOM will be used for virtually all NEAR communications. This includes extensions of the NEARnet to the

ITOGS as it moves with the spacecraft to GSFC and to the Kennedy Space Center (KSC) and Cape Canaveral Air Force Station (CCAFS). The cost effectiveness of using NASCOM for NEAR is multiplied because the arrangements for its use are provided by the DSN as a service.

A third major use of existing infrastructure is internal to APL. As discussed, the workstations, GSE, and peripherals of the MOGS and ITOGS are tied together as one large system via the NEARnet. Within APL, NEARnet uses an existing ethernet communications system called the APL Network Information System (APLNIS). APLNIS is ubiquitous throughout APL and supports multiple interface configurations. APLNIS supports TCP/IP protocols and has an existing connection to Internet, which provides off-campus access to the SDC. Connections of the ITOGS and MOGS to the APLNIS will utilize a router to provide protection against unauthorized access to spacecraft control and telemetry.

It should be noted that the NEAR spacecraft conforms to the standards of the Consultative Committee on Space Data Systems (CCSDS), and will be the first spacecraft to use CCSDS for uplinking. In using this system, NEAR is effectively making use of another set of existing infrastructure that results in reduced costs within the NGS.

Commercial-Off-The-Shelf Systems

An important aspect of the NGS implementation approach is the use of COTS mission operations systems. Although this industry is still young, a number of available systems offer capabilities in one or more aspects of spacecraft telemetry processing, performance assessment, and command and control. The core of the NGS is COTS. This core provides telemetry monitoring, alarming, and archiving, as well as spacecraft command and GSE control. Two systems are being procured for the MOGS and ITOGS; when augmented with additional workstations and custom software developed by APL, they will constitute the ITOGS and MOGS.

The core system includes a VME-based front-end, a workstation, and peripherals. The front-end provides the telemetry and command interfaces to the spacecraft (or more correctly, the spacecraft GSEs and/or the DSN via NASCOM) as well as realtime decoding, error correction, and data handling required to provide data for display on operator workstations. Workstation processing includes calibration, engineering unit conversions, display, alarming, and command script generation. Workstations may analyze realtime or archived data, or a combination. A large number of workstations can be supported on the NEARnet, and as described previously, these can be located anywhere.

Like many other current COTS systems, the NEAR MOC and GSS use networking and distributed processing. In each area, the workstations, peripherals, and command and telemetry interfaces are merely logical groupings of equipment on the NEARnet, with equal access to all data whether it enters the system via the MOC or the GSS. Each workstation has equal access to the "front end" of either area. The look and feel of the system remains the same in all locations; the parallel nature of the networked system provides a mutual backup capability.

This networked architecture permits the system to take advantage of distributed processing. The NEAR MOS has no large central computer with the resultant interference and speed problems as different workstations access and run processes on the central facility. These workstations simultaneously and independently run different processes on the same or different realtime or archived data. This permits a single database (e.g., telemetry and command dictionaries) to be accessed from any workstation, preventing the problems of maintaining multiple dictionaries. Incremental growth in the ground system can be easily accommodated without disrupting existing (operating) components.

The NEARnet extends beyond the MOGS and ITOGS, providing controlled (authorized) access to selected data on the NEARnet by other workstations or PCs. One recipient of data is the Science Data Center (which also has workstations and peripherals connected to the NEARnet).

The SDC is given essentially raw science data at the CCSDS Transfer Frame and Packet level and provides various levels of processing to generate products for the science community, which accesses these products via the NEARnet. Off-campus science teams may obtain access via the Internet. Two other Centers have access to the NEARnet Science Data Center. These are the Mission Design Center and the Mission Navigation Center.

One additional aspect of the ITOGS and MOGS worth noting is the use of an open operating system. All of the commonly recognized advantages of this approach are realized for NEAR. For example, access to commercial software is maximized; in-house software can be developed on non-NEARnet workstations or PCs with minimum problems in transporting these to MOS workstations. Further, the NEARnet configuration is much more supportable and expandable over the life of the mission.

Common architecture for I&T and MO

It is important to note that the MOGS and ITOGS are identical in configuration, software, hardware, and command and telemetry capability. This is significant in at least two aspects. First is the reduced development and maintenance costs resulting from identical workstations, front-end equipment, and peripherals. Because a single system design and architecture is used, overall complexity and design effort is reduced, as is the number and cost of procured components. Additionally, spares and maintenance costs are minimized.

The second significant aspect of using identical systems for I&T and MO is that the spacecraft will be flown as it was tested. The look and feel of the two segments is the same to the user. Since both sets of front-end equipment are also identical, (each supporting the three modes of interface with the spacecraft: RF GSE, umbilical GSE, and via NASCOM and the DSN), and since either can be accessed from a workstation in either the MOGS or ITOGS, the only distinction between the two is established by access authorization. While I&T activities will be principally controlled from the ITOGS due to its proximity to the spacecraft and GSE, considerable capability exists, and will be utilized, to exercise the spacecraft from the MOC during the I&T phase. When this commonality of hardware and software is considered in light of the current plan to have a number of mission operations personnel involved in integration and test, the transition from I&T to MO should be as seamless as is achievable. This blending of traditionally separate and distinct functions significantly reduces the total cost and development time for the ground support elements of the NEAR mission while improving the quality and reliability of the overall product.

SUMMARY

This paper began with a discussion of low cost mission operations, including a number of specific recommendations for controlling costs. These are summarized below: 1) Always take advantage of existing infrastructure where cost efficient; 2) Use Commercial Off-The-Shelf hardware and software systems where applicable and cost effective; 3) Use modern concurrent engineering techniques; 4) Design the spacecraft and Mission Operations System for operability; 5) Build systems that achieve simplicity through the use of common architectures; 6) Minimize the number of personnel needed to operate the spacecraft during post-launch operations by building spacecraft and ground systems that minimize personnel requirements, and; 7) Utilize advanced technologies, where applicable, to enhance productivity in operations. While these simple statements may seem obvious, they are frequently forgotten or overlooked as heritage often dictates the design and implementation of the MOS.

The second part of the paper included a description of the NEAR MOS and ground system with an emphasis on those elements of the system design that contribute to low cost operations. In the case of NEAR, we were able to apply almost all of the practices discussed in this paper. It is our hope that NEAR Mission Operations will introduce a new way of doing business for Discovery, and that this will lead others to identify even better approaches to controlling costs in today's cost-constrained environment.

111134

N95-17549

354199

GROUND STATION SUPPORT FOR SMALL SCIENTIFIC SATELLITES

p-8

R. Holdaway, E. Dunford and P.H. McPherson

Rutherford Appleton Laboratory
Chilton, DIDCOT
Oxfordshire, OX11 OQX, UK

ABSTRACT

In order to keep the cost of a complete small scientific satellite programme low, it is necessary to minimise the cost of the Ground Station Operations and Support. This is required not only for the operations and support per se, but also in the development of Ground Station hardware and the mission associated software. Recent experiences at the Rutherford Appleton Laboratory (RAL) on two international projects, IRAS and AMPTE, have shown that the low cost objectives of operations using smaller national facilities can be achieved. This paper describes the facilities at RAL, and the methods by which low cost support are provided by considering the differing implications of hardware/software system modularity, reliability and small numbers of dedicated and highly skilled operations staff.

INTRODUCTION

Rutherford Appleton Laboratory (RAL) is part of the UK Engineering and Physical Sciences Research Council (EPSRC) - formally the Science and Engineering Research Council (SERC). RAL has a long history of Space Science and Technology going back to the early 1960's, and in more recent times RAL has had TT&C responsibilities for a number of space missions. In 1983, RAL operated the Infra-Red Astronomical Satellite (IRAS) on behalf of NASA, SERC and the Dutch Aerospace Agency NIVR. Operations with IRAS covered all aspects of ground System work, including Mission Planning, Command Generation, Satellite Control, Data Reception, Satellite Health Monitoring, and Detailed Science Analysis. The mission lasted for 10 months, and operations went flawlessly, with no passes being missed. In 1984, the Ground System was re-configured for operations on the Active Magnetospheric Particle Tracer Explorer (AMPTE) mission. AMPTE was a UK sub-satellite operating as part of a NASA, UK, West German mission. Unlike IRAS (which was in a sun-synchronous orbit), AMPTE was in a highly eccentric orbit, taking apogee out to 200,000 km, giving real-time operations of up to 14 continuous hours per day. In both those missions, hardware, software and operations were developed and run by a closely-knit group of experienced space engineers, all contributing to a cost-efficient operational programme, even though in the case of IRAS it was not classified as a 'small' mission.

The RAL Ground Station is currently being re-configured again for operations with Small Satellites. Data reception monitoring will begin shortly on the Space Technology Research Vehicle (STRV) program. STRV is a UK Ministry of Defence mini-satellite, operating at S-band frequency. Once the downlink end-to-end system has been checked out, RAL will finalise plans for complete end-to-end, low cost operations on another mini-satellite programme, called BADR-B. BADR-B (Urdu for full-moon) is a Pakistan mini-satellite programme managed by the Space and Upper Atmosphere Research Commission (SUPARCO) in Karachi. Due for launch in 1995, BADR-B will be placed in a near-polar orbit at an altitude of about 800 km. Prime operations will be run from Karachi and Lahore in Pakistan, and UK operations will be run from RAL, using an ultra-low cost approach as defined in the remainder of the paper.

THEMES FOR LOW COST OPERATIONS

The starting point in defining the requirements on the Ground Station is to consider what the User actually needs (as well as what he wants, which may not necessarily be the same!). Overall, a rough guide to the main requirements may be considered as:

> Lowest possible cost, but reliable operations (not missing passes or losing data), fast return of critical data, regular return of bulk data, rapid response for critical commanding and ease of access to data

In order to achieve the low cost goal, it is not, however, unreasonable to expect some compromises to be made. These may include:

> Acceptance of occasional (1 in 20?) lost passes, acceptance of some (5%?) lost data, and/or non-rapid return of non-urgent data

With these ground rules understood, we can look at some of the potential areas of cost reduction.

COST REDUCING SCENARIOS

The cost of mission operations represents a significant portion of the total programme costs, often 20 to 30%. Thus the ground segment configuration (ie. hardware, software) and the operational modes (ie. complexity) have a significant influence on total costs and must be given serious consideration in overall system design.

The ground segment fulfils several functions:

- mission planning, including command preparation and validation,
- tracking, telemetry and command (TT&C) interface with the satellite,
- status and health monitoring of the satellite,
- reception of mission data via satellite telemetry,
- initial pre-processing of the data prior to distribution from the operations part of the ground segment to the user for final processing and analysis,
- use of EGSE before and after launch.

The following are some general considerations concerning the ground segment configuration and operation.

System modularity

In exactly the same way that satellite costs can be significantly reduced by greater use of common modularised subsystems, ground system configurations can also be modularised. Instead of developing individual EGSE (Electrical Ground Support Equipment) and Ground Segment equipment for every instrument and/or satellite, there are now being developed standardised off-the-shelf equipment that can subsequently be customised to the individual needs, at much lower cost. Within the ground system itself, computing power is sufficient these days to combine the tasks of TT&C into a single low-cost workstation. Of even more potential benefit is the reuse of

previous mission software for many of the data analysis functions. As an example of this, the data analysis software for the JET-X instrument, which will fly in 1995 as part of the Spectrum-X mission, is almost entirely based on software developed for the ROSAT mission launched in 1990. This scenario alone has cut the software development cost for this mission by a factor of three.

National facilities

Probably the greatest potential for cost reduction of the ground system is by making greater use of national facilities. Agency facilities are clearly required for large (manned and unmanned) missions, but are often too cumbersome and inflexible for small missions. It has usually proven far more cost-effective to employ national facilities - ideally utilising just a single ground station. For instance, the two European AMPTE spacecraft were controlled from single stations in Germany and England, respectively. The UK station was developed at very low cost by updating the original IRAS control centre to the requirements of the AMPTE mission. Although new software and operational procedures were necessary, very little new hardware was required. As an example, the 12 m S-band tracking station and control centre at the Rutherford Appleton Laboratory can be used for TT&C on an "as required basis", the operations staff being redeployed to other tasks during non-active satellite periods, thus significantly cutting down the running operations costs even for satellites producing many hundreds of Mbits/day. Similarly, for low-cost satellites producing kbits rather than Mbits of data, it is now possible to receive data using rooftop antennae and to command/receive using desktop PCs.

Reliability versus cost

For larger missions, it has always been normal practice to maximise the reliability of the ground system despite the associated increase in cost. This is not unreasonable for man-rated missions, but is often an unnecessary expense for most other missions. There is a very sizeable potential reduction in cost to be obtained by accepting just a small reduction in system reliability. It is proposed here to agree "up-front" that a small percentage (perhaps 5%) of satellite passes can be lost through ground system outage. This may (though not necessarily) lead to some data loss, but even so a data loss of a few percent is not usually significant. By agreeing to this reduction in reliability, the level of hardware redundancy (and perhaps software complexity) required in the ground system can be significantly reduced, and hence the cost is lower. Likewise, if the number of passes required per day to support the mission operation can be reduced through a slightly less than optimal coverage programme, the cost of operations can also be reduced.

Data availability

There is no doubt that for all missions it is essential to be able to process some subsets of the data in Real-Time and/or Near Real-Time. However, the less data that has to be processed in this manner the simpler the immediate ground system complexity becomes. For the majority of small satellite missions, it should only be necessary to process instrument/bus health data as a matter of urgency, thus decoupling the task of satellite "operations" from that of off-line data processing.

Data transfer

There are basically two different methods of transferring data from the operations part of the ground system to the user or data processing centres. The first (and most expensive) is via one of the many space or terrestrial data links. This is the common route for most satellite data and gets the data to the end user very quickly. However, it is more often the case that although the end user likes to have this data "as quickly as possible" it is not often an absolute necessity. In this case an alternative route via mailed magnetic tapes/optical disks can be

just as satisfactory; possibly some (small) percentage of the data can still be transmitted via a low bandwidth (and lower cost) data link; it is important to try to avoid the exclusive dedicated use of these links as this too adds to the cost.

Data access

There are as many different philosophies regarding methods of data access as there are concerning designs of satellite. Generally however, the most cost efficient and practical method is the concept of a Centralised Data Handling Facility which is accessible by users over local data networks. This concentrates the pipeline data processing in one place, whilst allowing the individual users both to develop their own specialised software and to make full use of centrally developed software.

With these principles addressed, we now look at the Ground Station and Operations facility at RAL.

RAL GROUND STATION HARDWARE

The Science and Engineering Research Council's Rutherford Appleton Laboratory (RAL) operates a Ground Station and Control Centre on its site at Chilton, Oxfordshire, UK. (51.57°N, 1.31°W).

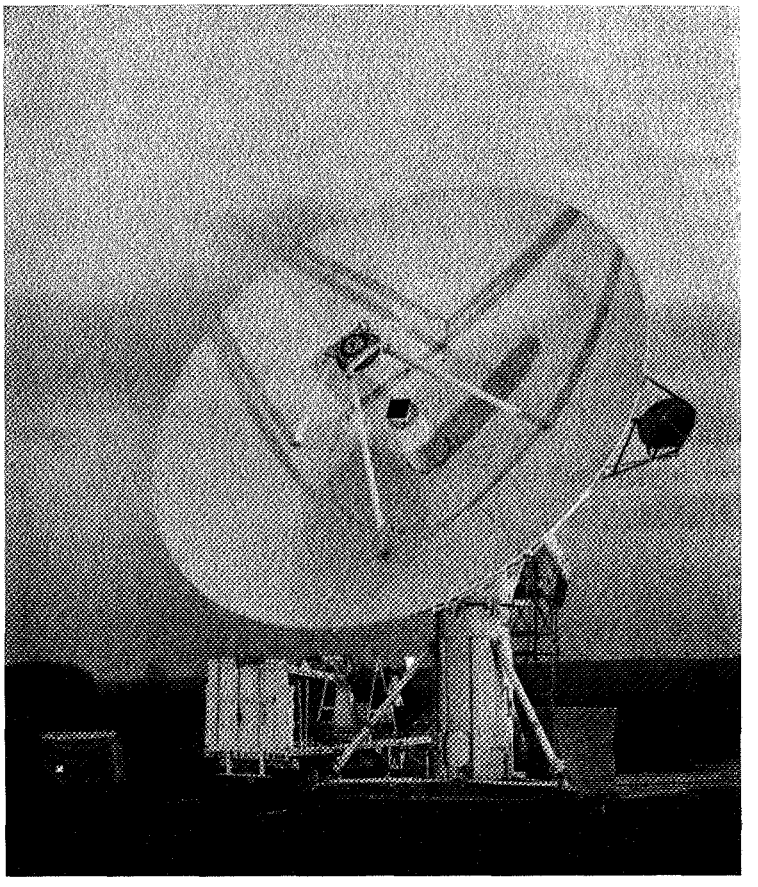


Fig. 1 RAL 12 m Antenna

The main antenna is a transportable 12 metre S-band cassegrain instrument (Fig. 1). Built in 1965 by the North American Aviation Company for the ATS project, it was re-commissioned in 1980 on the Chilton site as the prime antenna for the joint UK/US/Dutch mission IRAS (low-Earth polar orbit).

Antenna System

The main reflector of the antenna is a hyperboloid section, made with 20 petals constructed from 2 in thick aluminium honeycomb and faced with aluminium sheet. The reflector, the radio frequency feed, cassegrain sub-reflector and equipment cabinets are supported on elevation over azimuth bearings at the top of a cylindrical steel pedestal. Three tubular steel legs provide support for the pedestal and, with screw jacks, allow accurate levelling of the antenna structure. The whole antenna weighs approximately 32 tons. Attached to the edge of the main reflector is a 1.2 metre diameter paraboloid antenna which, because it has a wider beamwidth (ie. field of view) is used to locate satellites whose position is uncertain.

The radio frequency feed mounted at the vertex of the main reflector is a complicated waveguide structure. It is able to transmit and receive simultaneously at S-band frequency, in either right-hand or left-hand circular polarisation. In the receive mode, three output ports are available: one is the channel containing the received signal, the other two provide error signals (one each for azimuth and elevation axes) so that, with a servo loop, the antenna can lock on to an incoming transmission, allowing very accurate tracking of selected satellites. In addition to this autotrack mode, the antenna can be driven along a predicted path by computer.

The pointing error of the antenna is approximately 1 arc minute. The success of the antenna, as a machine for tracking moving sources, depends ultimately on the quality of the servo mechanism. The electric drive system incorporates two motors per axis and a redesigned set of servo amplifiers, aimed at maintaining the peak tracking error within 6 arc minutes, at mean wind speeds of up to 30 knots. Tests have shown that this figure is easily met and a typical peak tracking error is 2 arc minutes in a mean wind speed of 20 knots with gusts above 30 knots.

A summary of the technical details of the antenna are as follows:

Mechanical

Cassegrain configuration
12 m diameter paraboloid primary reflector, f/d ratio 0.325
1 m diameter hyperboloid secondary reflector
Eccentricity 1.413
Main reflector surface accuracy 0.89 mm rms
Mount: elevation over azimuth, Azimuth rotation ± 270 deg , Elevation rotation -5 deg to +95 deg

Drive

Electronically servo-controlled electric motors
Two motors in tandem in each axis
Static pointing accuracy ± 3.5 sec arc rms, Tracking accuracy ± 2 min arc rms
Velocity, azimuth and elevation 7 deg/sec max
Acceleration, azimuth and elevation, 4 deg/sec^2 max
Modes of operation: Standby, Manual or Program-Track

Data output: Position encoder 20 bits, Accuracy ± 1.23 sec arc

RF	Transmit	Receive
Antenna gain 45.8 dB		46.5 dB
Beamwidth (3 dB)	0.9 deg	0.8 deg
Nominal frequency	2075 MHz	2253 MHz (IRAS)

Transmitter power into antenna 10 watts
Max side lobe - 18 dB from main lobe
System noise temp 115 K at zenith
Feed: Four horn monopulse
Left or Right hand circular polarisation receive and transmit
Output: 3 channels - sum and two orthogonal error channels

Acquisition Aid

1.25 m diameter paraboloid reflector (fixed to rim of main antenna)
Receive only
Antenna gain 26 dB
Beamwidth 6 deg
Nominal frequency 2253 MHz
RH circular polarisation
Output: 3 channels - sum and two orthogonal error channels

Receive/Transmit System

The receivers and exciters were previously sited at the NASA STDN ground station at Madrid and were used on the Apollo programme. They are based on the NASA unified S-band system.

The system comprises:

(1) Two identical receivers with a common phase reference generator,

(2) Two identical transmitters with a common phase modulation drive,

(3) An RF path-switching sub-system,

(4) The Control and Monitor sub-system,

(5) The Calibration and Test sub-system.

These sub-systems are physically distributed between an inner cabin on the antenna pedestal (S-band components, adjacent to the antenna feed), outer equipment cabinets, also moving with the antenna, and the remainder within the Operations Control Centre about 250 m from the antenna pedestal base. Almost all of the OCC sub-systems operate at 50 MHz and below (Receive) and 65 MHz and below (Transmit). However, low-loss coaxial feeder is used between OCC and antenna.

Each receiver comprises three channels. The Reference/Telemetry Channel establishes carrier phase-lock, supports wide-band (Dump) telemetry and outputs video TLM. Two Angle-Error Channels detect the angular

deviation from antenna boresight in the X and Y planes relative to the antenna feed, and output error signals for feedback to the antenna servo drive to establish autotrack.

Each transmitter comprises a multiplier chain to raise the phase-modulated RF drive to the Uplink frequency and a stage of power amplification to produce the final RF level of 10 W into the diplexer.

An associated Translator unit samples the outgoing Uplink and converts this to the Downlink frequency, as a Test Input to the down-conversion stages of the receiver. The common phase-modulation drive is derived from a VCO, tripled and modulated with the Command Sub-Carrier which is itself modulated with command messages generated in the computer.

Control Centre Equipment

Equipment located in the Control Centre comprises a Unified S-Band (USB) TT&C set, PCM bit conditioners, a time standard, two wide-band instrumentation tape recorders and test gear. Control of the antenna and handling of the telemetry is accomplished with two desktop computers, which monitor the status and health of the Ground Station and satellite as well as generating the satellite commands for uplinking during each pass. At the modest data rates generated by small satellites, modern desktop PC's are quite capable of acquiring telemetry and processing it in real time, using hard disk as the primary storage medium.

Ground Station Performance

During the IRAS Mission, the antenna system successfully tracked over 1500 consecutive passes. Following the IRAS mission, the system was reconfigured for operations with the UK sub-satellite of the AMPTE mission (apogee 125,000 km), where passes over several hours duration were taken every day of the mission. Additionally, the ground station has been used to track several other spacecraft including LANDSAT-IV, IUE and EXOSAT. In all cases, command and control down to 8° elevation is possible, and for the majority of cases elevation down to 2¼° is possible.

GROUND STATION SOFTWARE

It is a traditionally held view that ground software has to cope with all of the problems which have been left by the hardware engineers. This may always be true to a certain extent, but the trade-offs for low cost need to be made in a detailed way early in the planning of a mission. A number of early decisions may, on the one hand, allow the in-flight component to be simplified, but at the expense of more complex operations and software. Alternatively, decisions on whether to adopt for instance, a standard telemetry format (CCSDS) would permit standardisation of ground software and minimal changes for successive missions. Thus the ground segments should always be considered to be an essential, integrated part of the mission right from the start.

Advantage can be taken of the increasing power and modularity of computers, both on the ground and in space. Thus, there is an opportunity to provide flexibility on-board the satellite to reduce data telemetry volumes, without the fear, which has existed up to the present, that irrevocable techniques could lead to at least partial mission failure if the instrumentation subsequently performs unpredictably. On the ground, sufficient checks can be built in to permit the use of automated passes, eliminating the need for expensive shift working. However, for this to be viable, the hardware design has to build in this requirement from the start and the control station hardware and software also.

The experience of RAL on many scientific missions has been that high efficiency and low costs can be achieved by using highly qualified and experienced staff throughout the design, development and operation of the ground systems. Despite the additional costs per staff year, there are significant gains in productivity from adopting this philosophy. Team members are selected to ensure a mixture of backgrounds in operations, formal computing training and research in the subject area of the spacecraft. Benefits are seen to be greater motivation, a strong understanding of mission objectives, which in turn makes the teams very adaptable and capable of proposing and implementing solutions to problems. Producing systems which the team themselves will have to run is a strong concentrator of the mind and the considerable costs of training and detailed documentation as the project progresses are significantly reduced.

It is not only in formatting that the adoption of standards can be of benefit. The existence of co-ordinated national facilities in the UK such as Starlink in the Astronomy area and the British Atmospheric Data Facility (formerly GDF) has also led to standardisation of data handling tools and data bases, allowing the reuse of software for analysis despite the widely differing instrumentation being flown. More could be done to exploit these universal tools, but a start has been made, although each project may have to accept compromises and possible lower performance if the goal of minimal new software is to be achieved.

CONCLUSIONS

It has been shown that mission costs for the Ground System can be significantly reduced by making just small compromises in data return, together with standardisation of hardware and particularly software subsystems, and in greater use of National facilities.

111135

N95-17550

Sep-14

354200

P-11

DESIGN OF GROUND SEGMENTS FOR SMALL SATELLITES

Guy Macé

MATRA MARCONI SPACE

31, Rue des Cosmonautes, Z.I. du Palays

31077 Toulouse Cedex - France

ABSTRACT

New concepts must be implemented when designing a Ground Segment (GS) for small satellites to conform to their specific mission characteristics : low cost, one main instrument, spacecraft autonomy, optimised mission return, etc... This paper presents the key cost drivers of such ground segments, the main design features and the comparison of various design options that can meet the user requirements.

Key words : Small satellites, Ground Segment, Mission Control, Data Acquisition.

1- Introduction

The Ground Segment for control of the spacecraft and for exploitation of their data represent a growing part in the space mission budgets. Therefore it has been considered important by Industry and by such Agencies as ESOC (1) and CNES (2) to review the state of the art for the Ground Segments that support the Small Missions, to understand the possible degree of optimisations and the cost implications.

Small satellite missions usually consist of one or two instruments aboard a small spacecraft thanks to technology progress. The development time frame and the programme costs are major drivers that will have to be fully considered for the definition of Ground Segment development and operations. The main driver to optimise the design while considering the cost constraints is thus to consider the space system (Figure 1) as a whole and to ***think integrated system***.

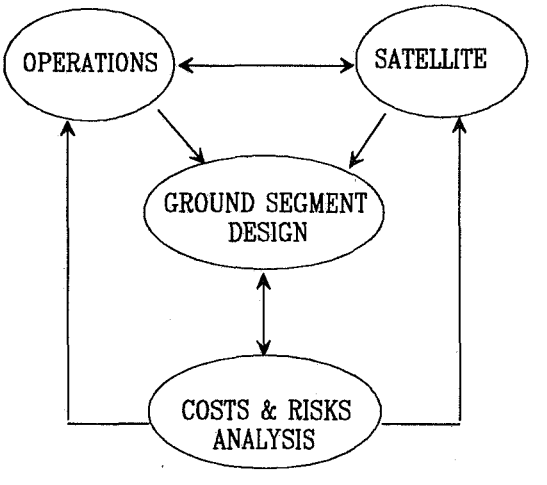


Figure 1 : Concurrent Engineering for Ground Segment design

The paper presents the cost drivers to be examined when designing a Ground Segment, the typical overall Ground Segment design characteristics and the main latitudes for optimisation. Finally the emphasis is placed on the definition of major Ground Segment elements, the Mission Control System and the Ground Stations to highlight where an optimum design can stand.

2- Mission related cost drivers

Since the cost constraints must be considered from the beginning, it is necessary to analyse where lay the cost drivers for ground segments. The cost drivers may vary from one mission to the other, may depend a lot on the category of service proposed by the mission : data for scientists, commercial service for telecommunications. However some general trends have been highlighted from examination of a number of conventional missions and of small missions.

For a typical observation small mission (Table 1), the GS design must consider with specific attention all requirements that may impact the number and definition of the Ground Stations and the Flight Dynamics functions on-ground. In this example, the Ground Station and Flight Dynamics elements have a sizing costs within the Ground Segment costs.

The accuracy of orbit restitution needed for payload data processing is a characteristic of this mission that directly impacts the flight dynamics processing on-ground. The ground station is a unique S-band station that supports both the Payload and the TM/TC housekeeping data. The other elements have a lower importance since either based on reuse of existing components or based on a limited development for a simple mission : for example, the mission planning function is limited due to only one payload instrument with no direct interaction with the users who require a systematic observation.

SMALL SATELLITE MISSION GS DEVELOPMENT COST (Most sizing cost driver graded 5)	Ground Stations	Comms Infrastruct.	Mission Control			
			Flight Dynamics	Mission Planning	TTC processing	Other functions
USERS REQUIREMENTS						
Mission purpose	4		3		1	
Permitted mission outage	1	1				
Availability of payload data	1	1	1			
MISSION REQUIREMENTS						
Mission Lifetime			1			
Satellite Pointing requirement	2		3		1	
RF Payload constraints	1					
SATELLITE DESIGN						
Orbit control	1		2			
Attitude control	2		3			2
TM/TC interfaces	1					
RF design	3					
Data rates/response times	4	1			1	1
Number/complex Ops modes			1		1	

Table 1 : Typical Cost Drivers for a small satellite mission (Observation)

The methodology followed was to identify what are the requirements that can impact the Ground Segment Design. In relation with the Users, the following requirements are identified as having a significant impact : the Mission Purpose that defines mainly coverage (image size, trajectory), resolution and duration of observation, the permitted mission outage expressed in possible interruptions of on-board service or observations, the availability of payload data criterion corresponding to the delay between the observation on-board and the time of reception of data at user site. The Mission Analysis then considers these requirements and the characteristics of a space system to derive such characteristics as the mission lifetime, the satellite pointing requirements or the RF payload constraints (e.g. number of ground stations, RF band selection). From the Mission Analysis a Spacecraft design will also impact the Ground Segment design with such requirements related to orbit control, attitude control, TM/TC interfaces definition, data rates and response times, RF links characteristics and link budget, number and complexity of operations modes that will have to be handled from the ground.

For comparison an observation conventional mission is considered : the GS costs are equally shared between the Ground Stations and Comms development, the Mission Planning and the Satellite Control Centre. For such a conventional mission, the main cost drivers were impacting most elements in a more distributed fashion as shown per Table 2.

The above elements must be given full consideration, when performing the necessary iterations between the Ground Segment design, the costs, the operations and satellite definition.

The main Ground Segment design characteristics for a small mission are now highlighted.

Conventional SATELLITE MISSION GS DEVELOPMENT COST (Most sizing cost driver graded 5)	Ground Stations	Comms Infrastruct.	**Mission Control**			
			Flight Dynamics	Mission Planning	TTC processing	Other functions
USERS REQUIREMENTS						
Mission purpose	3	2	2	3	1	
Permitted mission outage	2	1				
Availability of payload data	1	1				
MISSION REQUIREMENTS						
Mission Lifetime	1	1	1	1	1	1
Satellite Pointing requirement	2		3		1	
RF Payload constraints	2					
SATELLITE DESIGN						
Orbit control	1		2			
Attitude control	2		3		1	1
TM/TC interfaces	1				2	
RF design	3					
Data rates/response times	3	3			1	1
Number/complex Ops modes			1		1	

Table 2 : Typical Cost Drivers for a conventional satellite mission (Observation)

3- Ground Segment design

The Ground Segment design for a small mission must be such as to support the overall mission, but with much emphasis placed on costs aspects both for development and for the typical 3 years mission duration (2 to 5 years depending on the mission).

A first major trend of the design will be to maximise the use of existing components in the ground infrastructure : this trend limits the development costs and the maintenance effort since the hardware is based on off-the-shelf items and the software is flight proven in other programmes. This is why an important design effort will be dedicated to the overall architecture definition to identify the building blocks, to define their interfaces and the missing elements, and last but not least to react on requirements whenever it is felt to simplify the design while meeting the overall mission objectives.

To design a Ground Segment with building blocks will be more easily achieved if the system is built as a distributed system. And since cost efficiency for operations is an other major criteria, the collocation of the Ground Segment facilities must be enforced. Therefore a typical Ground Segment design for a small mission will be based with its components collocated around a Local Area Network (Figure 2) : Ground Station, Satellite Control System, Flight Dynamics, Mission Planning, Payload Preprocessing with the capabilities to communicate payload data to users either by mail or by communication links.

For small missions, the availability requirements can be less stringent than in conventional missions. No hot redundancy will be implemented as a rule : as experienced in conventional missions, it is costly since it requires more hardware, automatisms, specific procedures adding to the complexity of operations, documentation, training and maintenance.

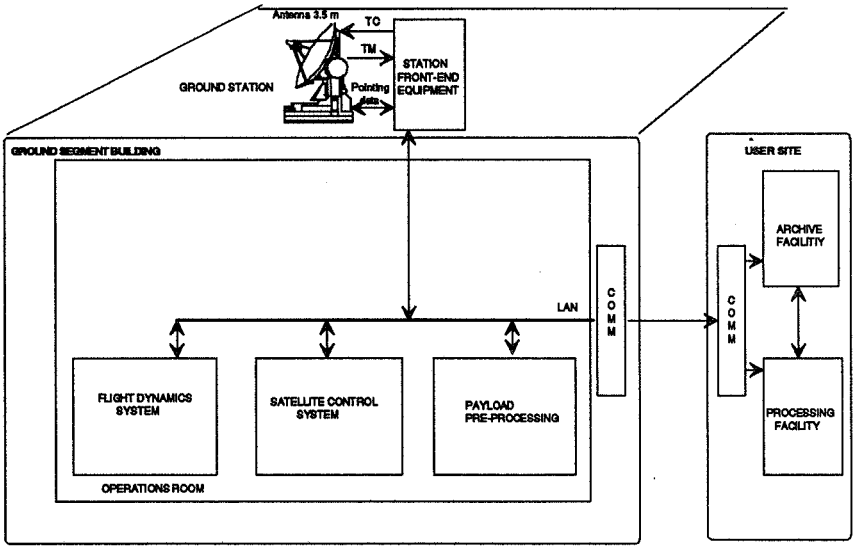


Figure 2 : Typical design for a Small Satellite Ground Segment

The other main features of the GS design for small missions are : collocation of facilities, reduced staffing, use of proven off-the-shelf hardware and software, automation of routine operations, compliance to standards (e.g. CCSDS and ESA COES) to enforce further commonalties for reuse. Table 3 hereafter compares the main features for a Low Cost ground segment option, for a Lower risk ground segment option and for the design attached to a conventional mission. The Lower Risk option will mainly differ from the Low Cost option in the operations concept that will provide a higher security level for operations and a higher mission availability.

How the Lower Risk option can best meet the overall mission requirements and what are the possible risks attached to a Low Cost option are given a preliminary answer in the following section.

	OPTION Low Cost	OPTION Lower Risk	CONVENTIONAL MISSION
MISSION AVAILABILITY	80-90% : normal working hours (+ on-call for w.e.)	> 90% : 7 days/week + on-call at night	> 99.9% : 7 days/week + 24 Hours/day
STAFFING	2 or 3 for all tasks	3 for ops + part support	6 shift x 2 for ops + important part support
FACILITIES DISTRIBUTION	Collocated	1 or 2 sites	Several sites
STATION Design	Standard products, small antennae	Idem + reuse of a station network	New development
MCS Design	Reuse existing packages Minimum adaptations	Reuse existing packages More tailored to ops	New development Many ops requirements

Table 3 : Main features for the overall GS Design

4- Ground segment optimisation

The allocation of costs for a Ground Segment must be carefully considered to select design options that will maximise a mission return criteria, i.e.. the amount & quality of data versus the investment.

Typical costs allocations are shown for a Telecomms conventional mission (Figure 3) and for a Small Mission (Figure 4). The total GS cost includes the following costs : Ground Stations & Comms, Mission Control System, Prelaunch operations (Flight procedures preparation, MCS database definition and validation, ground and flight operations validation and rehearsals) and a normalised period of 3 years operations.

In the conventional mission example, the Ground Stations costs were important due to the number of antennae considered and to a 11 meter antenna supporting accurate angular measurements. The ground station cost for the small mission was limited since VHF/UHF data links were considered both for payload messages (less than 20 Kbps) and for housekeeping TM/TC with no ranging requirements imposed on ground other than processing the on-board GPS transmitted data.

With these characteristics a significant cost of the small mission Ground Segment corresponds to the operations costs. Therefore it is important to analyse how these costs can be reduced and how this reduction can impact the GS availability and the risks for operations.

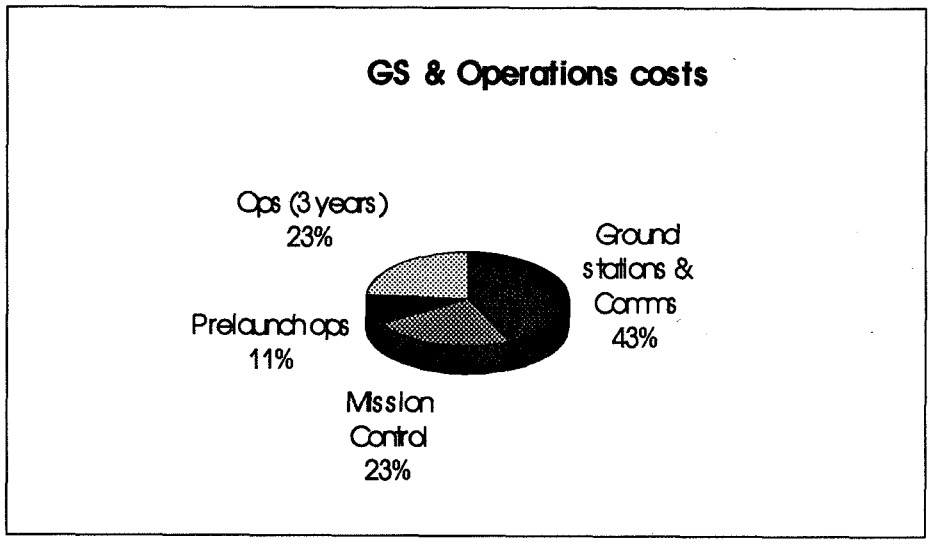


Figure 3 : Cost break-down for a Telecomms Conventional mission

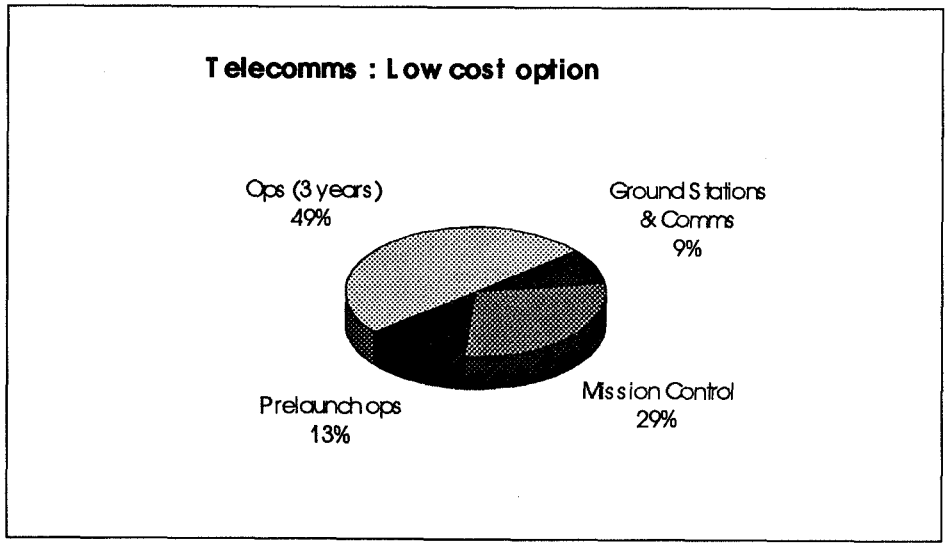


Figure 4 : Cost break-down for a Telecomms Small mission

Figure 5 below presents a typical example of a GS availability as a function of the GS total cost for typical GS options with different design, maintenance and staffing orientations. The availability was computed using equipment failure rates and mean time to repair as checked during several years of operations and the time for intervention considered for exploitation. The main difference between options availability characteristics proceeds from this time for intervention, i.e. time spent between the occurrence of a failure and the staff performing failure detection, investigation and replacement of the faulty equipment. With today's GS equipment high reliability figures, it is the exploitation characteristics that mainly drive the GS availability.

In the Low Cost option, staff is only available during working hours. In the other option (Low cost/24 Hours, ESOC reuse, Low risk) only the design is different when the staff is available day and night, including week-ends to react to any ground failure detected with spare equipment available for ground equipment.

The Low Cost option is interesting since it presents a substantial cost advantage of about 3 MAU with respect to the other options and a higher mission return per cost unit (defined as the amount of data a user can expect over the mission duration, and therefore proportional to the GS availability figure). From the user perspective the mission return is 2% lower but the sensibility of theses availability figures and their statistical meaning show that this will have little effect on the user satisfaction wrt the amount of data acquired over the 3 years. Therefore the 24 Hour Manning Low cost option does not bring a significant advantage to be considered.

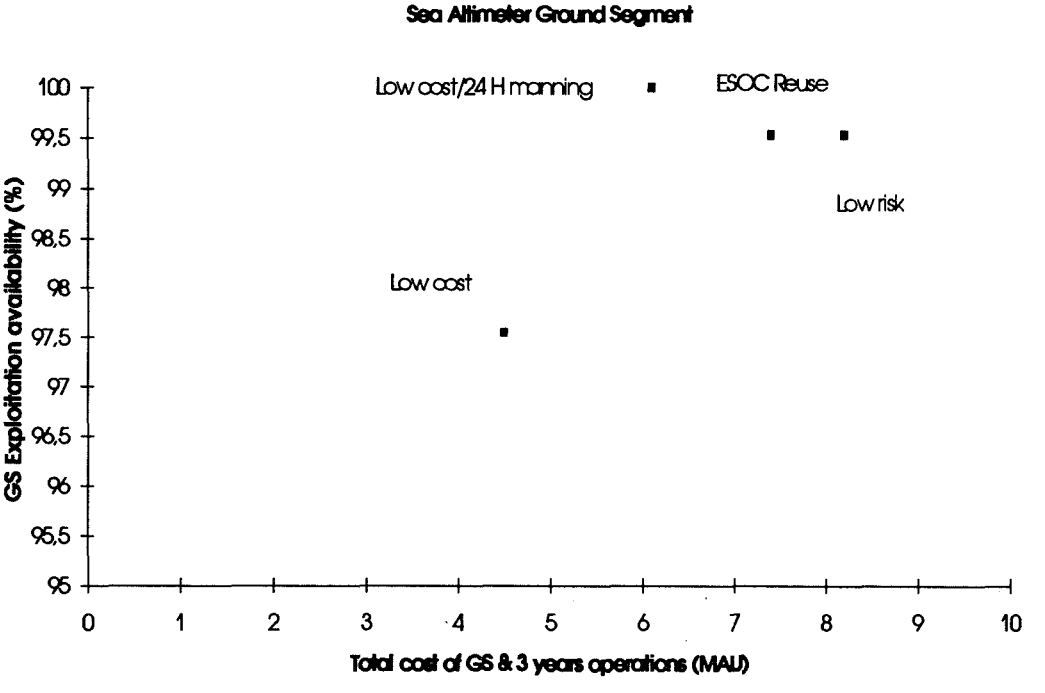


Figure 5 : Staffing & Maintenance impact on GS availability and cost

An alternative could be to change some characteristics so that the mission return be much lower at a significant saving. From the cost drivers analysis this orientation would bring a minor cost saving with a substantial degradation of mission return and a higher risk: for example operators only working upon automatic anomaly detection could be felt more risky without a significant advantage.

This is why it is of the utmost importance to appreciate the risks induced by the Low cost approach in comparison with the more conventional approaches. The following elements contribute to the risk specific to the Low Cost option and not supported by the other options :

- The whole expertise (spacecraft and ground) is supported by a 3 engineers staff coming from the spacecraft development team. In the other options an operations support infrastructure is identified that support spacecraft contingency analysis or such expertise domains as flight dynamics, or ground equipment maintenance. The difficulty consists in the level of skills required from this 3 engineers team and whether they can efficiently support contingency cases. The typical spacecraft autonomy of 1 week, the on-board securities and the expertise gained by the staff during spacecraft development should compensate most of the risk.

- The simulator is not foreseen in the low cost option and limited testing will be performed with the spacecraft (or its engineering model) on ground. A number of operations will not have been tested prior to launch : this could be accepted if the spacecraft is safe, robust to ground errors and that a number of spacecraft specialists are available at the beginning of life so that operations imperfections be detected quickly and correct procedures be validated. The beginning of spacecraft lifetime would lead to less data availability, what could be accepted since a first period is often considered for calibration and with full support of the spacecraft engineering team. However it is strongly recommended to keep the simulator even in a low cost option, since the foreseen benefit for operations security is important with regards to the cost of such a recurring product which represents less than 3% of the total mission cost.

Each of the GS components are further examined with emphasis on the major design options.

5- Mission Control System

The Mission Control System (MCS) is composed of the following functions : TM/TC function with real time control and satellite performance analysis, flight dynamics and mission planning. The main outcome for small missions will be the reuse of existing software packages. Most packages are running now on Unix workstations and the integration can be limited when only exchanging few data files.

An important trend to reach additional cost saving for small missions, will be to consider all Ground Systems needed in a programme : with reuse of existing EGSE and MCS building blocks, it is now envisaged to build a "Universal Test Bench" that can be used in all stages of the satellite development and operations.

Figures hereafter (Figures 6 to 8) examine the relative development costs for observation missions : a Conventional mission, a Sea Altimeter small satellite mission and a Cartographic small satellite mission.

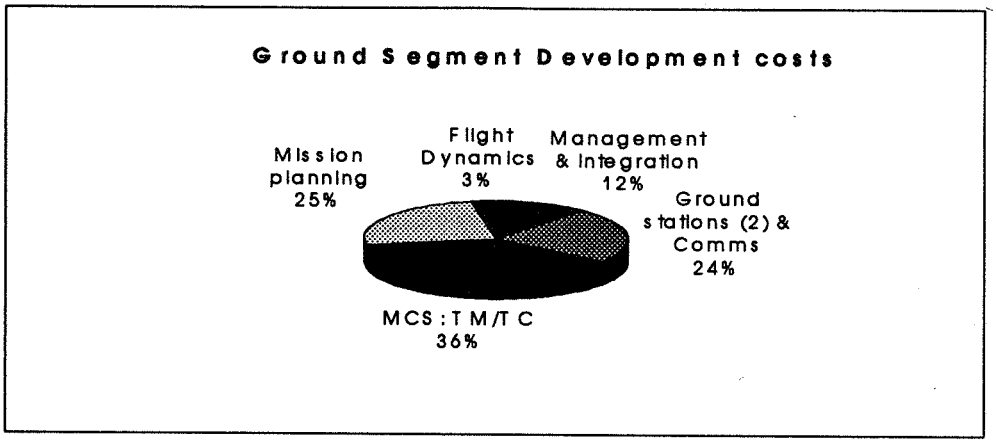


Figure 6 : MCS costs for a conventional mission (Observation)

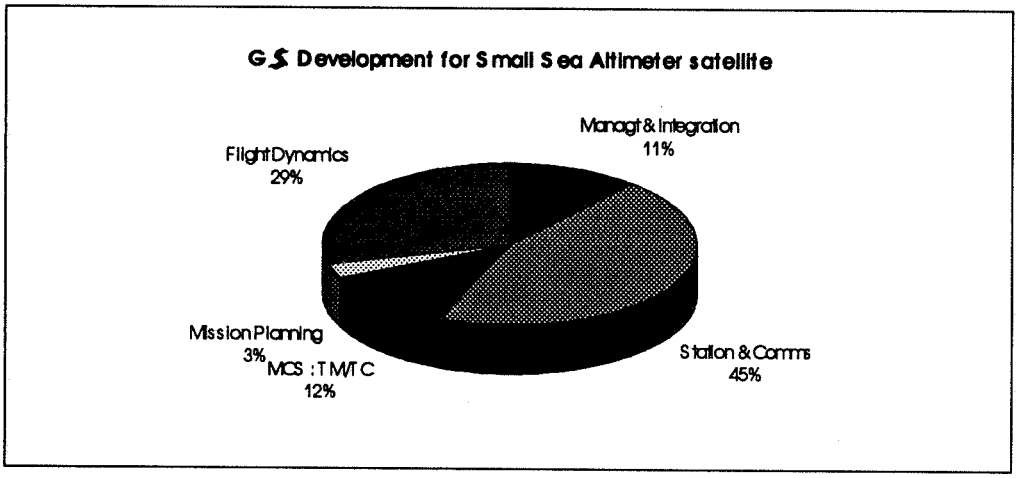


Figure 7 : MCS costs for a Sea Altimeter small mission

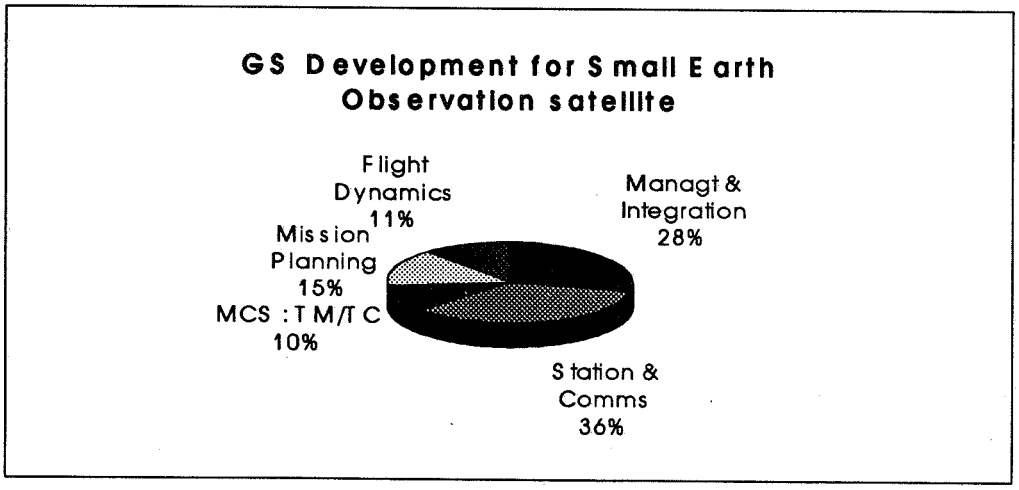


Figure 8 : MCS costs for a cartographic small mission

The above examples show that with this strategy of reuse with minimum adaptations, the amount of the Mission Control System in the overall development costs is lower than for conventional missions. Depending on the ground station characteristics, the MCS can weight 36% to 44% of the Ground Segment development cost.

6- Ground stations and Communications

The Ground Stations and the ground Communications part of a Ground Segment is usually a sizing ratio of the total development cost. Therefore special attention must be granted to the characteristics that contribute to the costs (Table 4).

The Antenna itself in the ground station can be the sizing cost element when high performances are required from the specified bandwidth and data rates. This is why a ground and board optimisation must take place to review the data rates with respect to user requirements, to review then the budget link requirements, to retain only one system of communications both for payload and housekeeping operations. The choice of the frequency bandwidth (X band, S band or lower band such as UHF) and the mission orbit characteristics will then make the price of the antenna. A common characteristic of many small missions is that only one antenna system is used for communications of both payload and housekeeping data of the satellite platform.

The RF equipment and Baseband equipment are then to be considered in the cost but they are usually off-the-shelf equipment with high reliability figures : the Monitoring & Control equipment can limit itself to the set-up of equipment configuration and to support investigation and no longer as a procedure driven system to act on the redundancies and switches. In addition, for a low cost solution, a new range of VSAT equipment is available at a lower cost with possibly lower reliability performances that can be adequate for small missions. As for other elements of the Ground Segment, a major contributor to costs, as experienced in passed conventional programmes, is the development of specific equipment or of new technology when off-the-shelf equipment exists.

COST DRIVERS	SMALL MISSION	CONVENTIONAL
OFF-THE-SHELF EQUIPMENT	Systematic	cost of technology changes
ANTENNA & RF	Only 1 station for payload and data	Network of stations Sizing costs
RANGING	Use of GPS	Can be costly on ground
LEOP	Interface with existing network Transportable TTC station S/C autonomy wrt LEOP	Specific requirements
COMMUNICATIONS	Collocation on LAN as baseline Files transfer at low data rates	Usually low relative costs

Table 4 : Cost drivers for Ground Stations and Communications

An other important cost item can be related to the requirements imposed on ground to perform the ranging. In conventional missions these requirements imposed range equipment in the station, or large antenna with complex mechanics for accurate angular measurements. For small missions these requirements are alleviated either by lower performance requirements on orbit determination or by the availability on-board of GPS or other equipment that provide orbit measurements.

Finally communications can be achieved more simply than in conventional missions with relaxed requirements for data timeliness that defines the time spent to provide the user with data. Depending on missions, simple mail procedures can be accepted or an electronic file transmission system using standard networks (e.g. INTERNET or other national or international networks) can be used. To decrease the communications costs, one solution if feasible may consist of having users collocated at Ground Segment site and receiving their data on the LAN. The communications analysis can impact the place where the data demultiplexing can be performed : either at Station or at Control Ground System level.

7- Conclusion

Small missions constraints enforce a new approach for development of both the satellite and its associated ground systems. With due consideration to existing technology and products, the project team must review in an iterative way the requirements, design and costs implications on both the satellite and the ground systems for satellite testing and for operations. This new approach can be summarised as the Integrated System Approach relying on a new ground system means, the "Universal Test Bench" which building blocks will be used according to satellite development and operations stages.

ACKNOWLEDGEMENTS

Mr. Van der Ha (ESOC), Mr. Oberto (MMS)

REFERENCES

(1) Small Satellite for Ground Segment and Operations : ESOC study conducted in 1993-1994 under Mr. J. Van der Ha 's responsibility

(2) Ellips study : CNES study conducted in 1994

111136 N95-17551

354202 p. 10

THE SAX ITALIAN SCIENTIFIC SATELLITE. THE ON-BOARD IMPLEMENTED AUTOMATION AS A SUPPORT TO THE GROUND CONTROL CAPABILITY

Andrea MARTELLI

Alenia Spazio S.p.A. - Turin Plant
Corso Marche 41
10146 Torino - Italy

ABSTRACT

This paper presents the capabilities implemented in the SAX system for an efficient operations management during its in-flight mission.

SAX is an Italian scientific satellite for X-ray Astronomy whose major mission objectives impose quite tight constraints on the implementation of both the space and ground segment. The most relevant mission characteristics require an operative lifetime of two years, performing scientific observations both in contact and in non-contact periods, with a low equatorial orbit supported by one ground station, so that only a few minutes of communication are available each orbit.
This operational scenario determines the need to have a satellite capable of performing the scheduled mission automatically and reacting autonomously to contingency situations.

The implementation approach of the on-board operations management, through which the necessary automation and autonomy are achieved, follows a hierarchical structure.
This has been achieved adopting a distributed avionic architecture. Nine different on-board computers, in fact, constitute the on-board data management system. Each of them performs the local control and monitors its own functions whilst the system level control is performed at a higher level by the Data Handling Application software.
The SAX on-board architecture provides the ground operators with different options of intervention by three classes of telecommands. The management of the scientific operations will be scheduled by the Operation Control Centre via dedicated operating plans.

The SAX satellite flight model is presently being integrated at Alenia Spazio premises in Turin for a launch scheduled for end '95.
Once in orbit, the SAX satellite will be subject to intensive check-out activities in order to verifiy the required mission performances. An overview of the envisaged procedure and of the necessary on-ground activities is therefore depicted as well in this paper.

INTRODUCTION

The SAX satellite is part of a scientific program whose objective is to observe celestial X-ray sources in the broad energy band from 0.1 KeV to 300 KeV. The SAX mission has been planned to achieve a systematic, integrated and comprehensive exploration of galactic and extra-galactic sources, providing significative improvements for more complete and extensive studies in X-ray astrophysics.
SAX is a joint program managed by the Italian Space Agency (ASI) and by the Netherlands Agency for Aerospace Programs (NIVR) coordinating the scientific interest of the Italian and Dutch scientific community and funding an international industrial team whose overall organization structure includes:

- Alenia Spazio as main contractor for the Space Segment
- Telespazio as main contractor for the Ground Segment
- Martin Marietta - Commercial Launch Services - as main contractor for the Launch Vehicle
- Italian and Dutch Scientific Institutes as Scientific Consultancy.

The SAX Payload hosted on-board consists of the following six scientific Instruments (Ref. 1):

- Low Energy Concentrator Spectrometer (LECS) whose task is to perform X-ray spectrometry/imaging in the 0.1-10 KeV energy range
- Medium Energy Concentrator Spectrometer (MECS) whose task is to perform X-ray spectrometry/imaging in the 1-10 KeV energy range
- High Pressure Gas Scintillation Proportional Counter (HP-GSPC) whose task is to perform X-ray spectrometry in the 3-120 KeV energy range
- Phoswich Detector System (PDS) whose task is to perform X-ray spectrometry in the 15-300 KeV energy range and gamma-ray burst monitoring in the 60-600 Kev energy range
- Two Wide Field Cameras (WFCs) whose task is to perform X-ray spectrometry/ imaging in the 2-30 KeV energy range.

PRECEDING PAGE BLANK NOT FILMED

PAGE 836 INTENTIONALLY BLANK

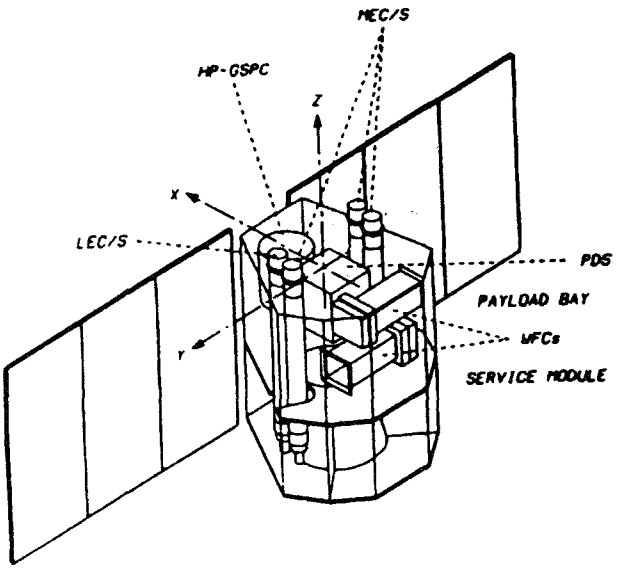


Fig. 1 - Satellite Overall Configuration

The WFCs are mounted along the +Y and -Y satellite axes, allowing an observation of a wide sky portion, whereas all the other Narrow Field Instruments are aligned along with the +Z axis. Fig. 1 illustrates the satellite overall configuration.
The SAX pointing capability ensures a target measurement accuracy of 1 arcmin and a pointing of 3 arcmin for a maximum of 10^5 seconds, i.e., one day.
All the project design has been developed to cope with a mission of at least two years preceeded by a commissioning phase period, estimated to extend for about eight weeks.
The satellite is currently in a very advanced C/D phase. The Flight Model is under integration as the last step of a system integration and test campaign involving the developing of a Structure Model, an Engineering Model, and a Software Verification Facility. The launch will take place by end '95 with an Atlas Centaur vehicle. The SAX Ground Station will be located in Singapore and will be connected via Intelsat to the SAX Operation Control Center and the Scientific Data Center, both located in Rome.

MISSION CHARACTERISTICS

The major constraint entailed by the scientific objectives requires a satellite orbit such that the background particle radiation for X-ray detection be very low and the effects of radiation from the South Atlantic Anomaly region be reduced. This leads to the choice of a circular low Earth orbit at a 600 Km altitude - Begin of Life (450 End of Life) - and an inclination of about 4°. The orbit period is thus of 97 minutes with an alternance of 60 minutes of sunlight and 37 minutes of eclipse.

One single ground station, located near the equator, will support the mission offering satellite visibility each orbit. The coverage period is anyway no longer than 11 minutes so that about 90% of orbital life is out of visibility.
The pointing domain is limited by the allowed sun incident angle range on the satellite solar array surface. A maximum of 30° (with occasional excursions to 45°) inclination is allowed with respect to the sun direction to ensure a proper battery charge. This implies a pointing domain for the Narrow Field Instruments limited within a band in the sky 60° wide available for observation each orbit (except some possible occultations by celestial bodies). In a one year period, the whole sky will be observable for a scientific activity that can be estimated as performing between 2000 and 3000 independent observations (Ref. 2).

THE SYSTEM ARCHITECTURE

The above introduced operational scenario determines the need to have implemented on-board the capability of supporting, in an autonomous way, the execution of on-ground pre-defined mission plans. That also requires the on-board architecture to manage the nominal activities as well as the pre-conceived anomalies, in all the mission phases, taking into particular account that most of the mission is out of the ground coverage.
The implementation approach of the required operation management is based on an avionic architecture which makes extensive use of a distributed on-board intelligence (Ref. 3). Nine on-board intelligent terminals constitute the SAX system architecture as shown in Fig. 2 (see following page).
Each of them performs the autonomous control of the relevant subsystem (S/S) local functions including the surveillance of its health status. The control of the system overall activity is assigned to a higher hierarchical level and is implemented in a Central Terminal Unit (CTU). The CTU is devoted to coordinating and controlling the Data Management and Communication System as well as to managing the system nominal operations and to undertaking the system level recovery actions. A set of non-intelligent subsystems, including the Telemetry Tracking & Command S/S, the Reaction Control S/S and the Electrical Power S/S are placed under the direct control of the CTU via serial lines through a Remote Terminal Unit.
The interprocess communication is based on the ESA standard serial digital bus arbitrated by the CTU and composed of:

- Interrogation Bus for CTU to local terminals interrogations
- Response bus for local terminals to CTU transmission of Housekeeping Data (HKD)
- Block Transfer Bus for Scientific Instruments to CTU transmission of Scientific Data.

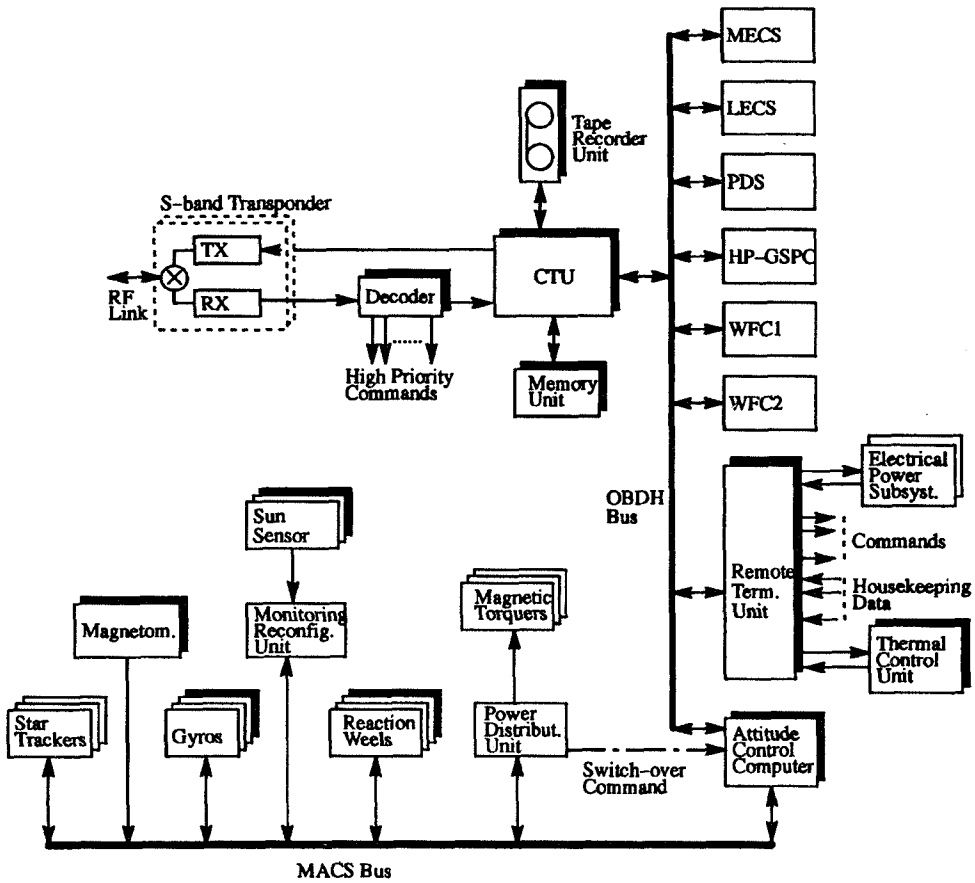


Fig. 2 - System Architecture

The data communication protocol is designed to ensure a collection of about 16 kbit/sec of HKD from the satellite subsystems and science instruments and up to 100 kbit/sec of scientific data. Two different formats of HKD can also be selected: one essential format including a basic set of SAX HKD, one intensive format including some extra information on hot redundant units and Data Handling traced operations. All the data gathered in non-visibility are temporarily stored on a dedicated tape recorder, with a capacity of 510 Mbits, until requested to be dumped to ground during the coverage periods. Two channels are implemented to dump to ground the satellite telemetry in High Bit Rate mode:

* channel "I" for dumping the real-time collected telemetry at 131 kbps
* channel "Q" for dumping the tape recorded data at 917 Kbps.

A 16 Kbps link is also available to implement a Low Bit Rate transmission mode.
The telecommand bit rate allows an uploading of 2 Kbps, that is about 20 frame instructions/sec.

ON-BOARD REDUNDANCY CONCEPT

The SAX mission characteristics have led to a system design with a high degree of reliability to cope with so long an autonomous lifetime.

All the spacecraft S/Ss are designed to be single failure tolerant whereas the Scientific Instruments implement redundancy only at interface level. Critical on-board items (e.g. receivers, decoders, gyroscopes, power units, protected memory) all operate in hot redundancy. In this context single spacecraft unit malfunction does not affect the nominal mission performance.

The intelligent subsystems - i.e., On-Board Data Handling (OBDH), Attitude and Orbit Control S/S (AOCS), Thermal Control S/S (TCS) - are based on a fully redundant architecture. Each of their unit classes includes one redundant item so that one fatal failure can be recovered by properly activating this redundancy.
The Scientific Instruments, not having implemented any internal redundancy, perform only a reduced Failure Detection and Isolation function for specific problems.

All the on-board computers maintain at least the software (SW) basic functions stored in Programmable Read Only Memories so that any reset/switch-over cannot cause the loss of the code, as it is downloaded from PROM to RAM any time a (re)-initialization takes place. Embedded circuitry for error detection and correction of corrupted memory cells by single event upset as well as a watch-dog circuitry for autonomous reconfigurations are provided in all the intelligent subsystems.
All the data considered critical for the proper on-board autonomous maintenance of the mission, in any nominal or contingency situation, are dynamically maintained in dedicated Protected Memory Areas. According to the relevant OBDH and AOCS performed control, this data set is so classified and grouped:

- OBDH Application SW (A/SW) vital data, including the Solar Array deployment status, the launcher separation status, the system and some critical S/S items active configuration (e.g. transmitters, battery discharge regulators, reaction control S/S branches, etc.)
- OBDH Basic SW (B/SW) vital data, including the launcher separation status, the OBDH active unit configuration, the redundancy management data
- Time Tagged commands to be scheduled at their own pre-set time
- Real Time commands to be executed at CTU switch-over
- AOCS S/S active configuration and launcher separation status, maintained in dedicated AOCS solid state latches.

The failure management of non-intelligent S/Ss is performed at centralized level by the OBDH A/SW. Some exceptions deviate from this general approach:

* Power S/S performs the failure management for its own units;
* the hydrazine flow control valves are under control of the AOCS when it makes use of thrusters;

and these are driven by time intervention constraints.

MISSION PHASES

The SAX mission can be divided into four overall mission phases.

- ***Launch Phase***
  LP begins at spacecraft power-on just before vehicle lift-off and extends to the physical separation between the launcher and SAX. In this phase the S/Ss are initialized and perform a continuous control of the powered units. No attitude manoeuvre is of course executed as the AOCS is in its initialization mode until the separation. The on-board produced data are stored on the tape recorder, just after the launch vibrations terminate, for later transmission to ground.
- ***Commissioning Phase***
  This begins at the SAX-launcher separation and extends to the completion of all initial in-orbit tests and calibrations. As a first step, it consists of an Early Orbit Phase which comprises a short post separation coast period, a reduction of any residual S/L body rates and a subsequent Sun/Earth acquisition period. Upon successful completion of these activities the deployment of the Solar Arrays is autonomously operated.
  The commissioning of the satellite shall proceed with an initial health check-out continuing with systematic functional checks of all the subsystem nominal functionalities.
  The Scientific Instrument activation and functional verification shall be operated as a last step. Some overlaps between the two shall be necessary for a complementary check-out of both the spacecraft and the Scientific Instruments. All these operations shall be initiated by ground and supported by the on-board SW tasks.
- ***Operational Orbit Phase***
  This phase covers the period of the satellite's useful scientific lifetime. It shall be nominally two years and shall be characterized by routine scientific operations. The satellite design shall, anyway, allow an extention of the mission beyond the nominal period up to a total four years lifetime.
- ***End of Life Phase***
  This phase covers the period when SAX is no longer capable of producing useful scientific information due to either component degradation or altitude decay.

SATELLITE MODES

The system mode design has been structured to cope with all the SAX mission phases (Ref. 4). The satellite modes - implemented with a direct correspondence with the AOCS modes - drive all the on-board autonomous operations. Their transitions can be initiated either upon ground commands or at the occurrence of automatic fallbacks caused by system autonomous emergency re-configurations. The SAX mode transitions diagram is reported in Fig. 3 (see following page).
The mission/science support modes are the principle configuration to support the scientific activity. The default/safety modes correspond to the main operative configurations to be assumed in case of interim science activity or on-board emergency. Two further modes support special operations during the launch phase and during the orbit raising manoeuvre - if ever needed.

- **Satellite Launch Mode (SLM)**
  Routine operations are performed to ensure an health satellite status ready to operate just after the SAX-launcher separation.

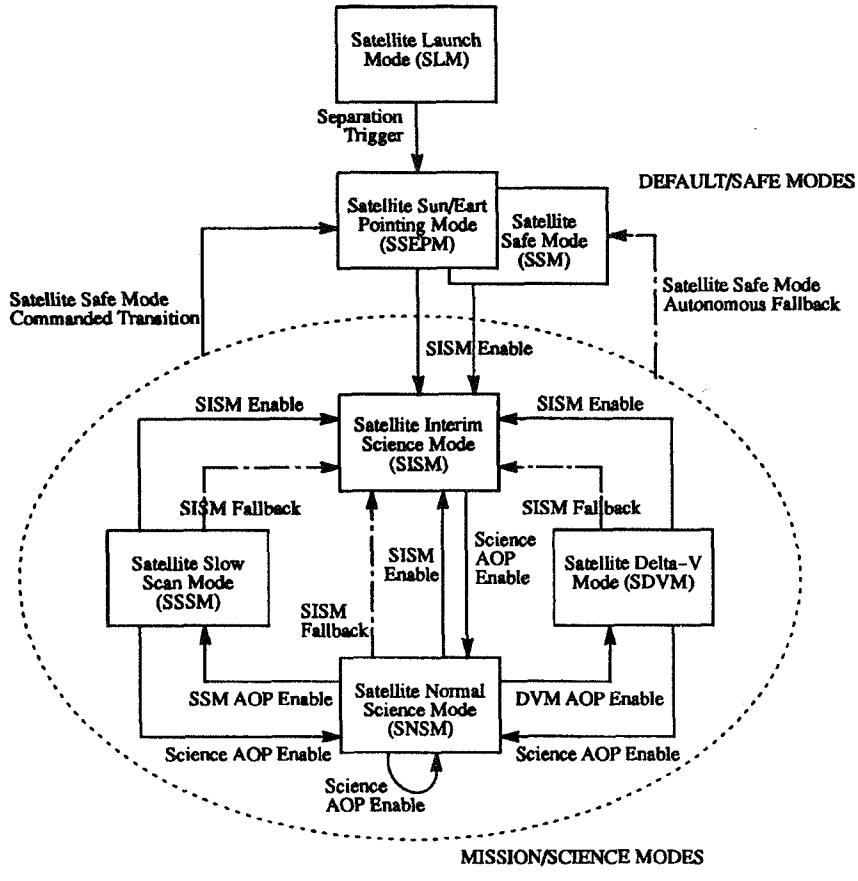


Fig. 3 - Mode Transition Diagram

The produced telemetry is recorded only after the vibration level is reduced - the activation of the on-board tape recorder is made by a time tagged command.

- ***Satellite Sun Earth Pointing Mode (SSEPM)***
  This mode is automatically entered either at SAX-launcher separation or as fall-back from the other Nominal Satellite modes. Purpose of this mode is to maintain the satellite in a 3-axis stabilized attitude optimizing the sun incidence on the Solar Arrays. As this mode is entered from the separation, it has to accomplish a very critical sequence of operations most of them to be performed autonomously since they are out of the ground coverage. The major operations are initiated by the OBDH and AOCS software that have to coordinate the safe attitude acquisition with the Solar Arrays deployment. Trigger of these operations is the SAX-launcher separation, detected by a dedicated fully redundant hardware circuitry and sent to both the S/Ss.

- ***Satellite Interim Science Mode (SISM)***
  This mode configures the SAX satellite in a accurate three-axes stabilized attitude making use of one star tracker, besides all the other used sensors. This fine pointing helps in keeping a default attitude (e.g. Polaris pointing) and in fastening attitude transitions to scientific modes.

- ***Satellite Nominal Science Mode (SNSM)***
  The satellite remains in this mode while operating the planned scientific observations. A very fine pointing is made by use of the AOCS star trackers. All the scientific data produced by the Scientific Instruments are collected by the OBDH according to a dedicated polling algorithm.

- ***Satellite Slow Scan Mode (SSSM)***
  This mode will mainly be used to perform calibrations of Non-Imaging Scientific Instruments by performing sequential slews across a known target.

- ***Satellite Delta-V Mode (SDVM)***
  This mode is designed to cope with the altitude decay, raising the satellite orbit in the case the SAX altitude decreases below the 450 Km.
- ***Satellite Safe Mode (SSM)***
  This mode is entered upon detection of specific system-level failures. A safe attitude is then maintained by the AOCS pointing the Solar Array surfaces toward the sun and aligning the satellite with the earth magnetic field.

OPERATION MANAGEMENT STRUCTURE

The management of the SAX system operating modes is implemented by a multi-level hierarchical structure (Ref. 5) involving, in increasing priority:

- the S/L Subsystems and Scientific Instruments
- the OBDH Application Software
- the Ground Operation Control Centre.

To the upper levels is assigned the task of initiating the scheduling of system level functions as well as the capability of controlling and overriding the lower level decisions. On the other hand, the main nominal operations autonomously performed at local level allow the proper control and setting of the relevant S/S. In particular, the intelligent terminals and Scientific Instruments are designed to be fully autonomous in performing their relevant tasks so that they can in principle continue operating consistently without any external intervention. Few inputs are, in fact, needed only for tuning their performances and their configurations with respect to either the system configuration or the current mission characteristics.
Each of the intelligent subsystems also performs a Failure Detection, Isolation and Recovery (FDIR) management on its own, keeping under control the configuration, functioning and health status of all its relevant units. In the case a malfunction is detected, the fault unit can be substituted by the redundant one. If the main S/S computer is affected an automatic switch-over takes place. The redundant intelligent unit will then be initialized assuming a safe mode of functioning.
The Scientific Instruments, not having a redundant architecture, adopt a self disabling policy, in particular, against a too high level of particle radiation able to damage the instrument itself.
Purpose of the OBDH A/SW is to keep under control all the subsystem level operations; that implies a system supervision to ensure the proper nominal/safety satellite consistency. What has been assigned to the A/SW is the role of the on-board coordinator of all the major flight operations between themselves and with respect to the ground scheduled plans.

It is in particular devoted to:

* perform Solar Array deployment following the launcher separation and sun/earth acquisition
* inform the AOCS of the new inertia matrix to be used after the Solar Array deployment
* support the distribution and enabling of operating plans to the AOCS and Scientific Instruments
* support the Ground-to-Satellite link acquisition and downlink telemetry operations
* enable/disable power resources to the non-essential satellite loads, i.e., Scientific Instruments, Reaction Control S/S, thermal control heaters
* perform the deployment of the Scientific Instrument baffles
* manage satellite mode transitions as a consequence of
  - Intelligent S/S switch-over
  - AOCS mode fallbacks
  - Power S/S protection triggering
  - Scientific Instrument particle over-radiation detection.

All the A/SW operations are coordinated and synchronized by the proper activation of dedicated pre-defined command sequences and command loops. These can be activated either by ground or autonomously to accomplish the above introduced operation set. The OBDH A/SW core is based on three principal modules acting as the kernel of the A/SW architecture, as illustrated in Fig. 4 (following page).

- ***The Mission Manager:*** it monitors the mode transitions of all the subsystems and instruments which require corrective operations. It is based on a mode transition table indicating all the actions to be undertaken at the occurrence of S/S mode transitions. It in particular specifies the safe configurations to be adopted in case of some critical mode fall-backs. It also drives the enabling/disabling statuses to be applied to the A/SW controls, as a function of the satellite mode configuration.
- ***The Fault Manager:*** it cyclically checks a pre-defined sub-set of the on-board produced monitors to undertake subsequent actions to isolate and/or recover the related problems. The data set includes all the mission critical on-board items, provided on a periodic basis and kept under control by means of a table driven FDIR manager. The control is performed by periodic tasks scheduled every second.
  A cross-check is then made between the measured values and their relevant expected ranges. Any discrepancy activates a direct recovery action on the non-intelligent S/S with possible extension to a system reconfiguration in the case the malfunction can severely affect the system performance.

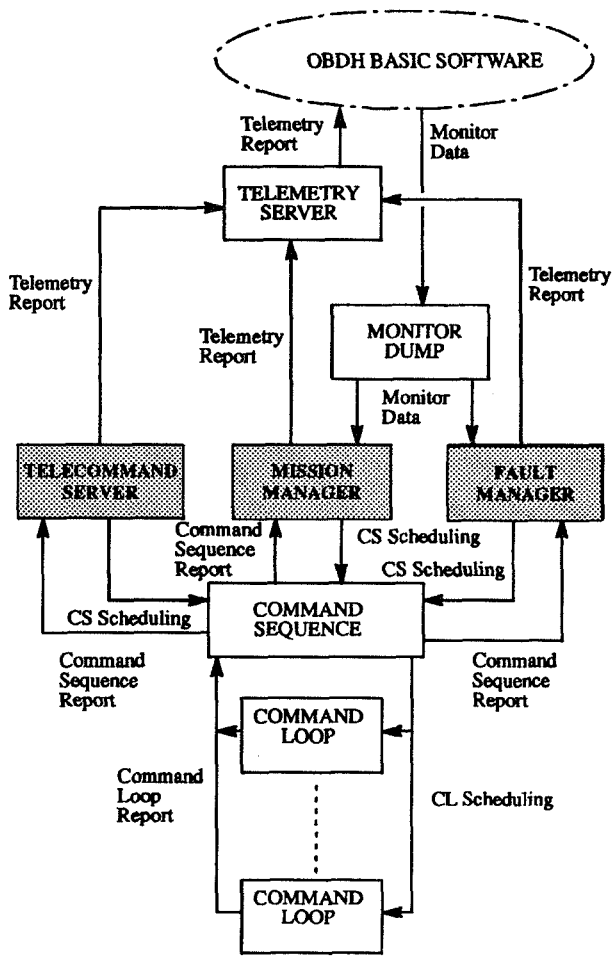


Fig. 4 - OBDH Application Software Architecture

- ***The Telecommand Server:*** it manages the ground initiated operations distributing and verifying the command execution.
  It furthermore applies a consistency checks on the ground up-loaded requests against the current system configuration. In the case a conflict is detected a report is provided in the A/SW telemetry but no action is undertaken until an explicit ground override is operated.

The top level of the SAX operations is, of course, a ground task. It is responsible for acting on the satellite configuration in order to set it up properly to accomplish the planned scientific observations. It has therefore to operate on both the spacecraft and the Scientific Instruments. Besides, routine maintenance operations have to be scheduled to cope with the orbit and mission events/constraints.
Some of the more frequent operations are anyway related to the orbit contact management whose ground intervention extends to:

* linking acquisition via the proper activation of the transmitter linked to the ground facing antenna. This is done by a time tagged telecommand acting on a dedicated A/SW command sequence which is devoted to verifying the correct functioning of the on-board link chain
* enabling the telemetry transmission to ground once the down-link carrier is obtained. This concerns the real-time telemetry and, on request, the on-tape data stored in the non-coverage period
* restoring the on-board data recording and termination of the link before the end of the contact period
* command the issuing of the on-board time samples for on-ground data correlation
* managing the antennae switch-over as the coverage concerning the facing antenna is going to end. Note that two hemispherical antennae are implemented on SAX in order to cover the whole space around the satellite.

Less frequent operations are related to scientific observation management. That involves:

* changes of the satellite attitude via dedicated AOCS Operating Plans
* changes of allowed pointing domains
* changes of Scientific Instrument operating modes
* Scientific Instrument configuration management, in particular at any entry/exit of the South Atlantic Anomaly.

Other infrequent operations are related to performance or maintenance aspects. In this context, the ground control centre shall periodically monitor the satellite dumped telemetry to keep under control the actual on-board configuration. It can therefore intervene for recovering any on-board assumed safe mode or, simply, for tuning some control parameters such as, for example, battery *End Of Charge* and/or *End Of Discharge* levels, thermal loop thresholds and/or enabling/disabling flags, sun vector and attitude quaternion values, etc..

GROUND COMMANDING CAPABILITY

The ground commanding capability is driven by three major parameters:

- the visibility period
- the up-link characteristics
- the on-board command management design & operations.

The major constraint on the commanding capability comes from the very limited visibility window. This requires the Operation Control Centre to prepare a well-defined timeline for a long period, e.g. one week - corresponding to about one hundred passages, operating in the interin of two passages just to analyze the dumped

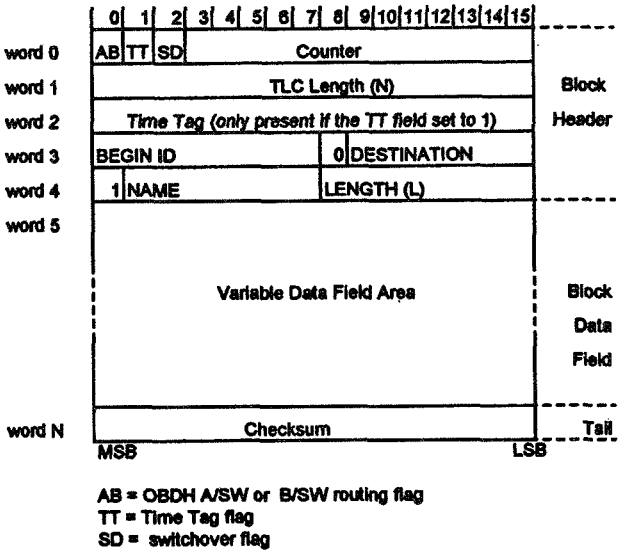


Fig. 5 - Block Command Structure

telemetry and to react to any anomaly detected. All the commands necessary for operating the satellite both in and out of visibility must be up-loaded during the contact period.
The up-link characteristics are based on the ESA PCM Telecommand Standard. It allows the transmission of 2000 bps, that is - as the ESA standard telecommand frame is 96 bits long - a bit more than 20 frames/second. The minimum instruction can be based on a single frame structure. In the case a complex command is needed, a mutiframe message - constituting a block command - can be up-loaded. The block command structure used on SAX is shown in Fig. 5.
Based on the above mentioned standard, the on-board design provides ground with three different options of intervention. Three classes of commands are, in fact, made available and properly managed on board.

- Single frame commands that can be used to up-load high priority command whose purpose is to operate on a critical subset of the satellite hardware devices. This type of commands by-passes any on-board SW control and, via the decoder, directly acts on the end items. This class is thus useful as a back-up in case of an emergency. Typical applications are switching operations involving, for example, unit selection and separation event override command to AOCS.
- Single frame commands that can be used to directly issue single instructions on the OBDH Bus to any terminal. This class might be used only in the case of OBDH B/SW bus management malfunction since they by-pass the OBDH B/SW control. Care might therefore be taken because such asynchronous instructions can affect the proper OBDH Bus protocol functioning.
- Block commands that represent the nominal way of commanding. Their structure can be flexibly filled in so that they can contain either one multi-parameter command or a set of single instructions or one operating plan. Their routing is performed by the OBDH B/SW according to the destination field content. Other syntactic/semantic information is contained in the block header for on-board verification and execution, i.e., begin pattern, destination, name and length.

What is particularly important to emphasize is the on-board capability of managing the block commands as delayed commands. By means of a dedicated flag, ground can, in fact, impose their execution at the time specified in the relevant tag field. A queue of one hundred time tagged commands is dedicated in the OBDH protected memory area - an estimation of about 60 block commands, as a maximum, has been evaluated as necessary each orbit for nominal spacecraft and Scientific Instrument operations. It is worthwhile noting that a dedicated flag is also present in the block structure indicating whether the command has to be deleted in case of CTU switch-over. Since a system reconfiguration takes place at the CTU switch-over, this option is quite useful to avoid any unwanted override unless not explicitly authorized by ground. The mission critical commands, e.g. *Transmitter ON* command, should, anyway, always remain in the queue until their scheduling time elapses.
Within these commanding possibilities ground can address specific requests to any on-board subsystem coordinating the mission operations both in and out of visibility.

One of the major aspects offered by the OBDH A/SW design is the capability of modifying the OBDH A/SW control, devoted to the system operations, by means of simple enabling/disabling commands. As the most important A/SW functions are implemented by a table driven mechanism, a flag has been associated to each of the table entries.

The relevant control can be made active or inhibited by setting the proper value of these flags. An easy updating of the table elements, used as comparison for activating autonomous recovery actions, can be, as well, easily done by mean of dedicated commands.

One of the more powerful features that are made available for emergency ground intervention is specific command to the OBDH operating system. The OBDH SW - in particular the A/SW - is based on a very modular architecture so that each command loop and sequence has been implemented as a stand alone task. Therefore, proper acting on the operating system primitives can modify the task scheduling mechanism.

In particular, the following main interventions can be run-time commanded:

* change the task priority
* init/start/stop tasks
* send/receive messages on mailboxes.

This mainly allows the introduction of a new task implementing new functionalities or replacing the current ones.
The lowest level of possible intervention by ground is the patching of the Intelligent Terminal software. It can be accomplished through the OBDH support which either autonomously executes the patch command on itself, if so addressed, or routes the new data/instructions towards the relevant Intelligent Terminal via the OBDH Bus protocol. The same can be done by directly sending patching commands to the AOCS and the LECS which implement the capabilities of executing the patching by themselves. This avoids putting the microprocessors in wait state until the patch is terminated.

Both the interventions on the operating system and the code have anyway to be planned very carefully with the support of a Software Maintenance Facility whose team shall have a very thorough expertise.
As far as the telemetry commanding capability is concerned, two major features are provided on SAX.
The first one concerns the housekeeping data transmission to ground whose format can be selected between two:

* one essential format corresponding to the produced data set from all the subsystems
* one intensive format that, besides the previous set, includes extra data packets from the hot redundant battery control unit and the B/SW tracing process.

The second is devoted to driving the scientific data collection algorithm. The algorithm, once the scientific activity is enabled, is executed every second, polling the six scientific instruments to get the number of ready scientific packets. The share of the successive scheduled acquisitions between the instruments is based on two ground configurable allocation tables, each of their entries indicating one, out of six, instrument address. Adjustment of the content of the two tables can be done by ground according to each Scientific Instrument data production forecast. Two dedicated commands are available for this purpose.
Last but not least, extra data can be required by ground, dumping both the code and the data segments of each Intelligent Terminal for diagnostic purposes. That in particular allows to obtain some memory areas of the Intelligent Terminals devoted to storing history or trace records not included in the periodic provided telemetry.

OPERATING PLANS

Setting the AOCS and Scientific Instrument configurations and modes usually requires many commands. This can overload both the time tagged command queue and the related scanning process. A solution to this potential problem has been found in grouping a consistent set of commands into only one Operating Plan.
Two types of plans are, in particular, implemented on SAX:

- **AOP** - ***Attitude Operating Plans*** - devoted to commanding the mode transitions of the AOCS and to controlling the attitude manoeuvres within the fine pointing modes
- **POP** - ***Payload Operating Plans*** - devoted to setting-up the instrument configuration and the data output formats for the required scientific performance.

These plans can be up-loaded encapsulated into one command block and then stored in a dedicated Parcking Memory Area. Their activation is requested by ground via the associated *Transfer* and *Enable* Commands, either in real-time or delayed with proper time tags. The actual execution, by the destination terminal, shall follow the correct reception and validation of the incoming Operating Plan only once the *Enable* command is received. Supervision of the whole consistency of this transfer/enabling process is centralized and autonomously made by the OBDH A/SW. It is, in fact, in charge, if enabled, of filtering the *Transfer* and *Enable* commands if not consistent with the satellite mode/configuration, e.g. in the case of Safe Mode fall-back.
As far as the *safe* AOCS modes are concerned no AOP are, anyway, needed since the related attitudes are autonomously acquired and indefinitely kept.

COMMISSIONING CHECK-OUT

The in-flight verification of SAX will be performed in a designated eigth week Commissioning Phase following its launch and separation from the launch vehicle.

The purpose of the Commissioning Phase is to validate the functionality and operability of the satellite and give the go-ahead to the scientific mission. The relevant check-out activity is comprised of two principal sub-phases.

Phase I involves the basic functional/ performance verification of each of the spacecraft subsystems.
Phase II complements Phase I by extending the verification to all the Scientific Instruments and completing the verification of the fully active system configuration.

A summary of the planned activities includes:

- ***Mode Functionality Verification***
  All nominal modes shall be verified for functionality, valid telemetry parameters and expected ranges with respect to the inherent functions of each mode.
- ***Commanded Mode Transitions***
  All nominal mode transitions requiring an uplinked procedure from Ground will be performed and verified. Certain transitions will be omitted for specific reasons, e.g. Delta-V mode transitions.
- ***Autonomous Mode Transitions***
  Verifications of autonomous fall-backs will not be performed as they require fault conditions forced by ground.
- ***Cyclic and Selectable Telemetry Verification***
  All the cyclically generated telemetry will be verified for correct protocol handling, telemetry block structures, parameter location and consistent time and block counter fields. Variable telemetry activated by ground will be verified as well, e.g. dumped data and scientific packets.
- ***On-board Memory Patch and Dump***
  Dump operations will be required to evaluate control parameters not visible in regular telemetry, e.g. AOCS database, history areas, etc. Patches of program or data memories are not a nominal activity but could sometimes be necessary for table item updating, e.g. LECS Instrument. A dump should always be required after a patch operation.
- ***Control Function Calibrations***
  Calibrations or maintenance are required to optimize the overall performance of both the Scientific Instruments and the Subsystems, e.g. thermal control loops thresholds, Instrument digital and analogue discriminator levels, etc..
- ***Redundant Unit Check-out***
  Under nominal operations all operative redundant units will be verified for correct functionalities, e.g. gyros, decoders, receivers, etc.. Cold redundant units will not be activated or verified unless necessary because of failures. It is considered more prudent to maintain a good nominal configuration rather than risk possible failure in activating the redundant one.

CONCLUSIONS

The SAX satellite is the result of a quite challenging mission requirement implementation.
Once in orbit it will support the extensive activity of six complex Scientific Instruments performing parallel X-ray observations.
The system design is based on a distributed intelligent architecture allocating to each of the on-board computers its own specific function. This has been designed to provide the maximum flexibility and reliability in autonomously executing the ground mission plans. The SAX implementation of the operating modes, in fact, allows the on-board configuration to be maintained by itself, supporting, at the same time, the ground required operations.

To conclude, the SAX mission will not only provide the most up-to-date results in the field of X-ray astrophysics, but it will also make operative a very powerful system that is the product of Italian scientific satellite engineering.

REFERENCES

1. Finocchiaro G., Santoro P., Attinà P. - *SAX Satellite Design and Verification,* XII AIDAA National Congress, Como, Italy, July 1993.

2. Butler C., Scarsi L. - *The SAX Mission,* Adv. Space Res., 8, 265-279, 1991.

3. *SAX Design Report,* Alenia Document SX-RT-AI-118.

4. *SAX User's Manual,* Alenia Document SX-MA-AI-019.

5. Martelli A., Fowler M., Ciampolini V. - *On-board Autonomy Concepts and Implementation on the SAX Low Earth Orbiting Scientific Satellite,* IFAC Workshop on Spacecraft Automation and On-board Autonomous Mission Control, Darmstadt, Germany, September 1992.

N95-17552

11137

354204
P-9

Small Satellite Space Operations

Keith Reiss, Ph.D.

CTA Space Systems
1521 Westbranch Drive
McLean, VA 22102

ABSTRACT

CTA Space Systems (formerly DSI) has played a premier role in the development of the "lightsat" programs of the 80's and 90's. The high costs and development times associated with conventional LEO satellite design, fabrication, launch, and operations continue to motivate the development of new methodologies, techniques, and generally low cost and less stringently regulated satellites. These spacecraft employ low power "lightsat" communications (v.s. TDRSS for NASA's LEOs), typically fly missions with payload/experiment suites that can succeed, for example, without heavily redundant backup systems and large infrastructures of personnel and ground support systems. Such small yet adaptable satellites are also typified by their very short contract-to-launch times (often one to two years). This paper reflects several of the methodologies and perspectives of our successful involvement in these innovative programs and suggests how they might relieve NASA's mounting pressures to reduce the cost of both the spacecraft and their companion mission operations. It focuses on the use of adaptable, sufficiently powerful yet inexpensive PC-based ground systems for wide ranging user terminal (UT) applications and master control facilities for mission operations. These systems proved themselves in successfully controlling more than two dozen USAF, USN, and ARPA satellites at CTA/SS. UT versions have linked with both GEO and LEO satellites and functioned autonomously in relay roles often in remote parts of the world. LEO applications particularly illustrate the efficacy of these concepts since a user can easily mount a lightweight antenna, usually an omni or helix with light duty rotors and PC-based drivers. A few feet of coax connected to a small transceiver module (the size of a small PC) and a serial line to an associated PC establishes a communications link and together with the PC constitute a viable ground station. Applications included geo-magnetic mapping; space borne solid state recorder validation; global store-and-forward data communications for both scientific and military purposes such as Desert Storm; UHF transponder services for both digital data and voice using a constellation; remote sensor monitoring of weather and oceanographic conditions; classified payloads; UHF spectrum surveillance, and more. Ground processing has been accomplished by automatic unattended or manual operation. Management of multiple assets highlights the relative ease with which 2 constellations totaling 9 satellites were controlled from one system including constellation station keeping. Our experience in small end-to-end systems including concurrent design, development, and testing of the flight and operational ground systems offers low cost approaches to NASA scientific satellite operations of the 1990's.

BACKGROUND

As Congressional budgets tighten and conventional military threats appear to dissipate, private industrial R&D, universities and other potential participants in primarily LEO missions are increasingly drawn to consider new options. While

STS flight availability and piggyback experiments flown on larger missions are still possibilities, the resurgence of small satellites as viable experiment platforms is a distinct part of the general solution. This is especially so for new commercial applications and the exercise of new technologies in the space environment where time from design to launch is of the utmost importance. Five years is not the answer while two years, or less, can meet competitive and marketing needs. On the other hand, science and technology innovations are difficult to fund on their own, but can often fit nicely into multi-mission oriented lightsats.

Costs of experiments borne by "lightsats" can dip considerably below many other options, though the lightsats may not offer the same degree of reliability as their larger and costlier counterparts. Lightsats are often deployed in clusters to diminish the relative launch costs. Complementing the reductions in space segment cost, the ground segment can usually support most missions at a fraction of the expense imposed by current standards. In the past ten years CTA/SS has produced a large number of "lightsat" system designs utilizing compressed schedules for development and test and very low key mission operations. The evolution towards more automated bus, experiment and ground operations and less cumbersome spacecraft command and control is leading towards provision of stable mission operations without the customary large levels of ground support. Additionally, inexpensive space-to-end user terminals have been developed. Such services can provide direct experiment to laboratory connectivity which is of great interest in university science and engineering applications as well as commercial or government circles.

SIMPLIFICATIONS

Small satellites with small budgets for operations must still satisfy broad requirements:

- Provision for bus control via ground commanding
- Provision for experiment/payload control
- Provision for onboard telemetry collection of both bus and experiment/payload systems
- Provision for on-board autonomous health and stability protection
- Provision for TT&C data flows and experiment/payload data flows

In the most common instance, mission operations are performed from a central location where the state of health (SOH) of the entire spacecraft is continuously assessed. It is generally here also that flight commands are issued to the spacecraft. In CTA Space Systems' history, we have built and operated the first GLOMR satellite in 1985 from a PC but without any automation of communications. Telemetries (TTMs) were received and commands sent aloft from an inexpensive adjunct transceiver module under micro-control and employing a simple roof-mounted UHF omni-directional antenna. Command streams were short and TTMs limited in this spacecraft, but for those that followed, there were many improvements and adaptations stemming from a growing assortment of mission requirements. It is important to embody certain "simplifications" into the fabric of the overall system design in order to facilitate low cost, yet reliable, small satellite operations.

We seek to accomplish certain key objectives:

1. Operate experiments from pre-established command sequence files
2. Provide pre-uplink command verification
3. Employ macro style bus and experiment commands
4. Provide spacecraft scheduled (i.e., for future) as well as immediate command execution options
5. Provide reliable (error free) and autonomous communications
6. Provide "intelligent" SOH displays/reports
7. Provide key mission operations software elements as part of the EGSE (avoiding full

probably separate efforts) and use throughout I&T (Integration and Test), the IST (Integrated Systems Test) and Environmental Testing

8. Offer autonomy in routine communications scheduling
9. *Wherever possible encourage provision of experiment autonomy with independence of other experiments and bus*
10. *Wherever possible use autonomous bus sub-systems (notably ADACS) requiring minimal ground attention*

Item (1) is tried and true through such programs as STACKSAT (three satellites: TEX, POGS & SCE); SCSC (two satellites known as "MACSATS" and seven "MICROSATS); REX; and RADCAL. The savings and reliability associated with the construction and pre-validation of operational sequences which make up mission operations segments are very significant. For example, a series of commands required to operate a diffusion pump and to trigger a particular set of experiment actions is accomplished by writing a series of commands under software control. Each individual command is range-checked and otherwise evaluated to be a valid command (as noted in item 2) and is encapsulated in a 16-bit check sum (CRC) to assure future integrity. The set of commands is saved as a file and can be evoked during all phases of ground-based testing as a block with individual command execution times shifted by a definable increment avoiding having to make up sets with specific pre-set execution times. The same segment can be conveniently recalled and sent to the spacecraft when on-orbit. The very significant work force necessary to conduct around the clock environmental and integrated systems tests is greatly reduced by *avoiding* the effective hand entry of large numbers of detailed commands. Errors are nearly eliminated in the process. Item (3) is a significant objective in that it suggests that wherever feasible, the ground to space interface is held to as simple a structure as possible. That is, the spacecraft bus or experiment/payload commands should be process-oriented if possible. For example, in the case of the preparation of an instrument application, there may be a 25 step timed sequence of "micro style" commands required. If the controller for that experiment or the bus processor can maintain that sequence as part of its operational flight code, then all the ground team needs to do is to evoke that process by a simple command such as "Experiment 2, Process 5, ON=2/23/95 13:00:00." Similarly the shutdown might be commanded "Experiment 2, Process 6, ON=2/23/95 13:45:00." These two commands are easy to deal with and will achieve the highest level of reliability. If this is not possible, then the command sequence file approach can be used instead with the operator simply evoking the two procedures adjusting the process execution times according to the plan. The disadvantage here is that there are now many commands to uplink and it is essential that they are *all* accounted for on the spacecraft prior to beginning the execution of the procedure. Verification of the presence of the entire *command chain* for a process in the past has usually been accomplished by a satellite schedule dump and on the ground review. The operator then had the option to re transmit missing commands or to delete commands. A better method involves the addition of a special command type that will inhibit execution of incomplete command streams. This command spawns a notification message to the ground that its powers have been evoked and that the sequence is either incomplete or OK. With present and emerging powerful and robust flight digital electronics including wide usage of EDAC RAM or other (nearly) non-volatile mass memory, storage of command chains onboard that can be evoked by an immediate or future-acting ground uplink command are more prevalent.

Normally commands are sent to the spacecraft in advance of planned execution and are executed at future times under the action of the spacecraft's software scheduler. Immediate commands (with zero tokens for execution date/time) are, however, allowed to execute immediately. Given that the uplink commands and downlink TTMs/data are

reliably communicated, there is little in the area of routine flight operations that necessitates constant operator attention much less "crisis-like" circumstances on the ground. Indeed, with easily interpretable and "to the point" SOH displays available on the ground, the missions are virtually made to "fly themselves" for considerable periods of time.

Item (7) represents an important ingredient to planning and executing a successful low cost small satellite mission. It is a standard practice at CTA/SS to develop the TTM and command formats and specifications early in the systems design stage and to build around them the essentials of ground station processing and communications software. These elements are assembled into the PC-based EGSE suit that accompanies the satellite from the I&T test bed, throughout I&T and environmental testing and beyond. These *same* elements which have accumulated much equivalent "flight time" and have been perfected in a natural manner are then incorporated into master or remote ground station packages. There is no separate team associated with the ground station operational software; it is basically an inherited evolute of the spacecraft development process.

Point (8) suggests an innovation that is currently underway in three CTA/SS programs. Unlike older systems that require scheduling and pre-programming of satellite communications events, some new programs are now operating via intelligent space and ground systems to totally avoid routine contact scheduling which is a tedious process. The UTs, for example, maintain their own ephemerides and simply come on the air when the satellite is known to be visible at some preset minimum elevation angle and/or in an allowed azimuth band. In some programs the satellite itself autonomously contacts ground units without cumbersome deterministic scheduling uplinked previously by the master controller. Not only is this much more efficient, it also allows dynamic response disallowed by too much pre-planning.

The final two points refer to spacecraft subsystem and experiment levels of autonomy and non-mutual interference. A very great deal of labor and engineering efforts are expended throughout testing and later mission operations; labor that escalates sharply when systems conflict in any way or when excessive and too-frequent monitoring and control is required. This placing the operators "in-the-loop" in the manner of a very stiff control law. Building inexpensive space systems for small satellites that do not impose these penalties may be a challenge, but should always be entertained in both the bus and experiment arenas.

SMALL SYSTEMS, SMALL OPERATIONS APPROACH

A Case Study: POGS

The typical lightsat ground configuration consists of a frequency agile UHF transceiver with mod/demod capabilities in various forms of BPSK and FSK modulations operating under a simple micro controller. This unit can be rack mounted inside, or configured in an environmental housing for exterior deployment. Coax connects the RF to a RHC or LHC circular polarized omni-directional radome-covered antenna or complementary pair. From the transceiver unit there is a standard serial (RS-232) line interfacing an ordinary PC. This comprises the minimal standard configuration. A variant is the replacement of the omni-directional antenna is a light weight directional antenna which may be either linearly or circularly polarized, but in either case can be driven by inexpensive light duty commercial rotors. Such directive antennas are driven by open-loop controllers connected via PC cards implants or via an additional PC RS-232 serial ports. A component of the ground station software generates antenna tracking data for each

pass and provides it to the antenna controller. While there is nominally only a few dB's of gain advantage, it is often useful in noisy metropolitan areas and the same system can also lead a high gain X-band antenna, for example, to acquire the satellite and allow it to switch to closed-loop tracking for more precise alignment.

A typical mission illustrates the major points in CTA/SSs small systems approach. The USAF STACKSAT mission deployed three small satellites into nominal 300 nm polar orbits. POGS (Polar Orbiting Geomagnetic Satellite) was dedicated to the primary NORDA mission of magnetic mapping of the Earth's field and was equipped with a 6 foot boom-mounted NASA magnetometer instrument and a 4 Mbit CMOS dual channel SSR (solid state recorder). To prepare the operational staff of a one or two individuals from Bay St. Louis, a two week long training program was conducted in McLean. Shortly thereafter, and while the future operators looked on, CTA/SS conducted the initial on-orbit testing procedures which commenced with the deployment of the spacecraft's gravity-gradient boom equipped with hysteresis rods to quench spin and libration. With the boom and antennas deployed, the satellite was ready for checkout and the entire procedure was handled from a PC system with omni antenna from the rooftop of our building just outside the Washington Beltway at Tyson's Corner Virginia. Despite high local noise levels various sources including one nearby arc-welder, everything went smoothly and the spacecraft was soon ready for hand-off. Operations had consisted first of a mission plan previously approved by the USAF and Aerospace. The plan permitted sufficient latitude thus avoiding serious delays while necessary variations would have been proposed and officially accepted. This is important to the low profile operation that we designed and budgeted. Finally the station at the user site at Bay St. Louis took over the operation that included two key unmanned high latitude receiving sites. Data down linked from POGS was automatically diverted to a WORM optical disk drive occasionally removed and mailed to the Bay. St. Louis facility although data was frequently recovered remotely via a dial-up link. Software updates and parameterization changes were facilitated via the remote dial-up link which also allowed operators at either Bay St. Louis or CTA/SS to "man" the remote station. This also allowed one to see all the displays and to operate the keyboard remotely as though present on-site. POGS provided its requisite magnetic data in a few months and is still operating after about four and a half years. POGS also has a number of communications capabilities and other sensors all of which have performed flawlessly and have been operated with the most minimal of ground support.

Communications scheduling for all ground elements and the spacecraft is handled over intervals of generally ten days time by the Bay St. Louis PC from which the appropriate files are simply "modemed" into the remote Arctic sites. The uplink to the satellite of command files can be accomplished from any of the three stations. Multiple sites provide excellent redundancy although the avoidance of non-standard computers, other equipment and software always provides inexpensive and obtainable components which need not be duplicated as spares. This approach permits a natural flow of technology improvements to the ground systems. It hinges on the use *wherever possible* of commercial software and hardware products and the use *where possible* of standard interfaces. This is not generally true of government systems.

Highlights of Cost Reducing Factors

- Relaxed Official Coordination/Documentation Requirements
- Technical Backup Availability Including On-Line Operational Support
- Spacecraft Supports Long Term Scheduling to Allow Autonomous Operation of *Both* Bus and Experiments for Days or Weeks on End

- Semi- or Fully-Automated Ground and Spacecraft Communications Scheduling Software
- Compact and Powerful Spacecraft Commands
- Telemetries and Telemetry Displays Keyed to Early Warnings via Color Coding at the Top Level
- Use of COTS (Consumer Off The Shelf) Products:
  - ◊ Generic IBM-like PCs and Peripherals
  - ◊ Land-Based Communications
  - ◊ Standard File Transfer (e.g., KERMIT)
  - ◊ Operating Systems (space and ground)
  - ◊ Use of Standard Protocols (HDLC[space-ground], TCP/IP[Internet], etc.)
- Planned Software Reusability
- Training and Simple SOPs (Standard Operating Procedures)

FUTURE IMPROVEMENTS

As small satellites (nominally of Class C construction) improve and advance with continuing miniaturization/weight reduction and other technical innovations, inclusion of new technologies and science applications are bound to create enhanced demands. Obtaining high cost-to-effectiveness for many future missions will depend on successes in modifying conventional approaches to today's large scale expensive launches and flight operations. These changes may be perceived as somewhat radical today, and yet to a large extent, they represent a rebirth of older principles of pioneering space developments that, over the years, have become somewhat anachronistic. The procurement process for DoD and NASA and associated regulatory demands are simply not structured to foster the rapid development of small satellite missions (including inexpensive LVs). To an extent they may also appear to conflict with STS mission elements since many tasks have and continue to be executed by manned crews- tasks that for a fraction could be carried out *not for days but for years* by small satellites linked to inexpensive ground systems and targeted to the needs of the experimenters/laboratories. Conventional approaches to flight operations are grand by comparison to the probable minimal needs of many potential candidate packages. To regain the spacecraft "pioneering" spirit of the '60s using today's small powerful computers both onboard and on the ground together with spectrum of technology improvements in both materials, components, structures, and manufacturing processes we can achieve magnitudes more results for the same relative costs.

A major and bold new NASA initiative is fully targeted towards achieving the goals and objectives typified by the small satellite mission under discussion in this article. The **Small Spacecraft Technology Initiative (SSTI)**, dubbed "pathfinder" by the program sponsor, will produce two spacecraft "LEWIS" and "CLARK" with the latter being built by CTA/SS with Martin Marietta utilizing a set of IPDTs (Integrated Product Development Teams) including commercial entities, universities, NASA research centers and others involved with technology and science infusion/assessment and in fostering US commercialization efforts. CLARK is a fast track 24 month-to-launch program lofting a 3-meter optical imaging payload, a variant of the successful MAPS instrument ("μMAPS"), an X-Ray Spectrometer, an Atmospheric Tomography Retro-reflector while also incorporating 36 explicit advanced technologies for space testing. Major bus elements including the 32-bit RHC3000 processor and SOA ADACS components offer unique opportunities to combine otherwise independent activities to provide enhancements in both science results and in operational efficiencies. The use of the μMAPS to detect clouds and prevent down link of useless images is but one example. NASA has adopted the entire tenant of the small satellite mission- form initial design and development, through launch and initial orbit, and throughout the flight. All of the concepts advanced in this paper are included in the CLARK plan which will allow the enhancement previously tested equipment,

software and operational methodologies in an expanded context enveloping the disbursement of larger volumes of experiment data and in the promulgation of other mission information utilizing more open Internet accessways to facilitate wide community participation in this interesting endeavor. The low cost of the entire mission makes necessary the reforms cited and includes the active participation of the NASA sponsor as an IPDT member and not as an outside force passing judgment based on periodic reviews. SSTI significantly *is* an Initiative and will add impetus to future small satellite programs. In this sense the moniker "Pathfinder" seems most appropriate.

Systems Development

1. Architectural Approaches

* *Presented in Poster Session*

PRECEDING PAGE BLANK NOT FILMED

111138 N95-17553

354206 p-8

EMBEDDED PARALLEL PROCESSING BASED GROUND CONTROL SYSTEMS FOR SMALL SATELLITE TELEMETRY

Michael L. Forman
NASA, GSFC, Code 513
Greenbelt, MD 20771
&
Tushar K. Hazra
Gregory M. Troendly
William G. Nickum
Martin Marietta Services, Inc.
GSFC, Greenbelt, MD 20771

ABSTRACT

The use of networked terminals which utilize embedded processing techniques results in totally integrated, flexible, high speed, reliable, and scalable systems suitable for telemetry and data processing applications such as Mission Operations Centers (MOC). Synergies of these terminals, coupled with the capability of terminal to receive incoming data, allow the viewing of any defined display by any terminal from the start of data acquisition. There is no single point of failure (other than with network input) such as exists with configurations where all input data goes through a single front end processor and then to a serial string of workstations.

Missions dedicated to NASA's Ozone measurement program utilize the methodologies which are discussed, and result in a multi-mission configuration of low cost, scalable hardware and software which can be run by one flight operations team with low risk.

KEYWORDS

Embedded parallel processing, Ground systems, Transputers, and Total mission concept.

1. INTRODUCTION

At the SPACEOPS '92 conference [5], it was shown that PC's could be used to control spacecraft and were capable of high throughput and performance if embedded processor methods were used [1]. Control centers using embedded serial processors were implemented for Nimbus 7 (N7) and the Meteor3/ TOMS (M3/T) missions, and have operated flawlessly since their inception in 1987 and 1991 respectively.

In 1991, development of embedded systems using parallel processing components based on transputer technology was begun. In 1992 we were tasked to develop a totally integrated control center using one Flight Operations Team (FOT) to operate N7, M3/T, and the Total Ozone Mapping Spectrometer - Earth Probe (TOMS-EP) missions. This facility is the TOMS Mission Operations Center (TMOC), and is leading the trend of combining similar missions with similar systems into multi-mission, single FOT facilities.

The trend in modern space mission control systems is moving towards standardizing telemetry systems design [9] as is evidenced by the move towards the adoption of CCSDS standards. These systems make use of the rapid advances in workstation or PC

technology and contribute towards making the multi-mission Mission Operations Center (MOC) a reality. This paper will discuss the TMOC configuration utilizing embedded processing systems suitable for TOMS and other missions as shown in figure 1.

2. EMBEDDED PARALLEL PROCESSING : SCOPE AND BENEFITS

For the most part, telemetry processing is a bit oriented, repetitive, series of operations which are usually implemented in a serial process. Throughput gains are attained either through hardware implementation of repetitive software processes or the use of higher speed processors such as the DEC Alpha chip. Most telemetry processing for today's spacecraft can be handled in a serial manner, especially if we confine our functions to engineering or command matters.

The rapid advance of very large scale integration (VLSI) technology, and the

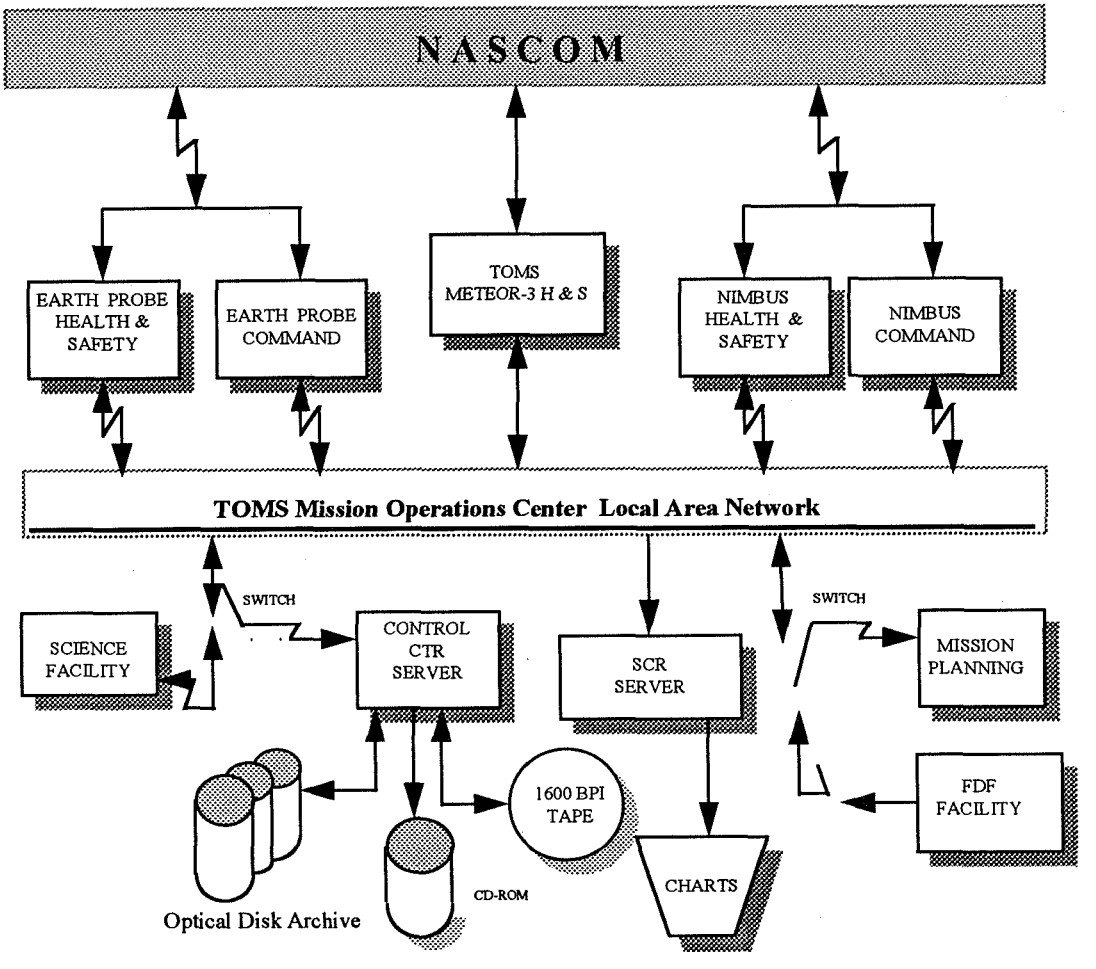


Figure 1: TOMS Mission Operations Center (TMOC) Configuration
Three missions are shown to interface to the NASCOM interface which provides inputs to the TMOC. All mission terminals and redundant ones (shaded) are connected through an internal LAN. All servers are switchable from the internal LAN to the NASA ethernet for security purposes.

availability of low cost processors have made it feasible to develop high performance, cost effective, and efficient parallel computer systems utilizing more than one processor, while maintaining a software design for implementation. These parallel methods lend themselves to bit and computationally intensive operations such as telemetry analysis, orbit and attitude analysis, and science processing and result in systems which are scalable, low cost, high performance, flexible, and reusable.

These systems [4,5,6,7] are based on the use of transputers as the parallel processing device. Transputers feature a built-in hardware scheduler which permits more than one concurrent process to share the processors time, and four DMA links to provide highly efficient inter-processor communication and data transfer. Hence, if the computation to communication ratio of component processes is considerably high, and the task allocation is uniform, multiple processes can be executed efficiently in parallel fashion. This strategy can be extended in dealing with future requirements by adding extra processing modules. In other words, embedded parallel processing offers scalability and flexibility to the system.

For our telemetry and command applications, a large body of software has been developed which executes on the embedded processor, requiring no significant resources from the host operating system, with a shared memory capability. This sets the basis for doing bit operations on the embedded processor while the host serves as the man/ machine graphical interface. More details on the scope of uniting of PC's and transputers as embedded processors can be obtained from references [2-7]. The benefits of utilizing the embedded parallel processing technology are:

a) Flexibility - Systems can capture all downlinked data, and immediately begin initial processing or data distribution. Systems are small, truly transportable, and require only normal office surroundings with clock and signal as inputs. Standard and non standard telemetry inputs can be processed simultaneously while commands are being output in required formats and rates.

b) Data Throughput - Base throughput rates depend on the number and architecture of the components as well as the parallel programming design. Rates in excess of 10 Mbits are achievable on our TOMS-EP system with NASCOM deblock only. Other levels of decommutation utilizing software algorithms slow the effective real time rate down to about 50 Kbits sustained for full health and safety while simultaneously archiving input data at the 10 Mbit receive rate.

c) Efficiency - System modularity, reusability, and ease of implementation lead to low costs, rapid implementation, and high performance.

d) Scalability - Systems can be built or expanded according to the demand of jobs or tasks to be performed and the systems can be reused in whole, in part, or with additional processing modules.

In addition, the use of embedded parallel processing and transputer technology contributes directly toward enhancing the unique features of the total mission concept. Based upon the granularity of parallelism exploited in the design, the system can be expanded to achieve the flexibility, reliability, and performance desired in the total mission system. The Total mission concept will be discussed in section 4.

3. CURRENT CAPABILITIES OF THE SYSTEM

The system architecture utilized in the TOMS Mission Operations Center (TMOC) is based on commercial off the shelf (COTS) products. Low cost, reliable, upgradable, user friendly,

multifunction, standardized hardware and software are just a few of the goals in the system design. The TMOC is entirely driven by Personal Computers (PC's) utilizing the Intel Processor family. Embedded parallel processing is added to critical systems where real-time processing and/ or high computational requirements may be needed, and hence eliminating the need for high cost workstations and related software, as well as separate, costly front end processors (FEP). The ability to selectively add special purpose parallel processing modules gives the total system great flexibility. At a relatively low cost, the system can be reconfigured to support many of the current and proposed NASA missions.

The major functional areas are shown in figure 2, which include Real-time Command and Control, Health and Safety, and Mission Planning. The individual control center systems are connected via a standard ethernet Local Area Network (LAN). This makes it possible to transfer data between major system functional areas, as well as between individual

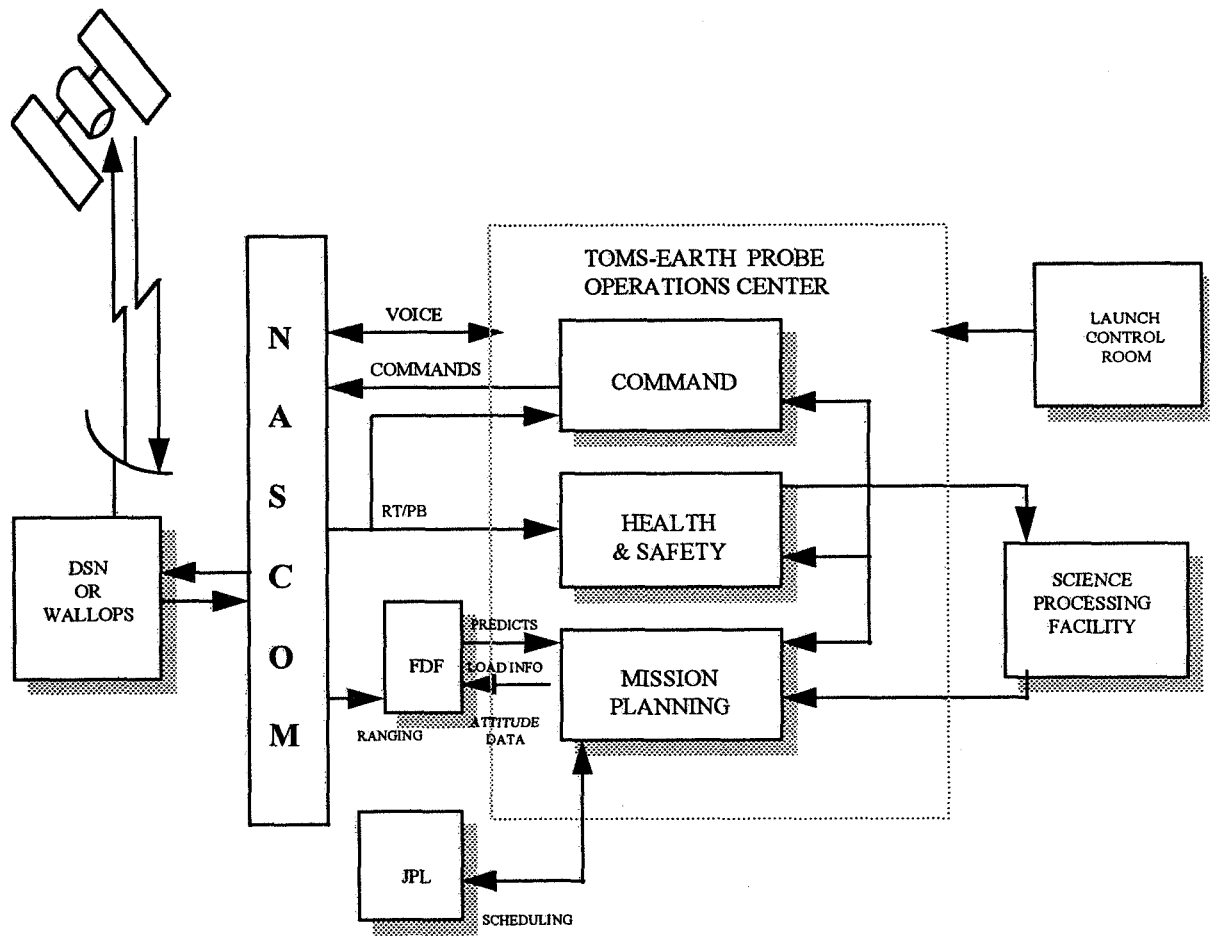


Figure 2 : An overview of the TOMS-EP mission system. The operations center exhibits its major operations -command,health and safety, and mission planning. The center interacts with the DSN or Wallops station via NASA communication network (NASCOM), and also with the launch control room, Flight dynamics facility (FDF), Jet propulsion laboratory (JPL) and Science processing facility during its telemetry operations.

systems. The TMOC interfaces with external components by several communication paths such as:

1. The Deep Space Networks (DSN) and the Wallops flight facility are utilized for support of command, telemetry, and tracking for the TOMS satellites. The NASA communications (NASCOM) network provides the interface between the DSN sites and the TMOC at the GSFC facility. The embedded parallel processor incorporates the FEP internally, making the single PC based workstation fully portable; the internal FEP is fully programmable for many packet formats received from TDRSS, DSN, MDM, IOS, or raw data from a bit synchronizer.

2. The Flight Dynamics Facility (FDF) provides the attitude determination and verification, as well as orbit determination support. The FDF products are transferred directly to the mission planning systems for incorporation into on-line databases. A standard ethernet network utilizing TCP/IP provides the interface to the mission planning systems.

3. The Jet Propulsion Laboratory (JPL) and mission planning coordinate and schedule all support for different components of the mission. Dialup/ dialback modems are utilized within the mission planning systems for the JPL interface. With the FDF data, JPL schedules, and instrumenter's command requests, command loads are prepared from the databases and transferred directly to the on-line Command systems.

4. The Science processing facility receives Level-0 processed TOMS data and further processes the data to create various products. The science facility receives data on a daily basis via standard ethernet using TCP/IP. Long term trending data is archived in the control center on CD-ROM media and furnished to the Science facility on an as required basis.

The on-line systems that are connected to the NASCOM network are all identically configured systems as shown in figure 1. For the TOMS-EP there are four systems on-line: a Primary Command system, a Primary Health and Safety system, a backup Command system, and a backup Health and safety system. Each on-line system has a UNIX Operating System (OS) with an X-windows based Graphical Users Interface (GUI) supporting the Motif X11R5 standard. All of the Command, Health and Safety, Level-0 processes, and analysis applications are written in C language using the Motif library. This standardized approach enhances the portability of the application source code to other platforms, as it may be necessary in future. Using UNIX and Motif also allows the system to incorporate NASA products such as Satellite Telemetry Operating Language (STOL) into the Command system. Since all of the on-line systems are identical, any one may execute the Command software or the Health and Safety software.

Currently, the TOMS-EP command system utilizes a command database specifically tailored for the EP satellite and TOMS instrument. The command system not only has Real-time command capability, but also full storage and forward capability. These features allow for frequent use of stored sequences of commands, and preprogrammed matrices that will be executed onboard the spacecraft at predetermined times. All commands transmitted are verified by echo blocks from the DSN site and further verified by the telemetry downlink. The telemetry downlink is in a CCSDS format and is fully decommutated in real-time in the internal FEP.

The TOMS-EP Health and Safety system provides a full analysis of both the

spacecraft and the instrument in real-time. The screen format is set up as four quadrants, each quadrant may be customized by a satellite subsystem. In addition to the four quadrants, a general status panel is always visible at the top of the screen for a running summary of pass statistics. From a pull down menu bar and also hot keys, multiple panels are available for display by pressing of a button or clicking a mouse. Every telemetry point is in a database driven lookup table that is being updated in real-time through a shared memory interface. The embedded FEP does all the decommutation of the telemetry which places the processed data into the shared memory interface. Within the database, several things are occurring on each entry point such as calibration, floating point conversion, mode, event and alarm determination. The entry is then displayed based on a user defined display format. A row of subsystem buttons continually show the current status of each subsystem by changing the color. Green implies a normal operation, Yellow and Red indicate potential problems. By moving the mouse cursor on the subsystem button and clicking the mouse button, the event and telemetry panels associated with that subsystem are immediately displayed for analysis. In addition, there are X-Y plots that may also be configured in the display panels.

Along with the primary Health and Safety UNIX based systems, there are several standard PC's without FEP's. These PC's are configured as standard office systems running MS-DOS OS and Windows and are connected to the LAN. With an X-windows package, they are capable of running remote Health and Safety sessions in a client/ server configuration. The UNIX system becomes the server and executes the prime Health and Safety program. The DOS PC logs into the server as a client and executes a slave version of the same program. This configuration allows multiple screens of several subsystems to be viewed simultaneously. The DOS PC is used to transfer telemetry data points from the UNIX system and run trend analysis tools such as Quattro-Pro, Lotus, Excel or other packages that a user is familiar with. A note needs to be made at this point, with this type of PC environment, a relatively low cost control center can be put into full operation.

A localized LAN is being utilized for the control center communications. The bandwidth of the LAN can support the slave DOS client systems, mission planning, and an Astromed stripchart subsystem. The Astromed stripchart subsystem consists of four Astromed MT95000, 16 channel digital strip chart recorders. The four recorders are controlled by a front-end PC connected to the same LAN. The prime Health and Safety system sends raw telemetry directly to the 64 channel subsystem. The telemetry points, recorder speed and all controls are setup through a pop-up X-window during the pass. Any page on any terminal can be "popped" up on any other terminal on the network.

4. THE TOTAL MISSION CONCEPT

The TMOC control center architecture is designed for its missions and is self contained but can be expanded to include flight dynamics and science processing within the control center. The system is very modular allowing dynamic reconfiguration 'on the fly'. Figure 3 represents a Total Mission Concept that may be implemented within the TMOC requiring very little external support by adding the following functions:

1. Flight Dynamics

a. Integration of some of the Flight Dynamics functions directly into the control center. Several off the shelf products are now available from commercial companies such as

STORM Technologies and Integral Systems Corporation that make this feature doable now for attitude computation and mission planning products. It is assumed that precision 2-3 line elements are available.

2. Science Processing

a. The Level-0 product is already processed within the control center. The system can easily be enhanced by scaling up the system compute power by adding parallel processing nodes to provide Levels 1, 2, and 3 processing. These products could then be distributed to users and archived.

b. The addition of image quick look capability to verify data quality during real-time and playback data recorder dumps.

The Total Mission Concept for a control center can be implemented today in a very cost effective scenario. The same operations personnel could perform all the functions listed above from mission planning, through acquisition, analysis, data archiving, and the creation and distribution of science products. Multiple missions may be controlled by the same equipment and operations personnel by just selecting the mission type on

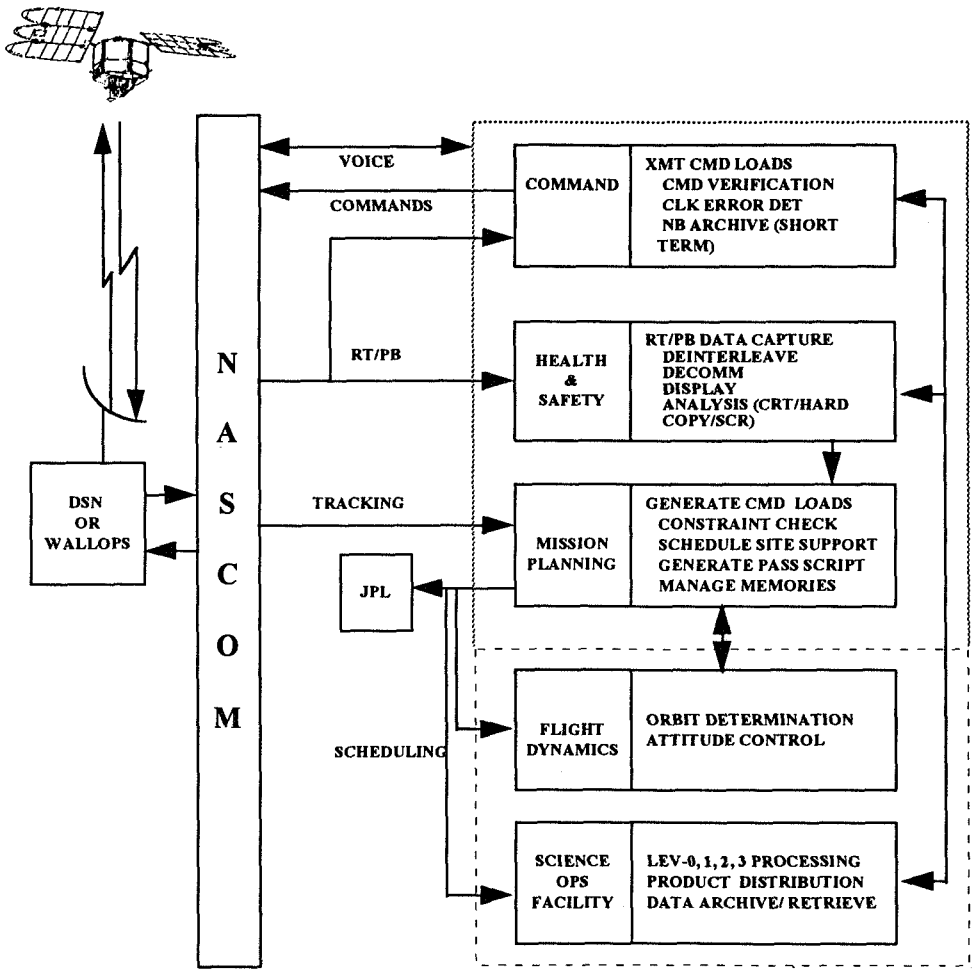


Figure 3: The Total Mission Concept implementation on the TMOC architecture. The dotted box exhibits the extension that can be added to achieve the flight dynamics and science processing capabilities to the system to perform orbit determination and level 1, 2, 3 processing.

screen. The nonreal-time DOS/ Windows systems are utilized in a multi-purpose mode from daily office operations to a client/ server based evaluation tool. All of this leads to efficient utilization of facilities, equipment, personnel and bottom line mission cost.

5. SUMMARY

The basis of using Transputers and Alpha chips in an embedded processing environment was to support the expansion of the Ground System from a simple command and telemetry analysis system to a system that supports spacecraft I&T, command & telemetry, and science processing and distribution. The cost effectiveness of this Total Mission concept and the ability to support multiple satellites simultaneously provides for a smaller operations staff resulting in an overall lower life cycle cost. In today's environment, this is a definite benefit when planning new missions.

Acknowledgments

The work described in this paper involves several engineers of the Martin Marietta Services team headed by Richard A. Stephenson. The project is sponsored by NASA under the contract no. NAS 5-31739 at Goddard Space Flight Center, Greenbelt, MD. The Authors gratefully acknowledge the support of NASA Goddard Space Flight Center, and the highly efficient design efforts exerted by the motivated team of B.J.Gonciarz, Dr.S.S.Chen, B.Singh, S.A.Burns, A.M.Larson, J.T.Fate,Jr. and others in system development and operations team.

Bibliography

[1] Austerlitz, H., (1991), Data Acquisition Using Personal Computers, Academic Press.

[2] Cook, R., (1991,June), Embedded Systems in Control, BYTE, (pp.153-158).

[3] Ellis, G.K., (1989,Oct.), Data Acquisition and Control Using Transputers, Proceedings of The Second Conference of The North American Transputer Users Group, (pp.61-76).

[4] Forman, M.L., Hazra, T.K., Troendly, G.M., Nickum, W.G., (1993,Oct.), Applying PC-based Embedded Processing for Real Time Satellite Data Acquisition and Control, Proceedings of The Twenty Ninth Annual International Telemetering Conference, Las Vegas, NV, (pp.165-173).

[5] Forman, M.L., Troendly, G.M., Nickum, W.G., (1992,Nov.), High Performance Low Cost, Self-contained, Multipurpose PC-based Ground Systems, Proceedings of The Second International Symposium on Ground Data Systems for Space Mission Operations, (pp.733-737).

[6] Hazra, T.K., Troendly, G.M., (1994,May), Designing the Earth Probe Control Center for Telemetry Processing Utilizing Embedded Parallel Systems, Proceedings of The European Telemetry Conference, (pp.287-297).

[7] Hazra, T.K., Stephenson, R.A., Troendly, G.M, (1994,Oct.), The Evolution of the cost effective, high performance ground systems: A Quantitative Approach, To Appear in The Proceedings of The Thirtieth International Telemetering Conference.

[8] Muratore, J.F., et al, (1990,Dec.), Real Time Data Acquisition at Mission Control, Communications of The ACM, Vol.33, No.12, (pp.18-31).

[9] Sielski, H.M., et al, (1991), Modern Space Telemetry Systems, ITEA Journal, Vol. XII, No. 4, (pp.27-33).

[10] Sloggett, D.R., (1989), Satellite Data: Processing, Archiving, and Dissemination, Vol. I & II, Ellis Horwood.

111139

N95-17554

354209

P-13

Open Solutions to Distributed Control in Ground Tracking Stations

by
Wm. Randy Heuser
Member of the Technical Staff
Jet Propulsion Laboratory
California Institute of Technology
Mail Stop 301-235
4800 Oak Grove Drive
Pasadena, California 91109
Office phone (818) 354-0956, Fax (818) 354-9068
email: rheuser@binky.jpl.nasa.gov

Abstract

The advent of high speed local area networks has made it possible to interconnect small, powerful computers to function together as a single large computer. Today, distributed computer systems are the new paradigm for large scale computing systems. However, the communications provided by the local area network is only one part of the solution. The services and protocols used by the application programs to communicate across the network are as indispensable as the local area network. And the selection of services and protocols that do not match the system requirements will limit the capabilities, performance and expansion of the system. Proprietary solutions are available but are usually limited to a select set of equipment. However, there are two solutions based on "open" standards. The question that must be answered is "which one is the best one for my job?"

This paper examines a model for tracking stations and their requirements for inter-processor communications in the next century. The model and requirements are matched with the model and services provided by the five different software architectures and supporting protocol solutions. Several key services are examined in detail to determine which services and protocols most closely match the requirements for the tracking station environment. The study reveals that the protocols are tailored to the problem domains for which they were originally designed. Further, the study reveals that the process control model is the closest match to the tracking station model.

Introduction

Tracking stations are a collection of different pieces of equipment, integrated into a single system to support communications between the ground and a spacecraft. The antenna equipment, the receiver equipment, the transmitting equipment and associated signal processing equipment are built by experts in their field. Over the past decade, computers have been incorporated into this equipment to operate and automate their increasingly complex functions. Today, this computerized equipment (called subsystems) can be linked together with communication protocols into an operating tracking station. However, the degree of difficulty to integrate these subsystems into a single tracking station, and the level of automation that can be achieved, will be a direct function of

the protocol selected. This paper examines a number of non-proprietary protocols that have been used or suggested as possible candidates for the tracking station application.

Today, commercial vendors market computer controlled components for tracking stations. As government budgets shrink and commercial exploitation of space grows, these products offer cost effective solutions to one-of-a-kind development efforts. However, vendors are looking to protect their share of the market and their proprietary products. To this end, some vendors offer complete, fully automated tracking stations. However, these turn-key solutions usually provide limited services. And in general, single vendor solutions are not attractive to government or industry. An "open solution" provides a multi-vendor environment where the best products for the job can be integrated into a single system. Commercial inter-processor communications protocols that provide an "open solution" while affording protection to proprietary products are needed to support the integration of different vendor components into a single automated tracking station.

Operational Scenario

An examination of the various candidate protocols is facilitated with a simple model of a tracking station. Consider the construction of a new tracking station to be built using commercial-off-the-shelf components. Four different companies will provide computer controlled equipment that will be integrated into a fully automated tracking station. The elements include: the antenna subsystem, the receiver subsystem, the telemetry subsystem and the command subsystem (see Figure 1). Each subsystem is operated by a computer integrated with the subsystem hardware. The subsystem computer performs specific functions directly related to the subsystem hardware. A workstation will be used to automate the operation of the tracking station and will provide a central facility to monitor the operation of the tracking station. The workstation and the subsystem controllers will be linked together through a Local Network Area (LAN). All of the software for these systems will be delivered as executable products. All of the systems will be installed and configured without software development, compilation and linking of code. The installation process will be automated to the greatest degree possible.

The operational scenario for this new station implements procedure control through the workstation. The workstation allocates the station resources required to support any given activity at the station. All high level control functions are initiated from the workstation. In turn, the workstation monitors the operation and performance of all the station subsystems and takes action to correct anomalies. Individual subsystems must initiate subordinate subsystem operations as required. And in turn, individual subsystems monitor the operation and performance of subordinate subsystems as necessary. In other words, all operation of the station is coordinated by the workstation, but individual subsystems will control and monitor other subsystems directly. Support files are managed by the workstation and transferred as required to the appropriate subsystem. The scenario outlined above encompasses the six basic functional requirements for monitor and control in the Deep Space Network tracking stations (see Table 1).

X-Windows

Several commercial companies are currently building tracking station components that provide an X-Window based Graphical User Interface (GUI) for operation of their equipment. Several NASA organizations have also provided an X-Windows based Graphical User Interface

(GUI) for operation of NASA developed equipment. Since X-Windows is a common solution to support remote operation of computers and in current use, we should examine its application as a standard for tracking station automation.

A tracking station built to be operated using the X-Windows protocol would require each subsystem to be designed as an x-client. In the example tracking station, each subsystem controller would come equipped with a GUI to support its operation. The Station Operations Workstation would be used as an x-server to operate each subsystem (see Figure 2). This approach permits the development of subsystems in isolation and safeguards the proprietary software of the commercial vendors. However, the X-Windows protocol was developed to support terminal operations on remote computers independent of the manufacturer. It was not designed to support automated operation of the remote computer. Consequently, an operator is required at the Station Operations Workstation to run the remote subsystems. In addition, X-Windows makes no provisions for the direct exchange of data between subsystems without operator involvement. The operational scenario requires subsystems to operate other subsystems and exchange data directly. Consequently, X-Windows alone will not fulfill the automation requirements.

Distributed Computing Environment

The emergence of the Open Software Foundation's (OSF) Distributed Computing Environment (DCE) has prompted speculation that DCE could be applied to the problems of tracking station automation. DCE was designed and developed to provide the services required by systems with multiple computers interconnected by a local area network (LAN) or a wide area network (WAN). As the name suggests, DCE services are designed to perform distributed computing. An underlying assumption for the development of DCE is that the work performed can be independent of location (that is, an application that requests a service is not concerned with where the service is performed). An overview of the DCE basic services with respect to the Open Systems Interconnect (OSI) Basic Reference Model is shown in Figure 3. There are five basic components to DCE:

1) The Distributed File Services (DFS) in DCE provide extensive tools to manage and manipulate files in a distributed computing system.

2) The DCE Time services provide for the synchronization of computer clocks in a distributed computing system.

3) The DCE Naming and Directory Services contains the names of users, machines and resources available in the distributed system

4) The DCE Management Services provide the tools to operate the distribute system.

5) The DCE Security Services control access to the distributed system.

All of these services use the DCE Remote Procedure Call (RPC) to access the network.

The application of DCE in a tracking station would likely rely heavily on the Remote Procedure Call (RPC) for most inter-processor communications. The DCE RPC provides an Interface Definition Language (IDL) which is used to create both client and server elements of an

RPC. The IDL also provides for the common representation of data in different computer systems. Once the IDL specification for an RPC is created, the IDL client and server elements are compiled and linked on the appropriate systems (see Figure 4). Applied to the example tracking station, each subsystem integrated into the station would include a set of client and server IDL definitions. The IDL definitions would be compiled and linked on the Station Operations Workstation. In addition, client and server IDL definitions would be compiled and linked on each subsystem that directly inter-operates with another subsystem. In the example tracking station, the application of the DCE RPC approach would produce the following scenario:

> A receiver subsystem is purchased and delivered along with a set of IDL specifications to support the operation of the receiver. The client IDL specifications are copied to the Station Operations Workstation, compiled and linked. Software is then developed to automate the receiver operation using the RPCs. In addition, the receiver operates as a client to access the antenna positions' values as a part of normal operations. The receiver also operates as a server to the antenna subsystem providing signal power measurements as required. The client and server IDL specifications for inter-operation of the receiver must be copied to the antenna subsystem, compiled and linked to support antenna-to-receiver communications. In turn, the antenna subsystem IDL specifications must be copied to the receiver subsystem, compiled and linked to support receiver-to-antenna communications.

A complex, highly automated tracking station would require hundreds (if not thousands) of RPCs to operate. Consequently, the management of RPCs will become a critical part of any DCE based tracking station. Though the DCE approach may offer a solution to the problems of inter-operability, compiling and linking RPCs from different vendors does not guarantee problem free integration. In addition, the DCE does not address the burden of software development for the Station Operations Workstation to automate the RPC functions.

The application of DCE Management Services (called Distributed Management Environment - DME) offers an alternative solution to compiling and linking IDL specifications into RPCs. The DME services provide high level data object management tools and are based on the Common Management Information Service Element (CMISE) standard. A DME based approach would be very similar to CMISE approach discussed in more detail in a later section.

Simple Network Management Protocol

Simple Network Management Protocol (SNMP) was developed in the Internet community to address the monitor and control of devices that support LANs and WANs. Network bridges and routers are typical devices where SNMP has be applied. To my knowledge, SNMP is not currently used or under consideration for use in tracking station operations. However, SNMP is similar to two protocols currently in use at tracking stations and is very similar to those protocols in its basic design. Therefore, a review of SNMP serves to identify common elements and functions in three similar protocols. In addition, deficiencies in the SNMP approach with respect to tracking station applications are identified.

SNMP provides a set of services designed to access the Management Information Base (MIB) established in a device. The MIB is a collection of objects that represent real resources in the device. For example, a network router used to bridge a local area network to an exterior

communications line will have a network address and sub-network address. Each address can be an object in the network router MIB. The SNMP Get service provides for the retrieval of objects contained in a remote MIB. The SNMP Set service supports the modification of an object in a remote MIB. Also, SNMP has a Trap service that provides for a remote node to report a changed condition to a management node. In addition, software to access SNMP services through a GUI is available for workstations.

The application of SNMP in the example tracking station would find a Management Information Base installed on each subsystem (or device). The Station Operations Workstation would access each subsystem MIB using the SNMP Get and Set services (see Figure 5). The configuration and operation of subsystems would be accomplished using the Set service to change objects in the MIB. The status and performance of the subsystems would be determined by accessing objects in the MIB through the Get service. Anomalous conditions in the subsystems could be reported to the Station Operations Workstation using the Trap service. SNMP provides for the common representation of data through the Basic Encoding Rules (BER) to formulate messages in Abstract Syntax Notation 1 (ASN.1). SNMP services could also be used to support subsystem-to-subsystem communications. Using the antenna-to-receiver example discussed earlier, the receiver would use the Get service to access the antenna positions directly from the antenna subsystem. In turn, the antenna could use the Set service to initiate signal power measurements on the receiver and access the results using the Get service. Finally, commercial software to access SNMP services would be used to automate the Station Operations Workstation

There are however, a number of problems with the application of SNMP in a tracking station. First, SNMP Set and Get services are designed to operate on simple data types: scalars and two-dimensional arrays of scalars. Using SNMP Version 1, access to large sets of MIB data objects require multiple Sets or Gets. The SNMP GetNextRequest can simplify the process but this limitation still imposes performance constraints where large amounts of data must be accessed. SNMP Version 2 will expand the supported data types and add the GetBulkRequest service to address current limitations. Also, SNMP does not provide a service to access a directory to the contents of the MIB. The contents of the MIB can be determined through interrogation with a series of SNMP GetNextRequests, however: it is a time consuming process. A directory to the contents of the MIB is necessary to access specific data objects with Get and Set services. In addition, SNMP provides no mechanism to establish an alias data object. In the antenna-to-receiver example, the object names on both subsystems must match for the antenna or receiver to access each others MIB. For example:

> Company A builds the receiver with the name of the data object representing the operating radio frequency as "RF_Frequency". Company B builds its telemetry processor with the same parameter represented with the name of "Operational_Frequency". Under this condition, an SNMP Get made by the telemetry processor to access the receiver value of "RF_Frequency" would fail and generate an error.

A service to create an alias data object that could be associated with an existing data object would minimize the problems of inter-operation of subsystems. Finally, most implementations of SNMP operate over the User Datagram Protocol which is not a guaranteed delivery service. The successful operation of the tracking station will depend on the inter-subsystem communications. Consequently, a reliable protocol will be required to support the automation of the station.

The SNMP services were designed and developed to manage systems performing dedicated tasks in local and wide area networks. The functions performed by these systems are limited in scope and the services of SNMP reflect that limited scope. The subsystems in the tracking station also perform dedicated tasks; however, the scope of these tasks varies over a wide range of functions. The contents of each subsystem MIB will be completely different and a directory service would simplify the installation and management operations. This deficiency in SNMP could be addressed with implementation requirements imposed on the manufacturers. For example, a file with a directory to the MIB could be delivered with the product, copied to the Station Operations Workstation and made available to an application or user. Similarly, provisions could address the creation of alias named objects in remote MIBs. And, reliable transport services could be furnished by TCP. However, these implementation requirements amount to amendments to the SNMP specification which are unique requirements to the tracking station implementation.

Common Management Information Service Element

The successful implementation by the European Space Operations Center of a tracking station based on Common Management Information Service Element (CMISE) is a compelling rationale for further examination of this protocol. The Consultative Committee for International Telegraph and Telephone (CCITT) and the International Standards Organization (ISO) jointly developed CMISE as the management standard for equipment in the communications industry. The basic approach to the design of CMISE is similar to SNMP, however the eleven services provided by CMISE are more extensive and robust. Like SNMP, the services of CMISE are designed to manage data objects in a MIB. The CMISE Set and Get services are designed to operate on virtually any data type. Consequently, CMISE is not as limited as SNMP. In addition, the CMISE Event service is more robust and sophisticated than the SNMP Trap service. Like SNMP, CMISE provides for the common representation of data through the BER to formulate messages specified in ASN.1. And also like SNMP, CMISE provides no service to access a directory to the contents of the MIB. However, CMISE does provide Create and Delete services that could be used to establish alias data objects on remotes. For example:

> Company A builds the receiver with the name of the data object representing the operating radio frequency as "RF_Frequency". Company B builds its telemetry processor with the same parameter represented with the name of "Operational_Frequency". The telemetry processor would use the CMISE Create service to establish a data object called "Operational_Frequency" on the receiver and associated with the data object "RF_Frequency". The receiver would then respond to a CMISE Get "Operational_Frequency". The association of the two data objects would be part of the subsystem installation procedure. At the end of the activity, the telemetry processor would use the CMISE Delete service to remove "Operational_Frequency" from the MIB of the receiver.

The application of CMISE in the example tracking station, like SNMP, would find a Management Information Base installed on each subsystem (or device). The Station Operations Workstation would access each subsystem MIB using the CMISE Get and Set services (see Figure 5). The configuration and operation of subsystems would be accomplished using the Set service to change objects in the MIB. The status and performance of the subsystems would be determined by accessing objects in the MIB through the Get service. Anomalous conditions in the subsystems could be reported to the Station Operations Workstation using the CMISE Event service. CMISE

services would also be used to support subsystem-to-subsystem communications. Finally, commercial software to access CMISE services would be used to automate the Station Operations Workstation. However, CMISE makes no provisions for file management. Consequently, an additional protocol will be required to move and manage the support files required to operate the subsystems and the station.

Manufacturing Message Specification

The process control protocol Manufacturing Message Specification (MMS) has also been successfully implemented in a tracking station. Originally sponsored by General Motors, MMS provides 86 services designed to support automation of factories. Like SNMP and CMISE, MMS is designed to manage the data objects in a MIB and provides for the common representation of data through the BER to formulate messages in ASN.1. And also like SNMP and CMISE, the systems managed through MMS perform dedicated tasks in the factory. However unlike SNMP or CMISE, MMS was designed to support systems that would span a wide range of manufacturing operations. Consequently, MMS provides 86 services to manage the resources in an automated facility.

The application of MMS in the example tracking station would find a Management Information Base installed on each subsystem (or device). The Station Operations Workstation would access each subsystem MIB using the MMS Read and Write services (see Figure 5). The configuration and operation of subsystems would be accomplished using the Write service to change objects in the MIB. The status and performance of the subsystems would be determined by accessing objects in the MIB through the Read service. Anomalous conditions in the subsystems could be reported to the Station Operations Workstation using the MMS Information Report service and the MMS Event Management services. MMS services would also be used to support subsystem-to-subsystem communications. Unlike CMISE, MMS provides an Identify service, GetCapabilityList service and a GetNamedVariableList service which describe the subsystem on request. The GetNamedVariableList service provides a directory to the contents of the MIB in the form of a list of the named objects contained in the MIB. The integration of different manufacturer's subsystems would be facilitated using the DefineNamedVariable and DeleteVariableAccess services to establish alias data objects on the station subsystems. Returning to the previous example:

> Company A builds the receiver with the name of the data object representing the operating radio frequency as "RF_Frequency". Company B builds its telemetry processor with the same parameter represented with the name of "Operational_Frequency". The telemetry processor would use the MMS DefineNamedVariable service to establish a data object called "Operational_Frequency" on the receiver and associated with the data object "RF_Frequency". The receiver would then respond to a MMS Read "Operational_Frequency". The association of the two data objects would be part of the subsystem installation procedure. At the end of the activity, the telemetry processor would use the DeleteVariableAccess service to remove "Operational_Frequency" from the MIB of the receiver.

Finally, the MMS file management services like FileOpen, FileRead, FileClose, FileDirectory, FileDelete and FileRename would be used to manage the support files required by the subsystems.

Beyond the basics, MMS provides services to support the kinds of subsystems commonly installed in tracking stations. The MMS Program Invocation Management services are designed to support subsystems with multi-tasking operating systems. Using MMS, a standard set of services can be used to start, stop, resume or kill programs running on remote subsystems without regard for the specifics of the target operating system. The Domain Management services support block memory transfers between subsystems. Using the MMS Domain services, subsystem configuration tables could be efficiently transferred between the Station Operations Workstation and the subsystems. The Journal Management services provide for the logging of activities and events in a process control environment. The Semaphore Management services provide support for systems with shared resources. In tracking stations with multiple antennas and limited equipment redundancy, contention for limited resources can be supported through MMS semaphore services.

An additional advantage to the employment of MMS, is the availability of "Application Enabler" products for use on the Station Operations Workstation to automate station operations. These products are commonly found in the manufacturing sector and often referred to as "Supervisory Control and Data Acquisition (SCADA)" packages. Used to automate factories, Application Enabler products are software packages that can be customized for a specific installation without software development. The companies that build Application Enablers provide communication drivers to access proprietary devices, like Programmable Logic Controllers (PLCs). Today, a number of these companies provide MMS communication drivers. Using these products in conjunction with MMS, the software for the Station Operations Workstation can be purchased and configured to operate the tracking station without software development.

Discussion

All five protocols surveyed could be used to build a spacecraft tracking station. However, each of these protocols were designed and developed for a specific environment. The question is "Which environment most closely matches to environment of a spacecraft tracking station?" A second question is "Which protocol will provide commercial vendors with the tools to develop and deliver products that can be installed and integrated without software development?"

Spacecraft tracking stations are composed of devices with dedicated resources performing dedicated operations. The antenna subsystem is dedicated to operating the antenna hardware while the receiver subsystem is dedicated to operating the receiver hardware. The operations performed by these subsystems vary significantly. X-Windows provides an environment for the remote operation of these devices but does not provide for automation. DCE provides an integration environment but does not relieve the burden of software development. The management of a device through its MIB with SNMP, CMISE or MMS can provide automation and relieve the burden of software development. However, the limited services of SNMP make it the least likely candidate for operation of a tracking station. Given the similarities between CMISE and MMS, what is the basis for a final selection? A detailed examination of these two protocols reveals some differences to direct a final selection.

At first glance, the CMISE Get and Set services appear nearly identical in function to the MMS Read and Write services. However, there are subtle differences between the two protocols that are derived from their intended applications. Consider the factory environment:

> A factory is a confined environment where control must be decisive. Arbitrary control of a server might create catastrophic problems on the factory floor. Therefore, an MMS client must establish an association with a server before a dialog of MMS services can begin. If an association can not be established, control can not be initiated. Server systems are designed to fail in a safe mode, protecting the plant and personnel. When problems develop on the factory floor, MMS-based automation alerts operations personnel to investigate the problem and take corrective action. To provide decisive control, the exchange of MMS control messages employs confirmed services that require the client application receive an acknowledgment from the server application. The MMS Write service is a confirmed service that requires acknowledgment for completion.

Now consider a wide area communications network environment:

> A wide area network is not a confined environment, frequently distributed over tens, or hundreds or thousands of miles. Communication device servers are also designed to fail in safe mode while redundancy provides for alternative means of communications. Rarely does a failure present a threat to life or property. Therefore, CMISE is designed to operate with or without an established association. The CMISE Set service can operate in both confirmed and unconfirmed modes.

The difference in these services in important for their respective applications. Corrective action in a factory frequently requires human intervention to safe guard life and property. Corrective action in a communications network can frequently be accomplished remotely. For example:

> A recurring fault can cause a network router to fail. The router can be designed to reboot on failure to safe mode, reboot on failure to diagnostic mode or reboot on failure to operational mode. The recurring failure results in the router continuously rebooting. The time interval between faults is too short to support the normal establishment of an association and leading to a Set service to force the router into the diagnostic mode. The unconfirmed Set service provides a mechanism to reset the router to diagnostic mode before the fault occurs again.

Another subtle difference between CMISE and MMS can be seen in the Event services.

Both CMISE and MMS provide Event services. Though similar in principle, the services perform differently reflecting the environments for which they were designed. A detailed examination of the event data structures reveal that both CMISE and MMS provide an attribute for event-priority. However, only MMS provides an attribute for event-severity. From my experience, I believe this distinction is derived from the difference between the communications environment and the factory environment. Rarely do events in communications networks produce property or life threatening situations. However, events on the factory floor can produce these conditions. Therefore, the MMS Event service provides for severity of a failure.

Consequently, it is my opinion that MMS offers the best fit to the spacecraft tracking station environment. Based on experience, MMS provides the commercial vendors with a standard for automation. Using MMS, a commercial product can be installed and configured into an automate tracking station without site specific software development. And the availability of commercial products for factory automation based on MMS, supports this conclusion. In addition,

MMS based Application Enabler products provide the tools to automate spacecraft tracking stations without traditional software development efforts.

References

Introduction to OSF DCE, Open Software Foundation, Prentice-Hall, Incorporated, 1992.

Schulz, Klaus-Jurgen; Service Management of a Spacecraft Ground Station Network in Support of Spacecraft Operations, Integrated Network Management, III (C-12), 1993 IFIP.

Stallings, William; SNMP, SNMPv2 and CMIP, The Practical Guide to Network-Management Standards, Addison-Wesley Publishing Company, 1993.

Tang, A. & Scoggins, S; Open Networking with OSI, Prentice-Hall, Incorporated, 1992.

International Organization for Standards, ISO 9506, Information Processing Systems - Open Systems Interconnection - Manufacturing Message Specification, 1989.

Heuser, Wm. Randy, Chen, Richard, and Stockett, Michael H.; Using Manufacturing Message Specification for Monitor and Control at Venus, Network Technology Conference, NASA Conference Publication 3241, 1993.

Table 1. This table provides a comparison of the functional requirements (down the left side) for monitor and control in Deep Space Network tracking stations and the protocols examined in this article (across the top).

Protocol / Functional Requirements	X-Windows	DCE	SNMP	CMISE	MMS
Allocation of station resources	Yes	Yes	Yes	Yes	Yes
Configuration and Control of subsystems	Yes	Yes	Yes	Yes	Yes
Monitor status and performance	Yes	Yes	Yes	Yes	Yes
Inter-subsystem data exchange	No	Yes	Yes	Yes	Yes
Event and Alarm handling	No	No	Yes	Yes	Yes
Logging	No	No	No	Yes	Yes
File distribution and management	No	Yes	No	No	Yes
***No software development, compilation and linking**	Yes	No	Yes	Yes	Yes
***Data Object Alias**	No	No	No	Yes	Yes

* Derived requirement to support commercial products derived as executable products.

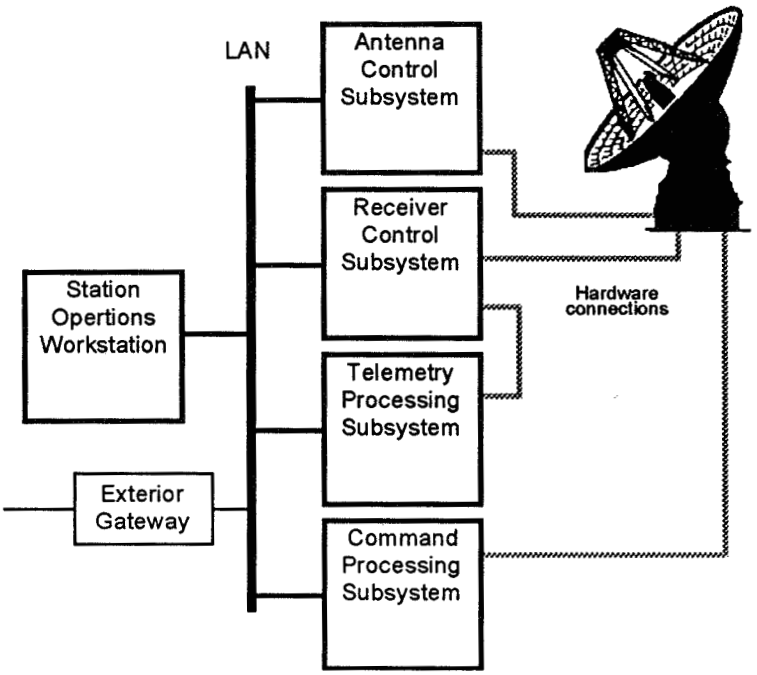


Figure 1. An example tracking station with four computer controlled subsystems inter-connected with a workstation through a Local Area Network.

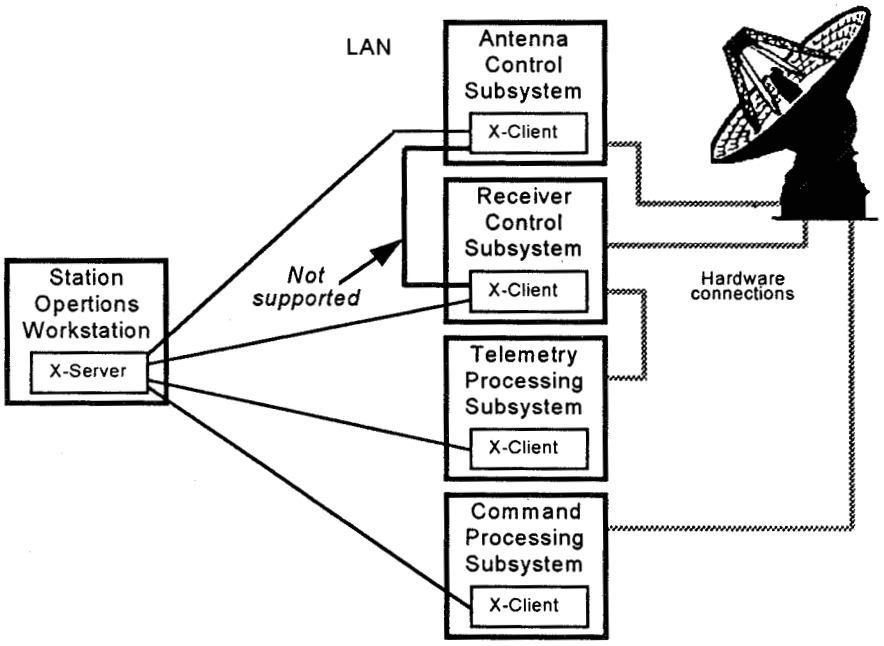


Figure 2. The application of X-Windows to support tracking station integration and automation would require each subsystem to operate as an x-client. The subsystems could be operated from the Station Operations Workstation operating as an x-client server. However, direct subsystem-to-subsystem data exchange is not supported by X-Windows.

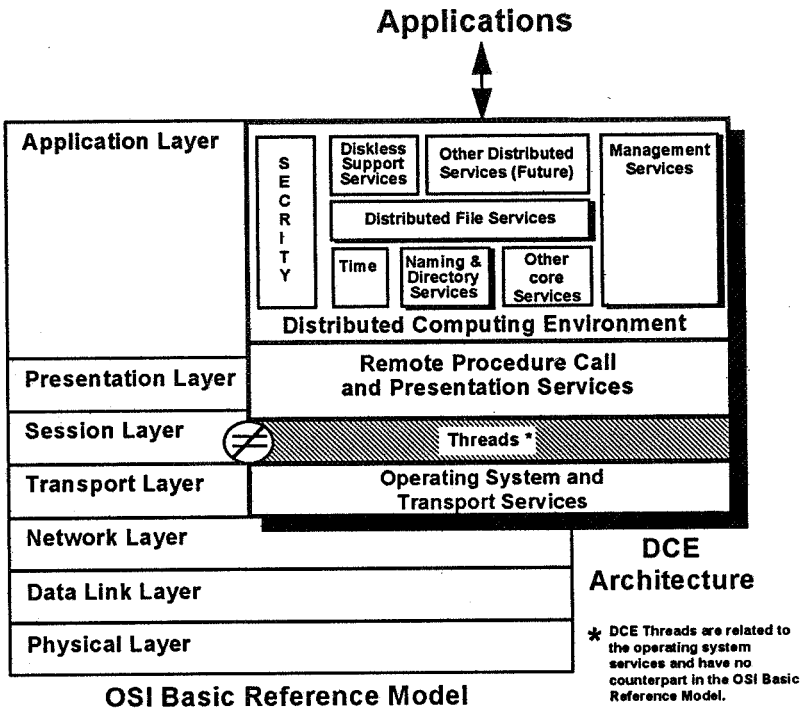


Figure 3. This figure shows the relationship between the DCE Architecture and the OSI Basic Reference Model.

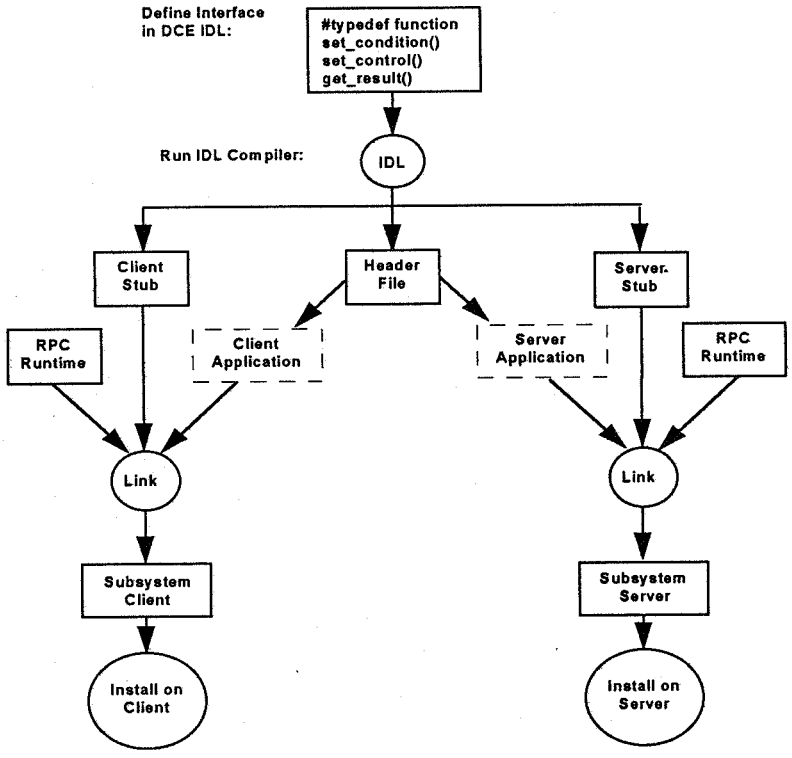


Figure 4. This figure shows the DCE process to create remote procedure calls from DCE IDL.

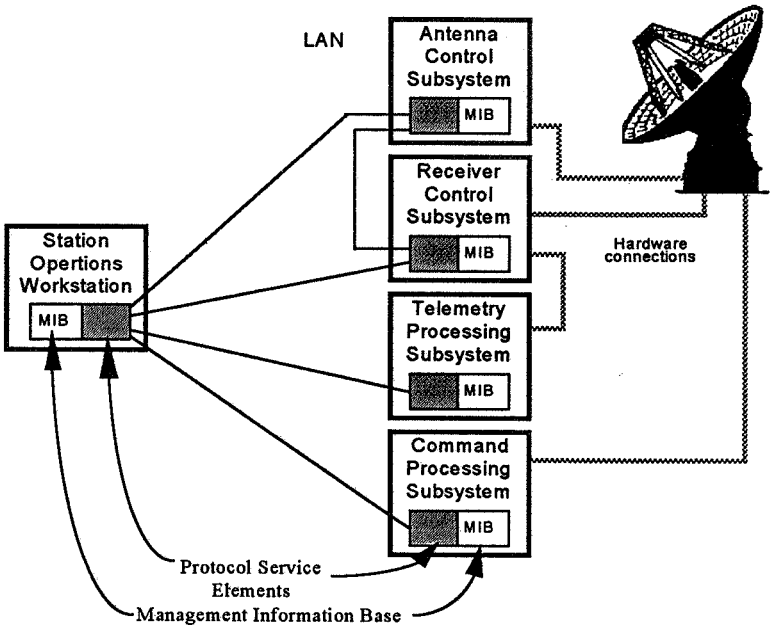


Figure 5. SNMP, CMIS/CMIP and MMS all provide services to access and manage a Management Information Base (MIB) on remote systems. In this example, the operator workstation provides monitor and control the of subsystems in a simple receiver only tracking station through services that access the MIB.

111478

N95-17555

354210
P-9

An Agent-Oriented Approach to Automated Mission Operations

Walt Truszkowski
NASA Goddard Space Flight Center, Code 522.3
Greenbelt, MD 20771
Email: truszkow@kong.gsfc.nasa.gov

Jidé Odubiyi
Loral AeroSys, 7375 Executive Place
Seabrook, MD 20706
Email: jideo@groucho.aerosys.loral.com

Abstract

As we plan for the next generation of Mission Operations Control Center (MOCC) systems, there are many opportunities for the increased utilization of innovative knowledge-based technologies.

The innovative technology, discussed in this paper, is an advanced use of agent-oriented approaches to the automation of mission operations. The paper presents an overview of this technology and discusses applied operational scenarios currently being investigated and prototyped. A major focus of the current work is the development of a simple user mechanism that would empower operations staff members to create, in real time, software agents to assist them in common, labor intensive operations tasks. These operational tasks would include: handling routine data and information management functions; amplifying the capabilities of a spacecraft analyst/operator to rapidly identify, analyze, and correct spacecraft anomalies by correlating complex data/information sets and filtering error messages; improving routine monitoring and trend analysis by detecting common failure signatures; and serving as a sentinel for spacecraft changes during critical maneuvers enhancing the system's capabilities to support non-routine operational conditions with minimum additional staff.

An agent-based testbed is under development. This testbed will allow us to: (1) more clearly understand the intricacies of applying agent-based technology in support of the advanced automation of mission operations, and (2) to access the full set of benefits that can be realized by the proper application of agent-oriented technology in a mission operations environment. The testbed under development addresses some of the data management and report generation functions for the Explorer Platform (EP)/Extreme UltraViolet Explorer (EUVE) Flight Operations Team (FOT). We present an overview of agent-oriented technology and a detailed report on the operation's concept for the testbed.

1.0 Introduction

Major advances have been made in the process of automating mission operations over the last several years. However, in keeping with changing operational requirements and the need to more effectively realize cost and manpower savings in the area of mission operations, the necessity for more advanced automation technologies is clear. As examples of areas for continued improvement consider the following:

- Mission Operations Control Center (MOCC) software systems are currently developed using classical software engineering paradigms. To bring about added degrees of flexibility in how these systems could handle unexpected problems, the engineering of these systems along agent-oriented technology lines looks promising.

PRECEDING PAGE BLANK NOT FILMED

PAGE 878 INTENTIONALLY BLANK

- Even with the increasing use of expert systems in support of telemetry monitoring and command constraint checking much reliance is placed on the manual intervention of operators. The use of agent-oriented techniques can effectively provide additional levels of automated support in handling these important types of operational activities and further reduce the need for manual interventions.

- With increasing automation of mission operations, there is a growing need for more advanced approaches to information handling. The use of agent-technology in support of the full range of information management functions will significantly reduce the growing possibilities of information overload on the part of operators.

It is becoming apparent that for future automated mission operations, more consideration will have to be given to the roles that distributed problem solving and computer-supported cooperative work will play. These increasingly important issues can be addressed by employing intelligent, distributed processes [6] found in a multi-agent based approach, described in this paper.

The rest of this paper presents an ontology [12], i.e., a conceptual framework for describing the mission operations domain, and an implementation framework for automating the operations in that domain. Our approach for dealing with the task of developing an agent-based mission operations environment is to first specialize by applying our agent methodology to automate the report generation function. Once this is accomplished we will then generalize and apply the agent-based approach to other functions in the MOCC as shown in Figure 1 below. Our approach for generating the agent-based report-generation solution is to employ an information agent model and define agent roles in a multi-agent environment required for this selected subdomain function.

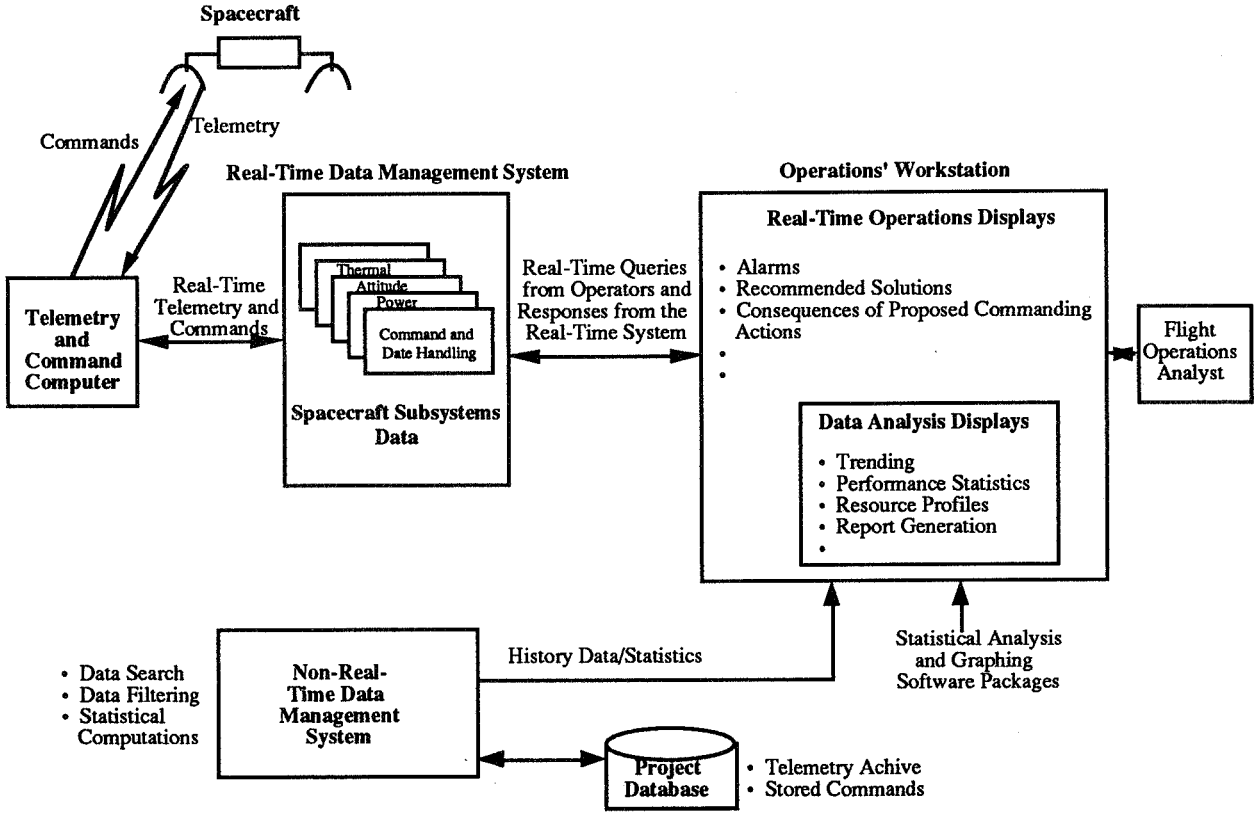


Figure 1: An Overview of a Spacecraft Mission Operations Control Center

We present related work on the use of agent-based approaches for automating information location and retrieval systems and the contribution of our investigation in proving the utility of agent-based technology in mission operations.

2.0 The EP/EUVE Report Generation Process

The EP/EUVE's operational environment is a heterogeneous network consisting of two MicroVAXs (VMS), a Sun workstation (UNIX), an HP-9000 workstation (UNIX), and i386 personal computers, using the X.25 and TCP/IP protocol.

Subsystem engineers for EP/EUVE are responsible for daily monitoring of the satellite's subsystem performance, detection of anomalous subsystem behavior, weekly reporting of subsystem performance, generation of commanding products for subsystem operations, and continuing preparation for subsystem anomaly Detection, Isolation, and Resolution (DIR). These products reside on heterogeneous distributed computing nodes. Off-line analysis (Trend system) provides daily plots of over 600 mnemonics for visual checks of subsystem performance and trends. Subsystem engineers' performance is evaluated based on how well they handle a spacecraft anomaly, not on their daily activities. For example, based on an analysis of operator activities over a period of time, it was concluded that 90 percent of their time is spent performing daily routines. Each week, three of the Explorer Platform's engineers spend a total of 40 hours generating a weekly report on the performance of the system. The routine activities that consume most of the operators' time can be automated to allow them to spend time on more critical tasks.

Three categories of reports are generated by the Flight Operations Team (FOT) of the EP/EUVE system. The three subreports which correspond to the three subsystems of the EP/EUVE are the Modular Power Subsystem (MPS) subreport, the Command and Data Handling Subsystem subreport, and the Modular Attitude Control Subsystem subreport. Other subreports included in the MPS subreports include reports on the Battery Health and Safety, the Solar Array Performance, the MPS Heater Duty Cycle, the Critical MPS Events' Summary, and the Thermal System. The critical MPS Events' Summary is generated from the computer workstation which generates the Real-time and Trend data.

Adequate preparation for a spacecraft anomaly's DIR is the key to successful spacecraft flight operations. The level of preparedness depends on the amount of "spare time" a spacecraft subsystem engineer has to study the subsystem, and the time between anomaly detection and resolution. Automating the report generation process will allow the spacecraft subsystem engineer to devote their time to more productive mission operations such as early detection of anomalies, data analysis, and development of scenarios for anomaly prevention.

The subreports for the Command and Data Handling Subsystem result from collecting six other subreports. The subreports are orbit decay (EP/EUVE 's decrease in orbit periods), tape recorder performance, clock delta trends, transponder performance, Ultra State Oscillator frequency trends, and Modular Antenna Pointing Control.

3.0 An Agent-Oriented Solution to Support the Report Generation Process

An Agent-based FLight Operations AssociaTe (AFLOAT) is currently being prototyped to support the FOT in generating weekly reports. Each agent is an entity that can function semi-autonomously in an environment where other agents exist, accept instructions from a user, and communicate with other agents. In addition, it can be persistent, and can migrate from one node to another to process and retrieve information as requested. The agent can operate independently in the background without interfering with user's actions. An overview of agent-oriented technology and our approach for applying this technology to automate the EP/EUVE operations report generation process are described in the following paragraphs.

3.1 An Overview of Agent-oriented Technology

What is an agent? In the most general form, a software agent as opposed to a hardware agent (e.g., a robot) can be defined as an entity that enables a user to specify what the user wants leaving the process of how and when to accomplish it to the agent [3]. Huhns and Singh [5] present a more comprehensive definition for a software agent as an active knowledge-based computational entity that has knowledge, intentions, and mechanisms for perceiving, reasoning, acting, and communicating. An agent, in our initial prototype, is characterized by a subset of the capabilities of the agent in this comprehensive definition, as explained in paragraph 3.5.

3.2 Distinction between Agent-based Systems and other Computer System Services

There is general confusion on what agent-based systems are and how they differ from other computer system services such as Directory Assistance Programs and Information Brokers [5]. Directory Assistance Programs support interoperation between conventional software programs by accepting requests and routing them to appropriate programs for execution. Information Brokers or Distributed Object Managers such as the Common Object Request Brokering Architecture (CORBA) the Distributed Information Manager (DIM) for EOSDIS, and the Dynamic Data Exchange (DDE) programs, either statically or dynamically provide access to information making the source of the information transparent to the user. In addition to serving as directory assistants, they can also execute requests and return results. All these system services use procedures to communicate with other objects. True software agents use declarative directives that are more expressive to reason and communicate complex concepts with other agents instead of relying on procedural directives which are efficient but they are less expressive.

3.3 Agent Types

An agent's behavior may vary along a spectrum of factors ranging from a controlled learning process to self-learning, controlled behavior to full independence, and simple to complex interactions. An agent's capability may be simple or complex; its interaction with its environment may be reactive or planned (i.e., deliberative). Reactive agents [2] are robot-like with very limited internal reasoning mechanisms while deliberative agents [4] have substantial reasoning capabilities. The agents in a multi-agent system may or may not coordinate their activities. All the agents may be identical or each may be unique, and they may communicate either by directed message passing or broadcast. The number of agents may range from a single agent to thousands. As you will see in paragraph 3.5, the agents in our prototype will be able to learn; each has some degree of independence. The agents can interact with their environment with deliberative reasoning, and communicate with one another through direct message passing and multicast via shared memory.

3.4 Essential Architectural Issues of Multi-agent Systems

To be successful in developing a multi-agent based system, the following four architectural issues must be addressed and the fifth issue is optional: (1) an approach must be established for describing, decomposing and distributing tasks among the agents; (2) a format must be defined for interaction and communication between agents; (3) a strategy must be formulated for distributing controls among agents in which a local control strategy demands that agents communicate only their results, or centralized control where one agent assigns all the tasks, or a predefined mixed results/tasks share control; (4) a policy must be made for coordinating the activities of agents, either by competition through negotiation as in ContractNet Protocol or cooperation through centralized or distributed planning; and (5) a rationale should be established for maintaining truths, i.e., consistent beliefs and conflict resolutions among agents [6] or mental states or trusts [10]. Our

current architecture, described below, can address the five architectural problems. Our initial goal is to resolve the first four issues making the resolution of the fifth issue a long-term goal.

3.5 A System Overview of an Agent-based Solution to Automate Mission Operations

Our objective is to develop AFLOAT as a multi-agent based system where a user interface agent interacts with the user to accept user requests, collaborates with other agents at a local host or over the Internet in locating, retrieving, and presenting the information to the user in appropriate form, with the correct amount and level of detail, and at the right time. An implementation framework for AFLOAT consists of an architecture for a software agent, a methodology for implementing the interactions between the user and a user interface agent, collaboration between multiple agents, and an approach for making background software agents specialize in data retrieval from distributed information sources. User-to-agent and agent-to-agent interaction issues are resolved by developing a communication protocol, a language format, and an agent migration process across networked computer systems. Our strategy for information location and retrieval is based on the premise that domain-dependent keywords used by the user will form an index to the information in the domain and to the specialized agent. If the key word does not exist, then retrieval is not possible, and the user interface agent will issue appropriate advice. Knowledge in AFLOAT can be stored as rules, objects, cases (examples), models, and programs. Each agent has access to a set of support services such as: creating, destroying, managing, or monitoring the activities of spawned agents; mechanisms for message transport; directory of other agents; information processing and presentation; and system performance monitoring.

In addition to supporting on-line and off-line flight operations of the EP/EUVE report generation process, agents in AFLOAT can also support the spacecraft platform and instrument Fault Detection, Isolation, and Recovery (FDIR) services.

Our architecture for automating mission operations has been designed to address the top four basic architectural issues and to be extensible enough to accommodate the fifth. The implementation framework is based on a deliberative agent architecture, depicted in Figure 2. The architecture has structural elements for data storage, coordination, and monitoring of activities between agents, execution of internal and external functions, inter-agent communication, and interface with other domains in the MOCC.

Architecture of AFLOAT's Deliberative Software Agent. Each of AFLOAT's software agents is deliberative, which means that it will reason before it acts. An architecture of such a software agent in AFLOAT is displayed in Figure 2. It addresses the issues that must be resolved in a deliberative multi-agent based system. The **coordinator** determines the type of coordination (task sharing or result sharing), and coordination policy (negotiation, shared memory, or an explicit domain-driven task delegation policy) that will be employed. In AFLOAT, agents coordinate their activities by sharing results, and an explicit domain-driven task delegation policy is employed since each agent is considered a specialist in a specific domain. The agent's coordinator module is also responsible for planning and scheduling the tasks of each agent. Each agent's **monitor** is responsible for monitoring interactions between agents, incoming and outgoing messages, the state of the agent, and maintaining a history of the agent's actions. Saving an agent's past actions aids it in learning by drawing from experience when presented with new tasks. The **external models** module of each agent maintains global functions that are accessible for use by other agents. Each agent must maintain its access rights to external information so as to aid the domain agents in the information retrieval process. The **internal models** module maintains functions (such as managing access to the skills of each agent or maintaining its message buffer) that are private to each agent and are not accessible to external agents except the AFLOAT executive agent. Each agent also has an **inter-agent communication module** which is responsible for validating inter-agent, semi-structured language format, sending outgoing messages, receiving incoming messages, and broadcasting messages to shared memory. The brain of each agent is its

information base where all the modules store their data and other information such as the name of the local system management agent (AFLOAT executive), buffers for incoming and outgoing messages, each agent's name, type, and state, and messages in shared memory. Communication with each agent is done by adding a message to its information base. Each agent can store knowledge as rules, objects, cases (examples), models, and programs. The structure of each agent, coupled with its behavior (i.e., capabilities) provides it with enough intelligence to respond effectively to information retrieval tasks delegated to it.

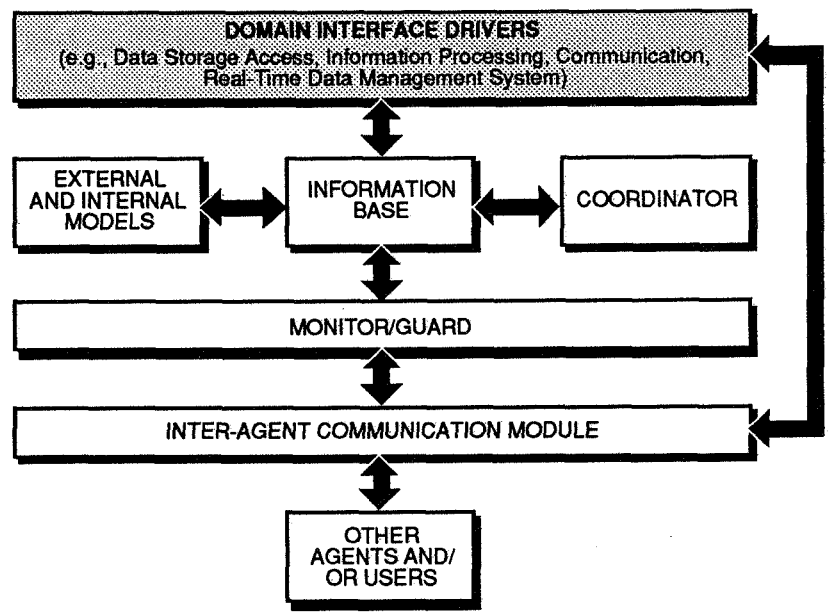
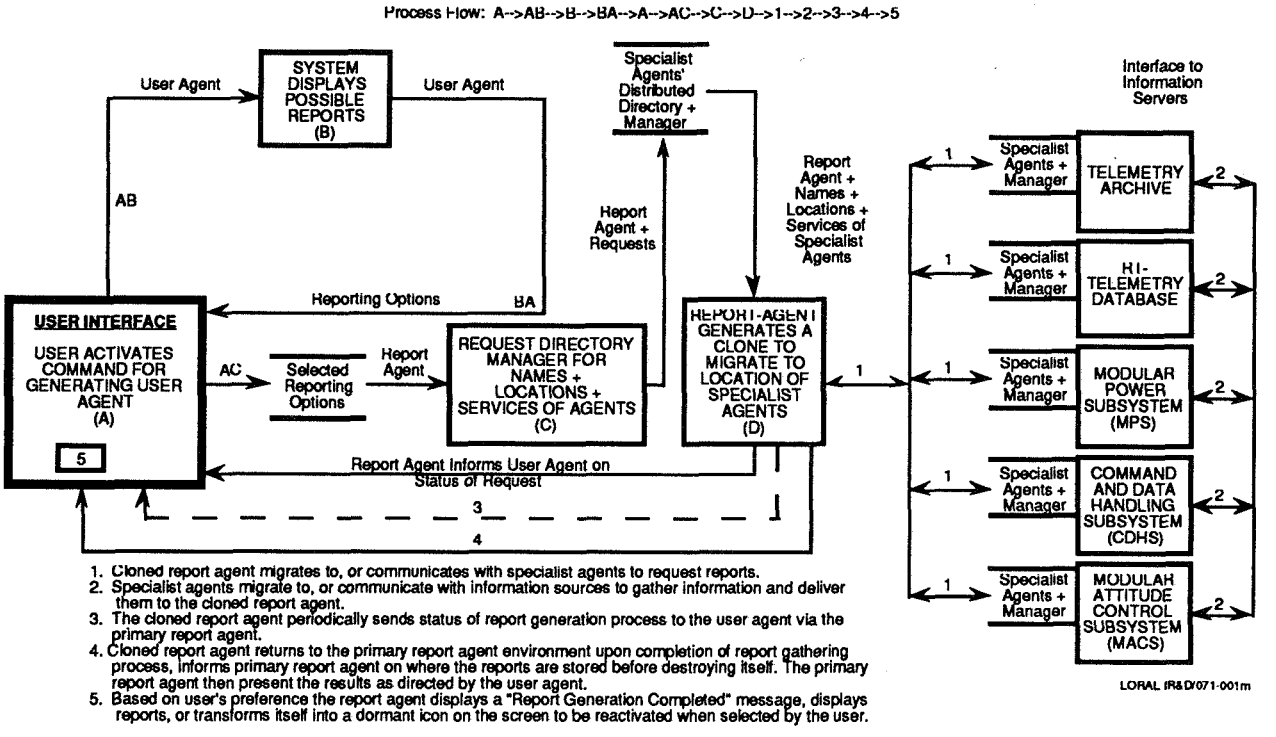


Figure 2. Architecture of AFLOAT's Deliberative Software Agent

An Information Agent Model for Supporting Information Retrieval: Agents in AFLOAT are characterized by five "action-oriented" [9] capabilities: First, migration, is the ability of an agent to move to other nodes to process or retrieve information. This ability can support load balancing, improve efficiencies of communication, and provide unique services which may not be available at a local node. Second, semi-autonomy, is the ability to respond to a dynamic environment without human intervention, thus improving the productivity of the user. Third, spawning, is the ability to create other agents to support the parent agent, thereby promoting dynamic parallelism and thus fault-tolerance. Fourth, persistence, is the ability to recover from environmental crashes and support time-extended activities, thus reducing the need for constant poling of the agent's welfare and better use of the system's communication bandwidth. The fifth and final capability is interaction mechanisms for supporting agent-to-agent and user-to-agent interactions.

Operations Concept for AFLOAT Prototype: An operations concept for the AFLOAT testbed prototype is illustrated in Figure 3. It describes the procedures for using agents to locate, access, retrieve and present EP/EUVE reports or information located at remote information sources. To do this, the user generates a user agent. The user agent requests the system to display a set of reporting options. The user then selects one or more items from the list displayed by the system. Upon completing the selection process, the user agent generates a report agent and assigns it the responsibility of generating the reports. The report agent identifies specific subreports and requests the agents' directory manager (or name/skill server) for the names, locations, and services provided by agents that can support the generation of requested reports.



1. Cloned report agent migrates to, or communicates with specialist agents to request reports.
2. Specialist agents migrate to, or communicate with information sources to gather information and deliver them to the cloned report agent.
3. The cloned report agent periodically sends status of report generation process to the user agent via the primary report agent.
4. Cloned report agent returns to the primary report agent environment upon completion of report gathering process, informs primary report agent on where the reports are stored before destroying itself. The primary report agent then present the results as directed by the user agent.
5. Based on user's preference the report agent displays a "Report Generation Completed" message, displays reports, or transforms itself into a dormant icon on the screen to be reactivated when selected by the user.

Figure 3: Operations Concept for Agent-based FLight Operations AssociaTe (AFLOAT)

In addition to knowing the names, locations and services provided by the specialist agent, the report agent must also determine if there are restrictions to services provided at certain locations. If an access is restricted to information sources or there is an absence of unique services required by specialist agents, the report agent may request the reports remotely via message passing. If there are no such restrictions, the report agent generates and sends a clone with enough information necessary to generate the report to migrate to remote information sources, interact with agents with special skills, and retrieve the reports. Allowing the report agent to send its clone to retrieve reports while it stays at the user's environment adds some fault tolerance to the system. Therefore, if the cloned report agent fails, the primary agent has all the information needed to create another clone. Periodically, the cloned report agent informs the primary report agent resident at the user's environment on the progress of the report generation process. Steps 1, 2, 3, 4, and 5 in Figure 3 explain the interactions between the agents and the report generation process.

Development Environment and Status and Plans for AFLOAT Project: The development environment for implementing AFLOAT is the NASA/Johnson Space Center developed C-Language Integrated Production System (CLIPS) version 6.0 with CLIPsTOOL

software from KNOWARE Inc. (for building the user interface) running on a Sun SPARCstation with UNIX operating system, X-windows, and OSF/Motif Style Guide. The application of the defmodule construct (in CLIPS) which promotes the partitioning of knowledge bases will enable us to achieve agent independence. We have just completed Build 1 of the AFLOAT testbed. This build provides location transparency to information sources for generating reports on Battery Charge/Discharge ratios of the three batteries on the spacecraft. This build is also being used by two George Washington University researchers to investigate the issue of trust of automated systems. In their experiment, an operator is assigned a task that he/she must perform plus an additional task of monitoring the quality and number of faults correctly detected by the agents. The operator's trust level of the agent is based on the frequency and types of incorrect faults. Build 2 of AFLOAT will provide the users with the ability to generate reports on the operations of the three subsystems, i.e., the MPS, the CDHS, and the MACS from distributed information sources.

4.0 Related Agent-based Information Retrieval Systems

Several agent-based information retrieval systems are being prototyped at several research laboratories. Most of the research work attempts to resolve the fundamental architectural issues described earlier in paragraph 3.4. The research work of Amy Lansky at NASA/Ames [8], and Bond and Gasser [1] focuses on multi-agent planning and addresses the issues of coordination, synchronization, and control of multiple autonomous agents. Shoham's work [10] investigates the issue of an agent's mental states as they relate to beliefs, intentions, and capabilities. Other research on agent-based information retrieval similar to ours include the work by Kahn and Cerf [6] in which agents, called Knowbots, each hard coded to perform a specific task, are used to retrieve information from digital libraries. Etzioni's work [3] on Softbots employs software agents to perform different UNIX tasks to support a UNIX programmer. A very important contribution of his work is the ability of the Softbots to retrieve information with an incomplete request. Papazoglou and Laufmann [9] employ coarse-grained agents with a semi-structured language and message passing to support information retrieval from distributed information sources. The semi-structured language format is quite expressive and it can help the agents in communicating their goals, results, and states, thus facilitating coordination among the agents. Gio Wiederhold [12] employs very coarse-grained agents called mediators which can be used to filter data by resolving any mismatches in the data. A major contribution of the mediator approach is the merit of this architecture over integrated or federated agent-based system architectures. While it is more difficult to implement, the mediator architecture is easier to scale up and add new interfaces than the other two.

While each of the research efforts described above address various aspects of the architectural issues of multi-agent systems, AFLOAT's architecture has been built as an extensible testbed and it can address all the basic architectural problems of a multi-agent-based system. In addition to its capability to automate distributed information retrieval, it can also support automation of other operations such as fault detection, isolation and recovery of satellite subsystems, and other domains. Whereas in a large majority of other multi-agent systems, the base prototyping language is either LISP or PROLOG which very often is not well received by the operations staff due to a lack of experienced programmers; AFLOAT is based on an expressive AI shell written in C with the UNIX operating system, making it readily portable to other platforms and acceptable to operations staff.

5.0 Conclusion

The distributed nature of the operations in a satellite MOCC calls for solution approaches to problems in the domain to consider the use of intelligent distributed modules instead of isolated intelligent systems. Such intelligent distributed modules have been modeled as a multi-agent system and prototyped as the AFLOAT testbed to support the automated report generation process, and described in this paper. An overview of agent-based technology has been presented with

essential architectural issues that must be addressed to successfully implement a multi-agent based system to support automated mission operations. We have shown how the architecture of each agent coupled with its behaviors (i.e., its capabilities represented as an information agent model), can be used to resolve basic architectural problems of multi-agent systems.

The use of multi-agent based designs is not limited to the mission operations domain. They can be employed in any environment where the user needs to delegate an associate to perform information management activities such as in telecommunications network management, software reuse management, and automated traffic incident management systems.

6.0 Acknowledgements

This work originated with a paper by Truszkowski and Moore [11]. We wish to thank Mike Moore for his major contributions to the development of the agent model and testbed concepts which are now being prototyped. The funding for this work is being provided by NASA Headquarters Code O (Office of Space Communications).

Bibliography

1. Bond, A. H., and Gasser, L. (1988). *Readings in Distributed Artificial Intelligence* , Morgan Kaufmann, San Mateo, CA.

2. Brooks, Rodney A. (April, 1991). *Intelligence Without Reason, Computers and Thought,* IJCAI-91, AI Memo No. 1293, pp. 1-27.

3. Etzioni, Ören, (1994, July). *A Softbot-Based Interface to the Internet,* Communications of the ACM, pp. 72-76

4. Ferguson, Innes A. (1992, May). *Touring Machines: Autonomous Agents with Attitudes,* IEEE Computer, pp. 51-55.

5. Genesereth, M. R., and Ketchpel, S. P. (1994, July). *Software Agents,* Communications of the ACM, pp. 48-53

6. Huhns, M. N., and Singh, M. P. (1994, May). *Distributed Artificial Intelligence for Information Systems* , MCC, Austin, TX 78759.

7. Knoblock, C. A., Arens, Y., and Hsu, C. (1994, May). *Cooperating Agents for Information Retrieval* , Second CoopIS-94 Proceedings, University of Toronto, Canada, pp. 122-133.

8. Lansky, Amy, (1994, April). *Data Analysis Assistant,* Recom/NASA Ames Research Ctr.

9. Papazoglou, M., Laufmann, S., and Sellis, T. K. (1992) *An Organizational Framework for Cooperating Intelligent Information Systems,* International Journal of Intelligent and Collaborative Information Systems, Vol. 1, No. 1, pp. 169-202.

10. Shoham, Yoav. (1990). *Agent-Oriented Programming, Technical Report,* STAN-CS-1335-90, Robotics Laboratory, Computer Science Dept., Stanford University, Stanford, CA.

11. Truszkowski, Walt, and Moore, M. (1992). *Towards an Information Ecology,* AIP Conference Proceedings 283, pp. 884-892.

12. Wiederhold, Gio. (February 1992). *Mediators in the Architectures of Future Information Systems,* IEEE Computer, pp. 38-49.

111141

N95-17556

225-7

354214

P-8

ADVANCED GROUND STATION ARCHITECTURE

David Zillig (NASA GSFC/Code 531.2)
Ted Benjamin (Stanford Telecomm, Reston, VA)

ABSTRACT

This paper describes a new station architecture for NASA's Ground Network (GN). The architecture makes efficient use of emerging technologies to provide dramatic reductions in size, operational complexity, and operational and maintenance costs. The architecture, which is based on recent receiver work sponsored by the Office of Space Communications Advanced Systems Program, allows integration of both GN and Space Network (SN) modes of operation in the same electronics system. It is highly configurable through software and the use of Charged Coupled Device (CCD) technology to provide a wide range of operating modes. Moreover, it affords modularity of features which are optional depending on the application. The resulting system incorporates advanced RF, digital, and remote control technology capable of introducing significant operational, performance, and cost benefits to a variety of NASA communications and tracking applications.

INTRODUCTION

The NASA Ground Network (GN) station architecture has been used very successfully over the last 25 years to support a multitude of low earth orbiters (LEO's), expendable launch vehicles (ELV's), geosynchronous (GEO's) and lunar missions in the Spaceflight Tracking and Data Network (STDN). The GN RF subsystem, based on the Multifunction, polarization diversity Receiver (MFR) and the STDN tone ranging equipment, still provides extensive support to NASA programs. This support includes: (1) Shuttle launch and landing at GN stations; (2) LEO's, including Small Explorer spacecraft at the DSN 26-meter subnet stations; (3) TDRS GEO spacecraft at the GN, DSN 26-meter subnet, and GRO Remote Terminal System (GRTS) stations. Its hardware has been upgraded and replaced over the years to maintain its ability to provide reliable support to NASA's critical missions, but its basic architecture remains the same as when the STDN was formed in the early 70's from the Space Tracking and Data Acquisition Network (STADAN) and the Apollo Manned Space Flight Network (MSFN).

While individual functional blocks have been and could continue to be replaced by modern electronics, it is expected that the biggest gains will result from developing a new system architecture that makes the most efficient use of emerging technologies for the most dramatic reductions in size, operational complexity, and operational and maintenance costs.

During the past year, GSFC/Code 531 has been studying new ground station architectures capable of high levels of hardware integration. The architecture incorporates flexible software configurability for implementation of a wide range of modes, and is designed specifically for effective automation of most operational and maintenance functions. The hardware systems are designed to mate with the overall station control philosophy of the Automated Ground Network System (AGNS). AGNS is based on an open architecture comprised of loosely coupled station subsystems (such as the RF subsystem of concern here) that maximize the use of commercial standards and interfaces.

A highly integrated, automated ground station with the capability of meeting stringent Shuttle S-Band communications and tracking requirements can also serve as the next generation near-earth-to-lunar, multipurpose ground terminal. It also lends itself to applications requiring compact, transportable systems and remotely controlled stations that supply direct downlinks to small satellite experimenters.

This paper briefly reviews current GN station architectures and hardware configurations. It then presents functional and signal processing requirements for the upgrade RF subsystem. The advanced station architecture is then described, followed by sections detailing the flexible advanced

PRECEDING PAGE BLANK NOT FILMED

PAGE 888

equipment that support this new station architecture.

CURRENT STATION ARCHITECTURE

The current GN station RF equipment is primarily comprised of what is referred to as the RER -- Receiver, Exciter, Range equipment. As configured today, these three basic functions are distinct equipments, each associated with a dedicated rack of electronics. For example, the MFR receiver consists of a 7' rack containing 7 equipment drawers or modules. The exact hardware configuration varies from station to station depending on specific requirements. A typical RER equipment group is comprised of about 6-7 equipment racks to support a user satellite link.

The GN station antenna system provides a sum signal (Σ) and two error signals (X and Y) in each of two orthogonal polarizations ---- resulting in six receiver input channels. The MFR performs optimal ratio combining of these orthogonal polarized signals and, thus, is referred to as a polarization diversity receiver. Experience has shown that this is a critical GN function that allows continuous operations even through significant fades of the polarized signal that is dominant through most of a pass. To accomplish this processing, and provide redundancy to meet stringent reliability requirements, 4-5 MFRs are typically used (each MFR requiring a rack of equipment) to support user services.

As noted above, substantial equipments are currently required to meet GN mission requirements. This, coupled with the fact that the underlying processing architecture is more than 20 years old, places a substantial burden on GN operations and maintenance. This situation is exasperated as the GN stations are called upon to support new and expanded requirements as user mission needs evolve.

DESIGN GOALS/REQUIREMENTS

General requirements and design goals are first presented. Key receiver, ranging, and transmit requirements are then discussed, in turn.

General Requirements. Except for some few obsolete requirements (e.g., FM Uplink), the RER Upgrade must support all current RER capabilities, and meet or exceed associated performance requirements. To accommodate Space Station, and the Shuttle Launch Support System (SLSS), the RER Upgrade must also support SN signal modes. This capability can serve as a ground-based SN backup capability. Support of both SN and GN modes by a GN ground terminal affords the option to user missions to reduce transponder power and weight by having only a SN mode capability.

SN modes use suppressed carrier modulation, as well as PN spread spectrum signalling. Spread spectrum operation also provides benefits by allowing NASA to mitigate RF interference into, as well as from NASA satcom links --- a key concern as the RF spectrum becomes increasingly crowded.

Receiver. The receiver must perform the following basic functions: Telemetry Data Demodulation, Polarization Combining, Baseband Telemetry Data Processing, Autotrack, Range Tracking, and Doppler Tracking.

Telemetry data demodulation is required for both SN and GN signals, involving both residual and suppressed carrier formats. Moreover, in the GN mode, up to 3 subcarriers may need to be supported (e.g., engine data from Shuttle's three main engines). The following signal modulations are possible, involving symbol rates from 100 bps to 5-10 Msps.

- Carrier PM Modulated by Data, Range Tones and PSK Subcarriers (0-3)
- Carrier PM Modulated by Data, CW Range Subcarrier (Shuttle)
- Carrier FM Modulated by PSK-Modulated Subcarrier (Shuttle Engine Data)
- Carrier FM Modulated by TV/Analog Data
- Carrier FM Modulated by Digital Data (FSK)
- BPSK, QPSK, PN/BPSK, SQPN.

Polarization combining of orthogonal polarized signals has been an important and necessary feature of the current GN MFRs. The extent that polarization is needed varies from spacecraft to spacecraft and even pass to pass. Polarization combining seems to be particularly critical for high elevation passes.

The antenna system provides X-axis (δx) and Y-axis (δy) error signals, each in two orthogonal polarizations, to the receiver as part of the autotrack function. Analogous to the processing of the two orthogonal sum channels (Σ_A and Σ_B), the receiver must: (1) optimally combine the orthogonal error signals for each axis, (2) amplitude detect the combined signal, and (3) provide the recovered X and Y error signals to the antenna tracking system.

Ranging. The existing GN ranging function is implemented in separate equipment from that of the exciter and receiver. For the RER Upgrade, an important goal is to integrate the ranging function into the receiver and exciter. This approach reduces and simplifies equipment, and thereby, reduces operations and maintenance costs. GN ranging is a tone ranging system, in which transit time is determined by comparing the phases of transit and receive tones. Tones from 500 KHz to 10 Hz are used, in conjunction with an ambiguity resolving PN code for range ambiguities of 644,000 Km. An accuracy of 1 meter (1 σ) is required at a C/N_0 of 50 dB-Hz.

Transmitter. The transmitter must perform the following basic functions: Command Data, Modulation, Range Tone Generation, Test Signal Generation, Frequency Upconversion (to S-band), Range Zero Set, and Command Echo Verification.

For the uplink command signal, the modulation is required to provide for (1) GN Mode: a PM signal with either data/range tones directly on the carrier or on a subcarrier, and (2) SN Mode: PSK signal with/without PN spreading. To enhance overall operability and maintainability, the transmitter must also be capable to operate as a test signal generator for the receiver, which requires the generation of all the input signal modes and formats noted earlier for the receiver.

In summary the RER Upgrade must not only meet current GN and SN requirements, but also provide this capability in a fashion that reduces costs and enhances operations. Also critical is that the Upgrade be compliant with AGNS, by facilitating high-levels of automation and standard interfaces.

UPGRADE RER ARCHITECTURE

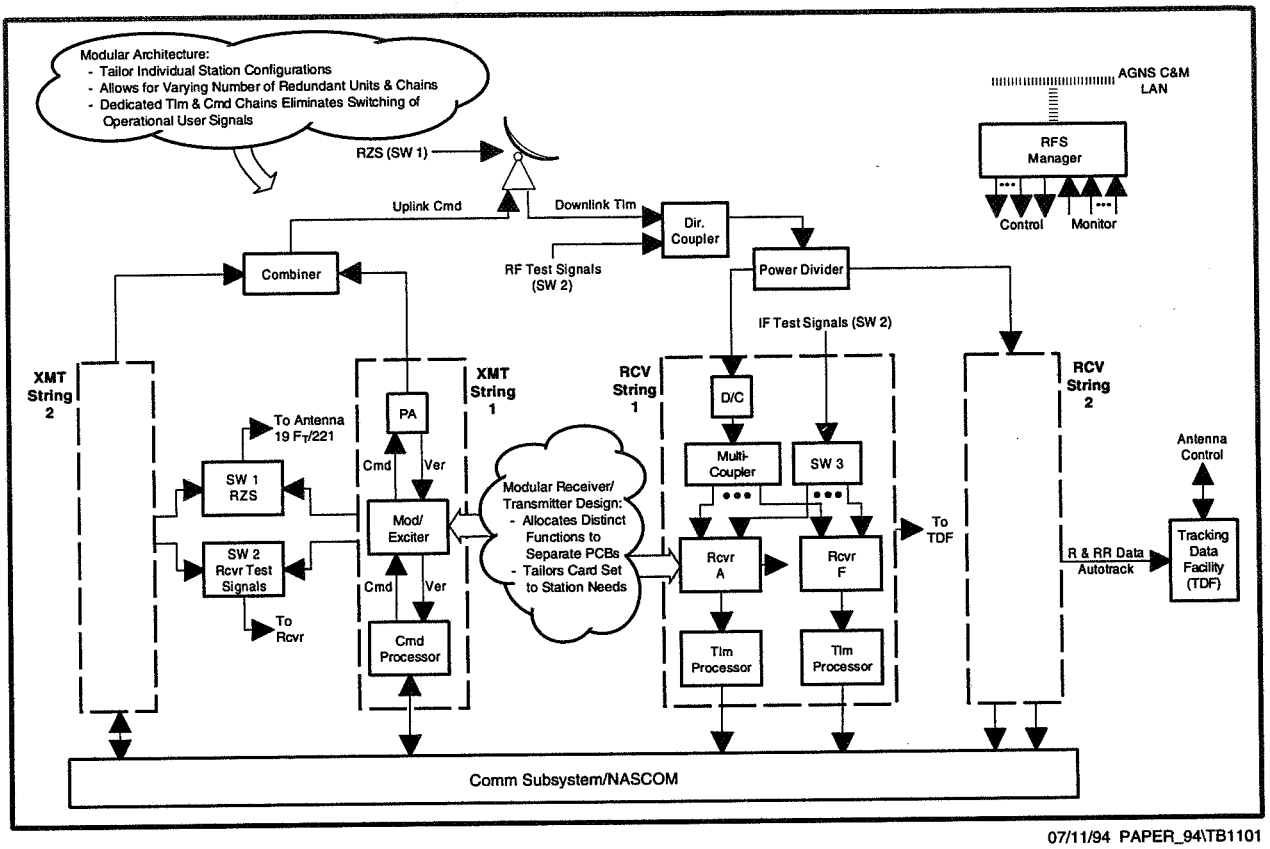
In response to the above needs, an upgrade architecture has been developed and is shown in Exhibit 1. Both uplink command and downlink telemetry signal processing are implemented in equipment chains or strings. Key features to note are:

- A processing "chain" consists of dedicated equipments that handle all processing between baseband and RF, thereby effectively eliminating all switching in operational signal paths
- Levels of reliability are achieved through redundant processing chains, which can operate in various "stand-by" modes, depending on outage/contingency requirements
- Additional receive chain reliability is achieved by configuring two or more receivers within each receive chain at the multicoupler output.

The upgrade architecture is modular, flexible, and expandable --- critical characteristics to meet current and future growth requirements. Accordingly, each station can tailor the specific number of chains and redundant units within chains to suit their individual needs and service support requirements. For example, Shuttle support, which requires high reliability, may be achieved with additional processing chains and/or additional receiver units within a receive telemetry chain.

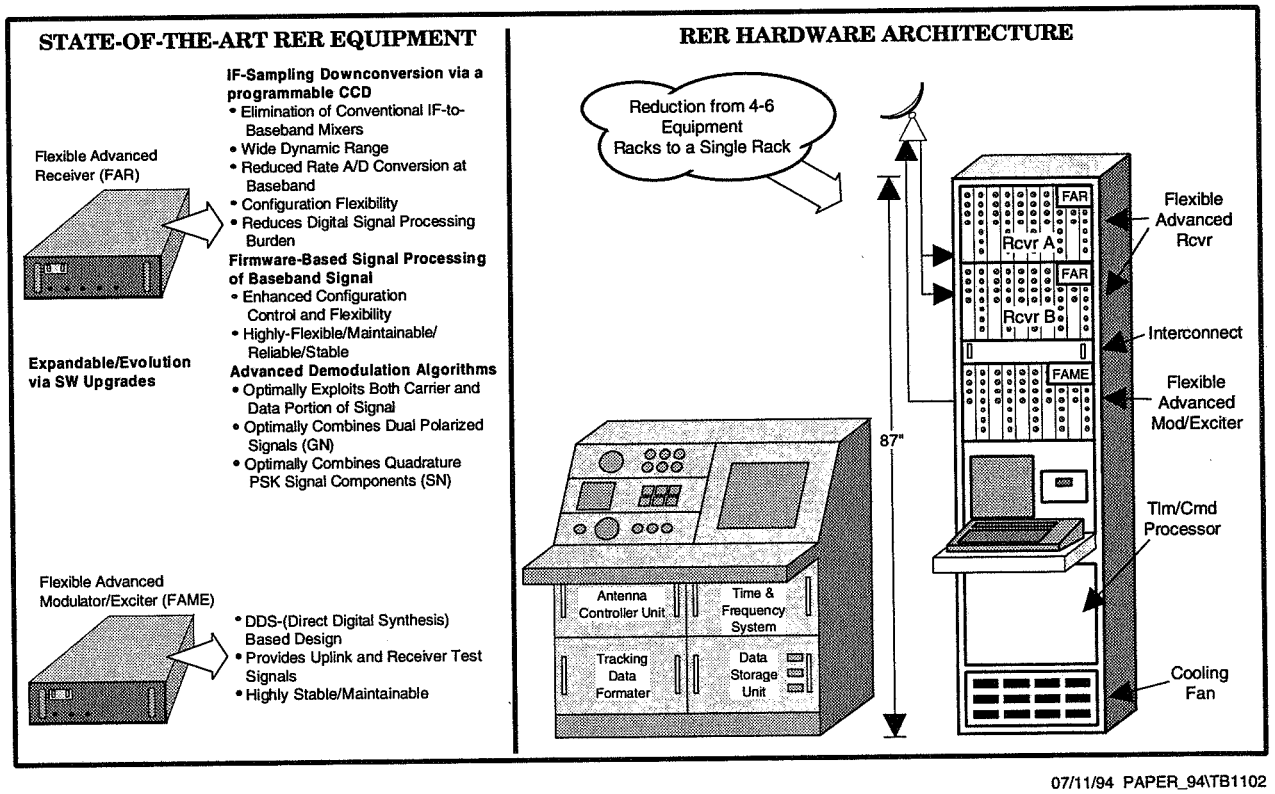
This so-called "string" architecture has also been adopted by NASA's STGT (Second TDRSS Ground Terminal) in response to lessons learned from WSGT, which uses a pooled equipment approach to architecture. The GN Upgrade architectural approach is greatly facilitated by advanced flexible receiver/transmitter units (described below) which are compact and relatively low cost. Today's rack of equipment for a single receiver or transmitter can be reduced to a single chassis or drawer within a rack.

In another related effort, all telemetry baseband processing is being performed within a single PC, further enhancing the "string" architecture approach. Based on these efforts and advances in signal processing, Exhibit 2 depicts the corresponding hardware configuration that supports the advanced



07/11/94 PAPER_94\TB1101

Exhibit 1: RER Upgrade Architecture



07/11/94 PAPER_94\TB1102

Exhibit 2: Advanced Architecture Hardware Description

Station Architecture. In effect, one command chain and one telemetry chain (with two receiver units) can be reduced to a single rack -- a reduction of more than 4 to 1.

RECEIVER ARCHITECTURE

A receiver design has been developed to meet the requirements and design goals noted earlier. The receiver, referred to as the Flexible Advanced Receiver (FAR), is an evolution of the advanced CCD/Software receiver technology developed under sponsorship of NASA HQ/Code O (Advanced Systems) and GSFC/Code 531. The FAR is a state-of-the-art (SOA) system employing novel architecture and advanced technology to provide extensive capability in a compact package. Moreover, as the name indicates, much of the receiver processing is performed in software which promotes the desired flexibility and maintainability.

The receiver is comprised of two fundamental processing blocks that maximize the use of SOA analog processing, employing programmable CCD's (Charged Coupled Devices) followed by firmware processing, using multiple Motorola DSP96002 DSP chips. The CCD is essentially an analog tapped delay line with programmable tap weights. The FAR CCD is the 2-ATC chip which is the latest of Lincoln Lab's programmable CCD chips. The 2-ATC chip is specifically tailored for NASA/SN applications, and was developed under sponsorship of NASA HQ/Code O (Advanced Systems) and GSFC/Code 531. The resulting architecture is extremely powerful, yet flexible to support a wide range of signal formats and conditions through software changes only.

Exhibit 3 presents the FAR receiver architecture, showing support to all six input channels required to handle polarization combining and autotrack processing. As shown, there are four basic modules whose functionality is highlighted below:

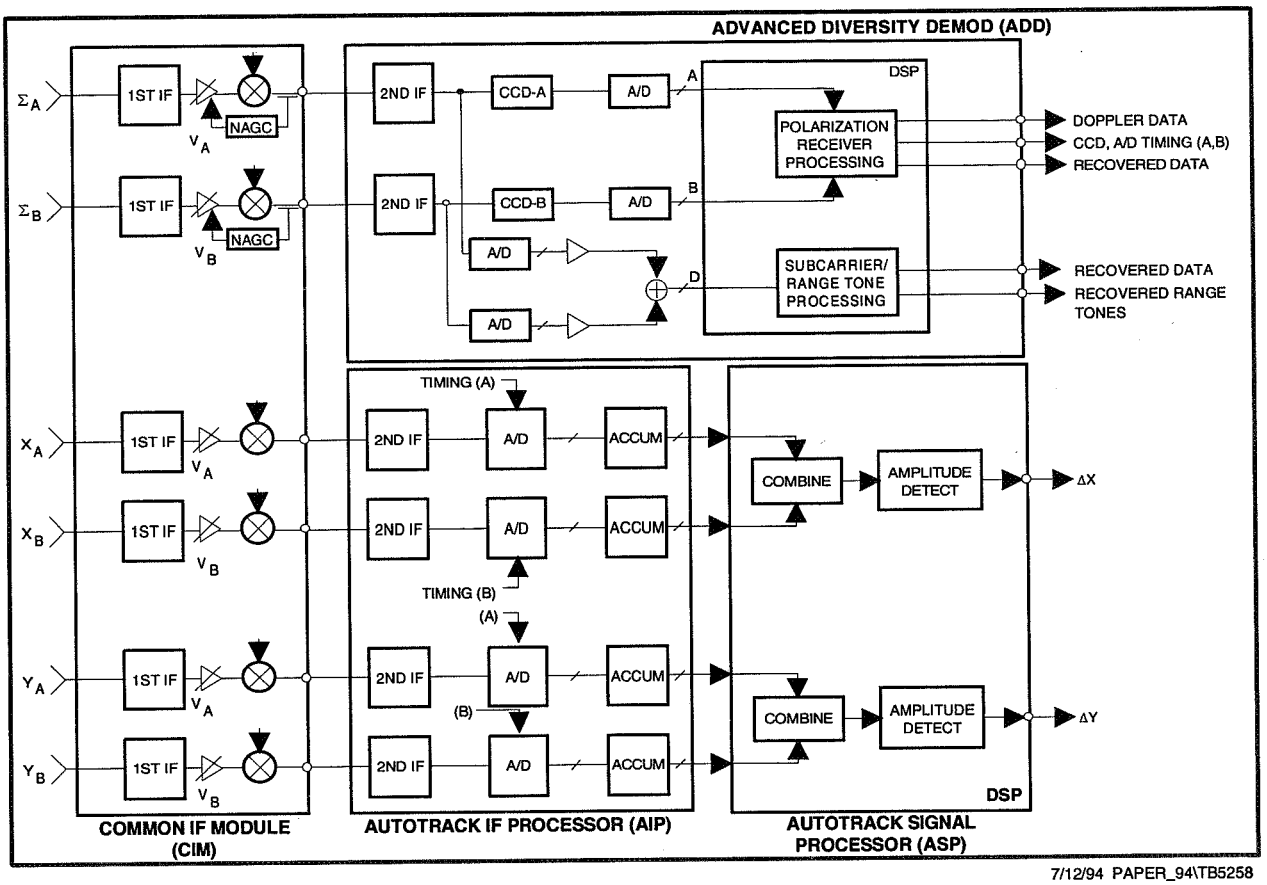
- Common IF Module
  - Tunes 1st IF to a Common Fixed IF (e.g., 140 MHz)
  - Performs Noncoherent AGC on Wideband Input Signal
- Advanced Diversity Demod (ADD)
  - Optimally Combines GN Orthogonal Polarized Sum Channels (Σ_A and Σ_B)
  - Optimally Processes SN PSK Quadrature Components (I and Q)
  - Demodulates Carrier/Subcarriers to Provide Telemetry Data & Range Tones
- Autotrack IF Processor (AIP)
  - Provides Digital Difference Channel Samples to ASP
- Autotrack Signal Processor (ASP)
  - Combines Dual Polarized Channels
  - Provides Amplitudes to Antenna Subsystem for Antenna Pointing.

Preliminary design analysis indicates that the FAR receiver, in its full capability, will consist of 15 printed circuit boards or cards. Noteworthy is that specific functionality is assigned to distinct cards, so that a station needing less capability can simply remove corresponding cards and save costs. For example, a user not requiring autotrack can reduce the card set by five. The card set is comprised of a combination of COTS (Commercial-off-the-shelf) and custom cards.

The heart of the FAR is the Advanced Diversity Demod (ADD) which provides the powerful signal processing capability. The ADD high-level architecture is shown in Exhibit 4, which depicts the analog front-end followed by DSP firmware processing.

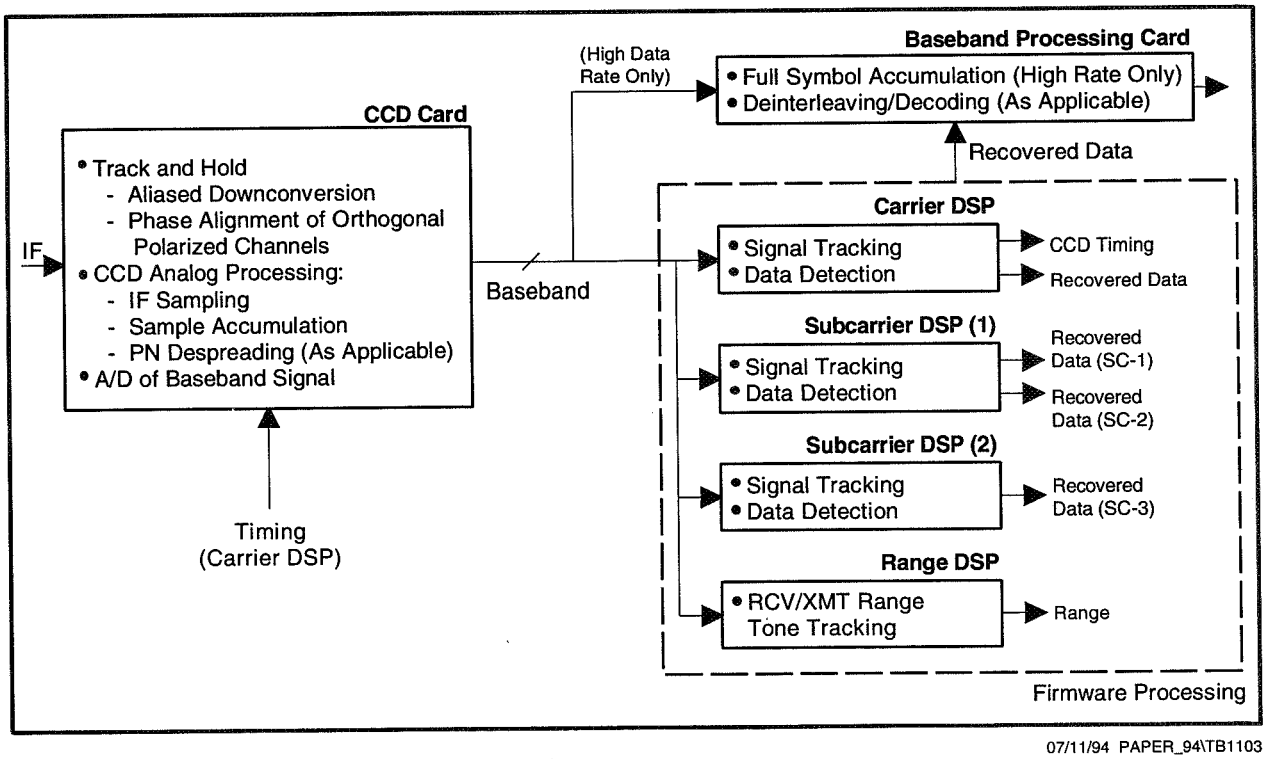
The CCD card receives the IF sum channels (Σ_A and Σ_B) from the Common IF module. The input IF is 140 MHz, and is downconverted to a third IF through a novel scheme using a Track and Hold Amplifier (THA). The THA, whose sample rate is controlled using a NCO provides an aliased signal component at a lower IF which is extracted by the anti-aliasing low-pass filter.

The lower IF is then IF-sampled by the CCD to provide an analog sampled baseband output signal. The signal consists of alternate quadrature I and Q samples. Relative to conventional mixing to baseband, IF sampling eliminates the "sin/cos" mixers, and provides all the information in a single path, with substantially reduced complexity. Moreover, by appropriately adjusting the CCD programmable tap weights, the CCD performs as a



7/12/94 PAPER_94\TB5258

Exhibit 3: Advanced Receiver Architecture



07/11/94 PAPER_94\TB1103

Exhibit 4: Advanced Diversity Demod (ADD) Architecture

matched filter in three ways: (1) matching to the alternating {1.-1} of the "peaks" of the IF CW, (2) data matched filtering by accumulating samples from the same symbol within a CCD length, and (3) PN code despreading with a local PN code (for spread spectrum signalling).

The CCD weighted-sample accumulation is a critical aspect of this unique architecture in that it greatly reduces the processing requirements imposed on the subsequent digital/firmware processing. This, coupled with the wide dynamic range inherent in analog processing, provide significant benefits over pure digital receivers. Furthermore, the 2-ATC chip provides two distinct CCDs on a single chip (ideally suited for two orthogonal polarized signals or quadrature QPSK components) offering the potential for compact, low power applications.

The analog CCD output is A/D converted and provided to the digital cards for signal processing-- all in μP firmware. There are four DSP cards to handle carrier, subcarrier, and range processing. All DSP cards are identical, having the same hardware architecture. Key features are listed below:

- Four 32-Bit DSP96002 Floating Point DSPs
  - Arranged in a Fully Interconnected Modified Hypercube Architecture
  - Operating at 20 MIPS each with Full Resource Redundancy
- Design Repetition at Each Processor Standardizes Programmer's Interface
- Serial Communications
  - 4 LAN and up to 8 Serial Ports
  - Eurobus Digital Interface Facilitates System Expansion through Memory-Mapped Add-On Cards.

Receiver signal processing uses the receiver architecture discussed above to perform the following basic functions: (1) Signal Tracking (carrier, symbol, PN code), (2) Polarization Combining, (3) Subcarrier Processing, and (4) Range Processing. All signal processing performed to support signal tracking is performed in DSP firmware that, in turn, adjusts appropriate NCOs to effect tracking. An "integrated" receiver tracking approach is used in which, for example, the symbol synchronizer is used for data-directed carrier tracking operations. This improves overall SNR performance relative to conventional Costas Loop operation for PSK signalling. For the FAR, it is also applied in a novel way to optimally demodulate PM modulated signals.

TRANSMITTER ARCHITECTURE

To complement the receiver performance upgrades, and support the overall RER Upgrade architecture, a new, flexible transmitter design has been developed. The new transmitter architecture, described in Exhibit 5, is referred to as the Flexible Advanced Modulator/Exciter (FAME). It makes use of emerging technologies such as Direct Digital Synthesis (DDS) and embedded micro-controllers that allow for effective automation.

The FAME architecture is divided into five functional blocks: (1) Baseband Modulator, (2) Upconverter, (3) Verification Receiver, (4) Synthesizer, and (5) FAME Controller.

The Baseband Modulator stands to benefit significantly from DDS technology. Exhibit 6 is the high-level Baseband Modulator architecture, and shows the extensive use of highly integrated ASICs now available for DDS, Forward Error Correction (FEC), and PN Coding. Use of ASICs promises dramatic size reductions as well as enhanced automatic control capability. To make the transmitter as flexible as possible and make efficient use of the emerging technology, eight MUXs allow for the routing of digitally represented waveforms in a variety of paths such that it can assemble a diverse set of signal structures. The DDS ASICs themselves offer excellent phase and frequency resolution with minimal or no calibration.

SUMMARY

An advanced station architecture has been designed that promises to substantially reduce equipment and operational complexity. The architecture is based on new, flexible receiver and transmitter units that uniquely leverage the state-of-the-art in both analog (e.g., CCDs) and digital signal processing (DSPs) technologies. Noteworthy is that the capabilities of this equipment can simply evolve and expand through software changes.

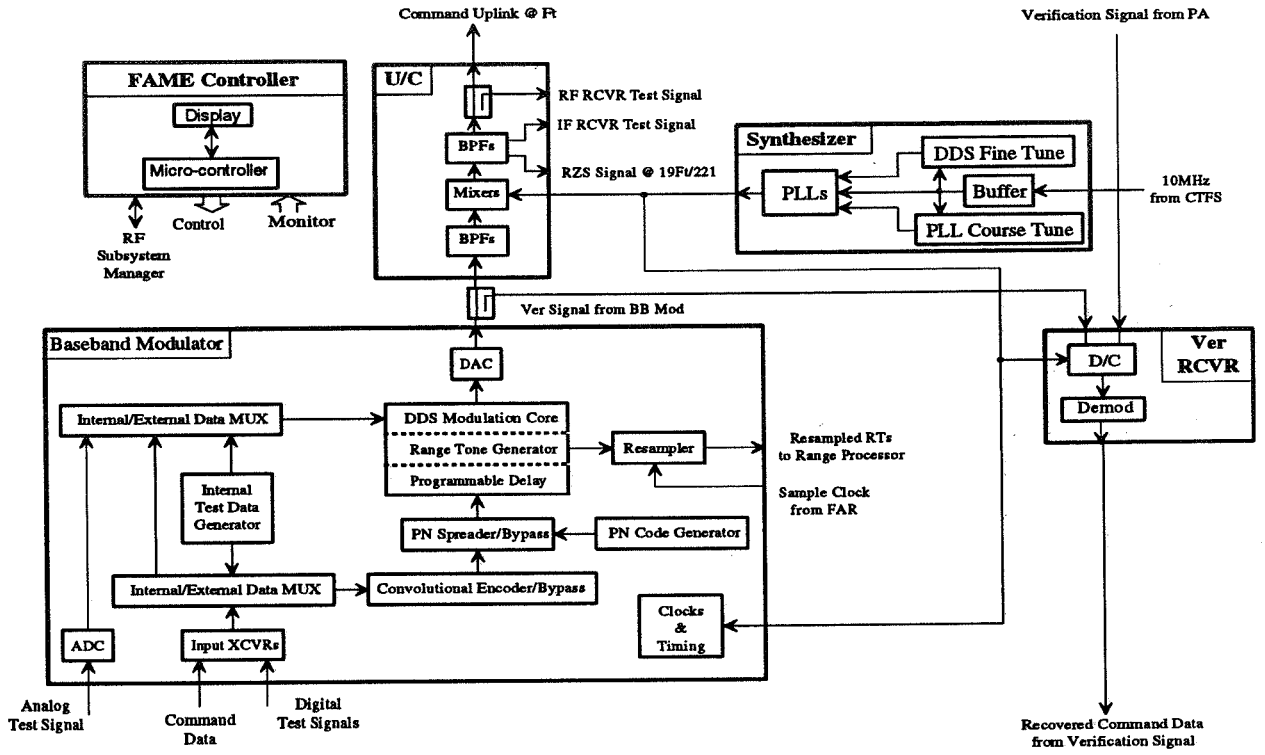


Exhibit 5: Flexible Advanced Modulator/Exciter Architecture

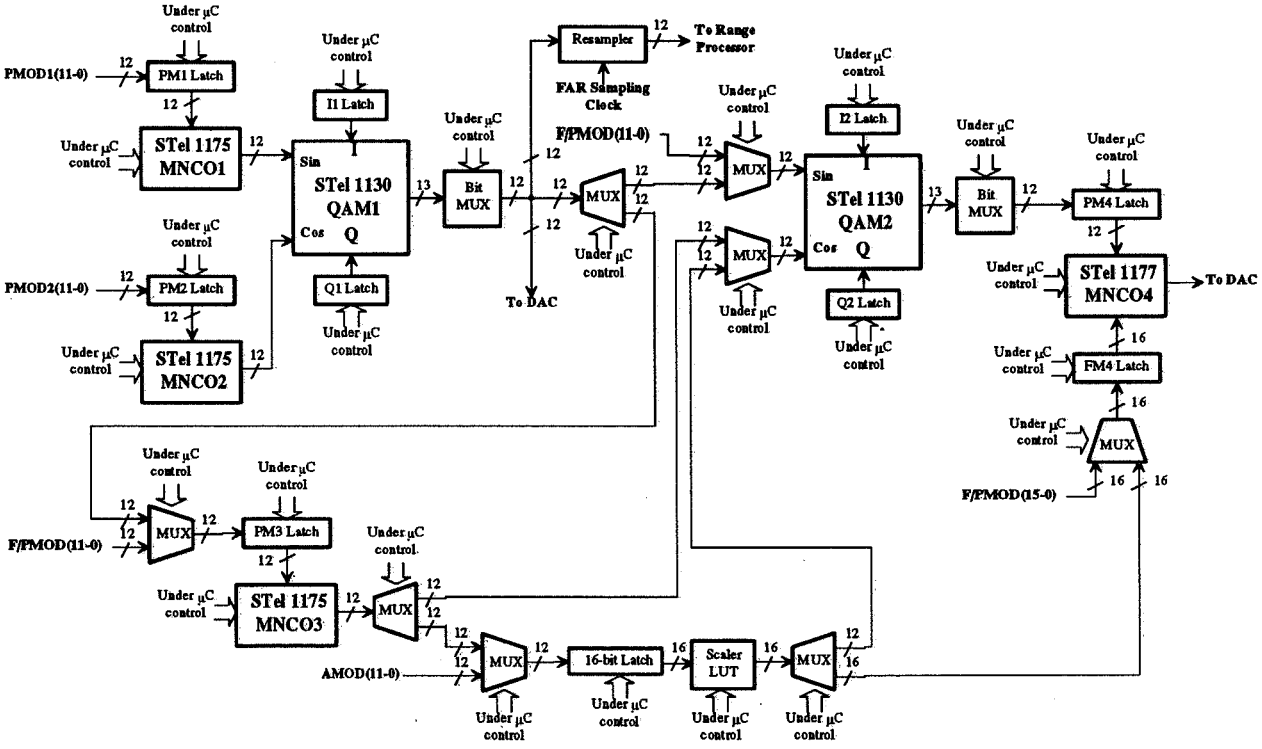


Exhibit 6: Baseband Modulator Design

* *Presented in Poster Session*

111942 N95-17557

354215 p. 6

Automating Testbed Documentation and Database Access Using World Wide Web (WWW) Tools

Charles Ames
Jet Propulsion Laboratory
California Institute of Technology
Mail stop 179-206
Pasadena, CA 91109
818 354 7098
818 393 6154 (fax)
chuck@tsunami.jpl.nasa.gov

Brent Auernheimer
California State University
Dept. of Computer Science
Fresno, CA 93740-0109
209 278 2573
209 278 4197 (fax)
brent_auernheimer@CSUFresno.edu

Young H. Lee
Jet Propulsion Laboratory
California Institute of Technology
Mail stop 301-340
Pasadena, CA 91109
818 354 1326
818 393 4100 (fax)
young@natashi.jpl.nasa.gov

Abstract

A method for providing uniform transparent access to disparate distributed information systems was demonstrated. A prototype testing interface was developed to access documentation and information using publicly available hypermedia tools. The prototype gives testers a uniform, platform-independent user interface to on-line documentation, user manuals, and mission-specific test and operations data. Mosaic was the common user interface, and HTML (Hypertext Markup Language) provided hypertext capability.

Introduction

The Jet Propulsion Laboratory's Test Engineering Laboratory (TEL) evaluates new technologies for possible use during spacecraft system testing.

Formal test environments are highly structured and information intensive. Information that may be useful for later analysis of failure reports or change requests is not always obvious during system test. Clearly, it is better to err on the side of collecting data that may never be used. Testers also consult numerous reference documents, including test plans, handbooks, acronym lists, and glossaries.

For these reasons, spacecraft system testing is a paper-intensive operation. The project described in this paper addresses this problem using freely-available, multi-platform hypertext interfaces.

Several NASA centers support related work. An inter-center working group, ICED[1] (InterCenter Electronic Documentation workgroup) is informally organized to share information among groups exploring the use of hyper- and multi-media interfaces to testing, operations, and ground data systems.

This paper is organized as follows: the context of the prototype, the JPL system test environment, is described; next, the development of the prototype is outlined; the transition from prototype to product is documented; finally, future work is described.

The JPL System Test Environment

JPL's Advanced Multi-Mission Operations System (AMMOS) is a networked computer system consisting of 28 software and hardware subsystems. Its principle purposes are to sequence and uplink commands to spacecraft and to process downlinked telemetry. Both testers and users provide feedback to AMMOS developers about needed repairs and improvements in the form of Failure Reports (FRs) and Change Requests (CRs) which are stored in the Anomaly Tracking System (ATS) database. Developers and testers refer to this database to prioritize their work.

[1]ICED has regularly scheduled teleconferences and maintains an on-line repository of findings. The contact person for ICED is Anthony Griffith, agriffith@jscprofs.nasa.gov.

PRECEDING PAGE BLANK NOT FILMED

PAGE 898 INTENTIONALLY BLANK

Preparation for system test occurs in parallel with system development. Test preparations include: writing test plans; organizing test cases, data, and scenarios into test procedures; defining acceptance criteria; and negotiating the test schedule.

System verification and validation includes functional, performance, security, and reliability testing. Test logs are maintained, reports are generated, and FRs are written detailing software, hardware, or configuration failures. Engineers generate CRs in response to FRs. A change board approves or disapproves each CR after impact analysis.

As proof of concept, a variety of physical documents used by testers were converted to hypertext. These documents include:

- References: Test Engineering Handbook, Acronym List, and Glossary
- AMMOS User manuals and guides
- Flight project specific documents: test plans, procedures, and reports
- Articles posted to the Internet about software testing.

More than 4MB of testbed specific documents were converted to hypertext. All of these documents are accessible through a WAIS (Wide Area Information Server) full-text search and retrieval [WAIS]. Figure 1 is the result of a WAIS search of software testing articles.

HTML (HyperText Markup Language) was used to decorate text with hypertext tags (links and anchors), and to make explicit the logical structure of documents [HTML]. A client-server relationship is a fundamental assumption behind the use of markup languages and related presentation clients (viewers). That is, authors embed tags in their documents to make the logical document structure discernible by client viewer programs. For example, an author may wish to organize information as a bulleted list. Figure 2 shows the document as authored, and the document as presented by two client viewers (Mosaic and Lynx [Mosaic, Lynx]).

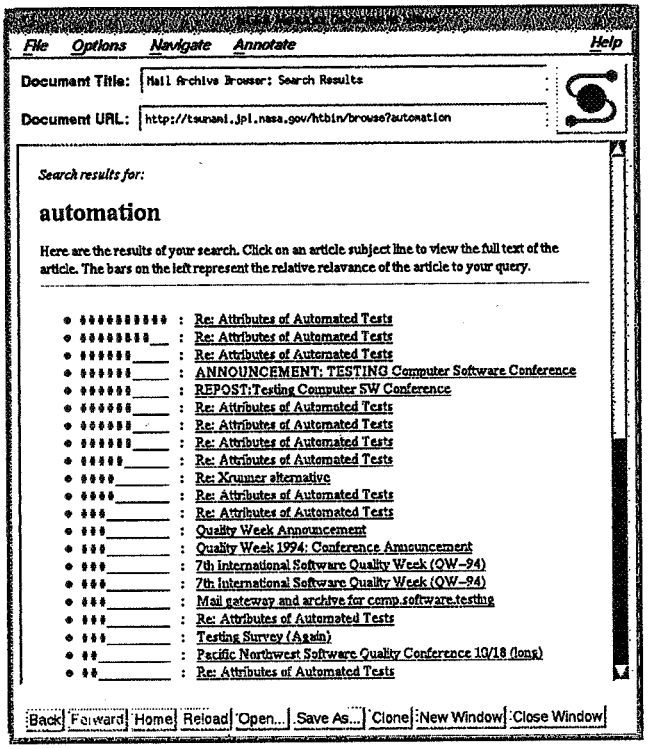


Figure 1. Result of WAIS search

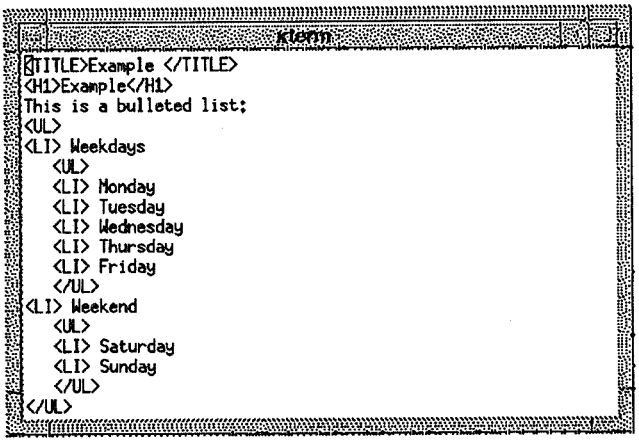


Figure 2. HTML, Mosaic and Lynx example (cont'd on next page)

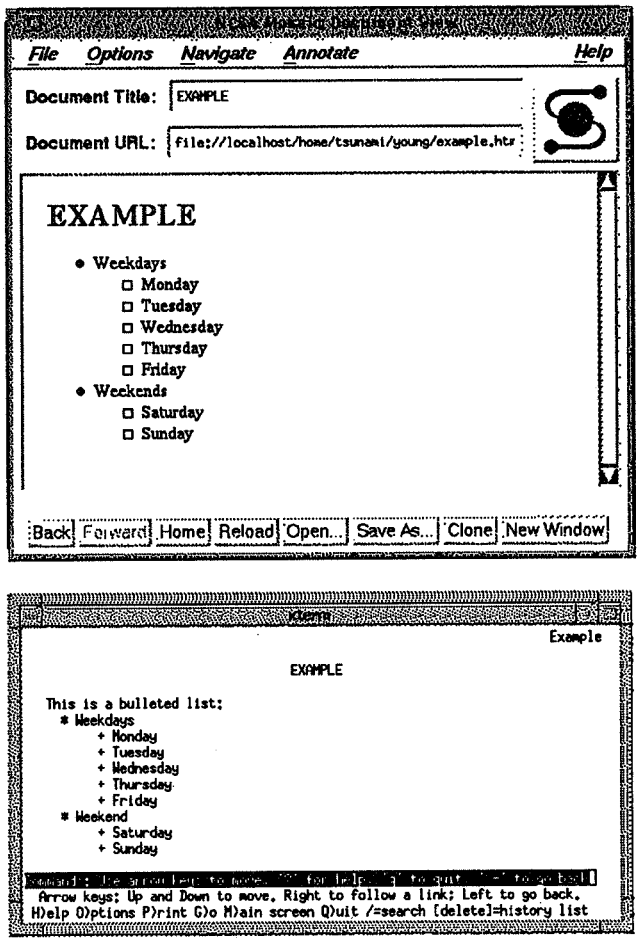


Figure 2 (Cont'd). HTML, Mosaic and Lynx example

It is important to note that the format of the presentation is determined by the client interface. The advantage of this separation of logical structure and format is that HTML clients exist for several platforms. A disadvantage, however, is that authors cannot be sure of exact placement of objects on users' screens. This is unacceptable for certain engineering and operations tasks.

The TEL prototype demonstrates the use of graphical data to resolve this problem. Graphical data can be traditional images or documents requiring a specific display format. Mosaic invokes data-specific viewing applications during the interpretation of an HTML document. For example, mission Sequence of Events (SOE) schedules and Space Flight Operations Schedule (SFOS) timelines are difficult to represent in HTML. The SFOS is a graphical timeline representation of critical information contained in the SOE. The prototype maintained a uniform user interface by launching special viewers for these documents from Mosaic. Figure 3 is the result of a query for an SOE segment.

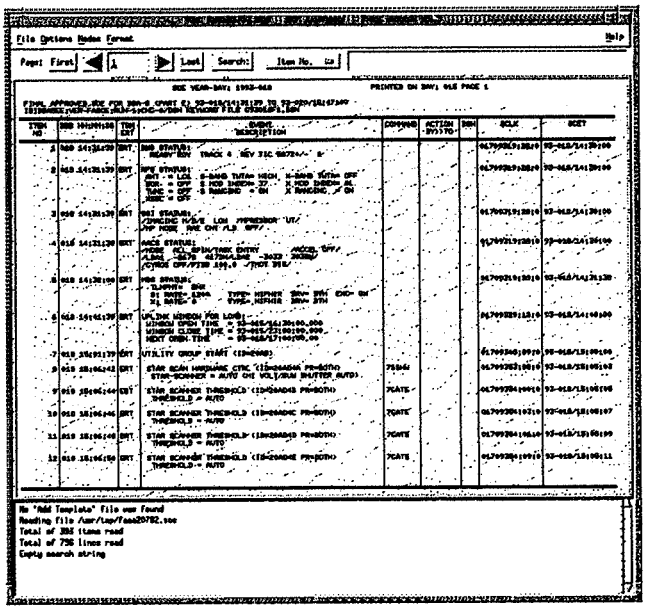


Figure 3 An SOE segment.

Finally, the prototype's most innovative aspect is the access provided to the existing Anomaly Tracking System (ATS) database of failure reports (FRs). The ATS is essential to the daily work of JPL testers. The prototype allows ATS information to be queried in a straightforward way by any combination of spacecraft, subsystem, criticality, date, and other criteria.

Previously, access to an FR database required the use of a commercial relational database interface, or telephone calls to support personnel requesting that a query be submitted. Using the capabilities provided by Mosaic it is possible to significantly simplify query formation and submission. This makes the FR database accessible to users unfamiliar or uncomfortable with relational databases. No modification to the existing ATS system was necessary.

Figure 4 is the search form as it appears using a Mosaic interface. Users compose a query by clicking buttons to choose menu items. The form in Figure 4 has been set up to choose a "listing" format of all open failure reports. The query is submitted by clicking the "generate" button. This new interface provides simple and consistent

access to users from any workstation. Users have reported a reduction in time required to access the ATS and an increase in utility of the ATS system. The result of the query is shown in Figure 5.

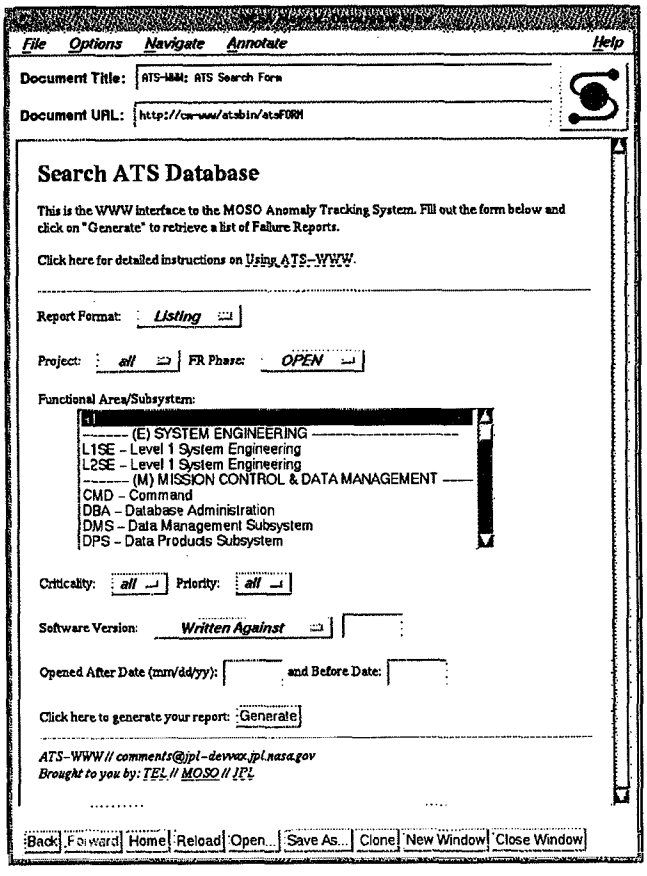


Figure 4. FR database query form

The Mosiac interface to ATS was implemented using a Common Gateway Interface (CGI) extension to a World-Wide-Web (WWW) server[2] [WWW, CGI]. CGI extensions are used to create interactive documents. Figure 6 illustrates how CGI defines the interaction between a WWW server and programs run by the server to carry out special client requests. User inputs are encoded by Mosaic as special Uniform Resource Locators (URL) and passed to the WWW server [URL]. The server invokes the CGI application and passes the user's inputs to it. The CGI application then carries out the user's request (e.g., extracts data from a database) and sends the result back to the WWW server in HTML format. Finally, the WWW server forwards the result back to the client viewer for presentation to the user.

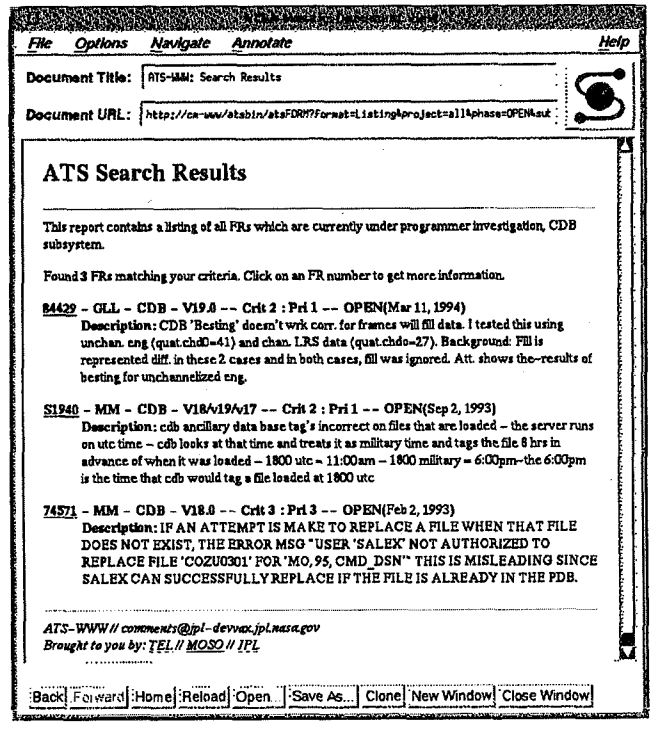


Figure 5. Result of FR database query using Mosaic interface

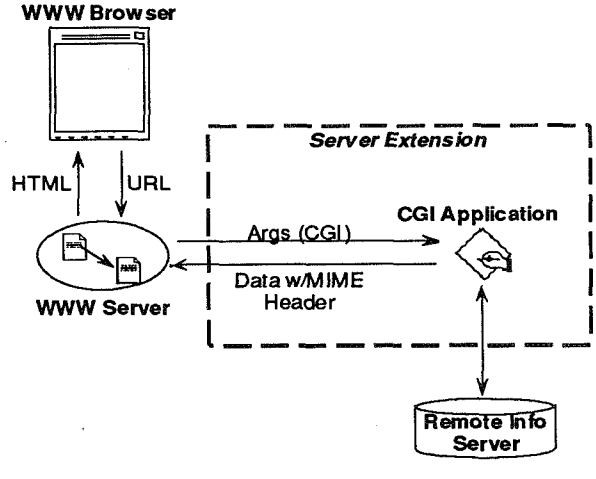


Figure 6. WWW server extensions

[2]NCSA's httpd v1.3 was used for both the prototype and delivered system

Development of the prototype

The prototype system was developed over 12 weeks by three people. It consisted of approximately 2000 pages of hypertexted hard copy documents, and 1500 lines of Perl scripts to interface with the existing ATS database front end [Wall and Schwartz].

One of the advantages of using HTML and Mosaic viewers was that potential users were able to see working prototypes quickly as development continued.

The prototype has provided a foundation for future work by demonstrating user-level integration of separate information systems and providing a uniform view of these systems across workstations.

HTML and Mosaic were chosen over other systems for several reasons. Adobe Acrobat[3] offers excellent cross-platform document browsing capabilities, but provides only rudimentary support for hyperlinks and does not support client-server interaction, making it difficult for one server to support multiple platforms over a wide area. Hyperman [Crues], developed at the Johnson Space Center and based on Adobe's PDS (Page Description Language), allows personal annotations and stronger hypertext capabilities, and will support the client-server model in the future. However, neither of these tools support "on-the-fly" document generation required for access to ATS, nor do they allow integration of user-defined viewers for unanticipated data types.

Current status

The TEL's prototype system has become a product supported by the Multimission Operations Systems Office (MOSO). The production version includes a hypertext form for submission of change request (CR) queries, as well as forms for submission and update of FRs and CRs. A larger effort is under way to convert AMMOS user documentation to HTML format, and the Cassini project is making much of its project documentation available through HTML clients.

[3]Acrobat is a trademark of Adobe Systems Incorporated.

Future work

One problem with using client user interfaces to interpret tagged hypertext documents is that clients may interpret logical organization tags in documents as suggestions rather than commands. Clients are free to display documents in idiosyncratic ways. In practice, the behavior of clients is not as anarchical as it sounds.

Because of the necessity of absolute format control in some engineering and operations documents, the TEL is continuing to evaluate extensions to HTML. In particular, HTML+ [HTML+] promises to provide increased support for mathematical symbols, tables, change bars, and floating panels (sidebars).

Second, future prototypes will allow testers to attach "personal annotations" as well as MIME (Multimedia Internet Mail Extensions) [MIME] format objects (i.e., screen dumps, core files, support documents, etc.) to FRs.

Third, the Deep Space Network (DSN) maintains a similar problem report tracking database accessed by sites worldwide. A system based on the TEL prototype and MOSO ATS product is being developed.

Summary

The TEL prototype demonstrates an integrated, consistent view of existing distributed information systems using low cost tools. In some cases, greater integration is achievable using hypertext (i.e. linking references to FRs in documents to the FRs themselves). Making information available in this way reduces delays due to information not being readily accessible when needed.

Acknowledgments

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Specifically, David Hermsen, manager of the TEL, recognized the potential of this work and supported the effort. John Louie assisted with the prototyping of the Perl interface to the ATS. Tho Le developed the ATS production system interface. Diane Miller and Debbie Tsoi-A-Sue provided logistical support.

Availability

The URL for the TEL's homepage is http://tsunami.jpl.nasa.gov/tel-home.html.

References

[CGI] The CGI specification is `http://hoohoo.ncsa.uiuc.edu/cgi`.

[Crues] E. Z. Crues. HyperMan 2.0 Enhanced Electronic Document Viewing. Presentation to KSC Mini-Workshop on Electronic Documentation, February 1994. Dr. Crues' email address is `ezcrues@gothamcity.jsc.nasa.gov`.

[HTML] The specification is `http://info.cern.ch/hypertext/WWW/MarkUp/HTML.html`. A beginner's guide is `http://www.ncsa.uiuc/demoweb/html-primer.html`.

[HTML+] The draft HTML+ document is `ftp://ds.internic.net/internet-drafts/draft-raggett-www-html-00.txt`.

[Lynx] Lynx is described in `http://www.cs.ukans.edu/about_lynx/about_lynx.html`.

[MIME] MIME is described in RFC 1341, RFC 1343, and RFC 1344, available at `ftp://ftp.internic.net/rfc`.

[Mosaic] The home page for Mosaic is `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html`.

[URL] An overview of URLs is `http://www.ncsa.uiuc.edu/demoweb/url-primer.html`. The specification is `ftp://info.cern.ch/pub/doc/url-spec.txt`.

[WAIS] A bibliography of WAIS documentation is `ftp://quake.think.com/wais/bibliography.txt`.

[Wall and Schwartz] L. Wall and R. L. Schwartz. *Programming perl*, O'Reilly & Associates Inc., 1990.

[WWW] Links to documentation about the World-Wide-Web, including a bibliography, are at `http://info.cern.ch/hypertext/WWW/TheProject.html`.

N95- 17558

11143

354216 p. 9

TOWARDS AN INTEGRAL COMPUTER ENVIRONMENT SUPPORTING SYSTEM OPERATIONS ANALYSIS AND CONCEPTUAL DESIGN

E. Barro, A. Del Bufalo, F. Rossi
VITROCISET S.p.A.
Via Salaria 1027
00138 Roma - Italia

ABSTRACT

VITROCISET has in house developed a prototype tool named System Dynamic Analysis Environment (SDAE), which aim is to support system engineering activities in the initial definition phase of a complex space system.
The SDAE goal is to provide powerful means for the definition, analysis and trade-off of operations and design concepts for the space and ground elements involved in a mission.
For this purpose SDAE implements a dedicated modelling methodology based on the integration of different modern (static and dynamic) analysis and simulation techniques.
The resulting "system model" is capable of representing all the operational, functional and behavioural aspects of the system elements which are part of a mission.
The execution of customised model simulations enables:

- the validation of selected concepts w.r.t. mission requirements;
- the in-depth investigation of mission specific operational and / or architectural aspects;
- the early assessment of performances required by the system elements to cope with mission constraints and objectives.

Due to its characteristics, SDAE is particularly tailored for non conventional or highly complex systems, which require a great analysis effort in their early definition stages.
SDAE runs under PC-Windows and is currently used by VITROCISET system engineering group.
This paper describes the SDAE main features, showing some tool output examples.

1. INTRODUCTION

Modern space systems are evolving towards higher levels of complexity in both the functional and behavioural domain. This is a natural consequence of the increasing reliability of technologies based on intelligence and automation.
Spacecraft on board autonomy levels are progressively enhanced, and more "intelligent" and sophisticated operation control and support systems are conceived and developed.
Such a context demands for a complex engineering effort in the first phases of the system life cycle, when

- the suitable identification and / or selection of mission elements,
- the definition of system functions and functional sharing between elements,
- the establishment of a mission operations concept,
- the identification of system design and performance drivers,
- the validation of system conceptual definition w.r.t. mission objectives, requirements and constraints,

imply in depth analysis and trade-off among a wide scope of interdependent technology and implementation solutions.
The selection of an optimum mission configuration and operational strategy also affects heavily elements procurement or development and utilisation risks and costs.
In parallel with the evolution of space operations conduct and support technologies, it is therefore necessary to adequately improve engineering support aids to the conceptual design of the

mission and its constituting space and ground elements.
This can be achieved through extensive use of modern computer aided modelling and simulation methods and technologies.
VITROCISET is working since some years in this field, through:

- a methodological effort based on the definition of an integral modelling methodology for a complex system, capable to suitably support different kinds of representations (operational, functional, architectural) for conceptually different systems.
  Such a methodology has been derived by exploiting commonly adopted description, analysis and simulation synthaxes (e.g. OOA, SADT, Petri Nets).
- a development effort for the integration within a unique computer environment of system description and analysis capabilities, providing in this way the user with a single point of access to the whole system information, and means for information derivation, handling, consistency check and executable simulations preparation, execution and evaluation.
- an application effort, aimed at exploiting the computer environment capabilities in the frame of concrete projects and at deriving from the application experience requirements for environment upgrades.

System definition and analysis methodology has been already presented and discussed in precedent papers of the same Authors (Ref. 3, 5).
In parallel with the methodology development and refinement, VITROCISET has developed a PC based tool named System Dynamic Analysis Environment (SDAE), which has been progressively enriched in the last years up to covering with automated support a large part of the methodology characteristics.
The System Dynamic Analysis Environment finds its natural application in the fields of system operations analysis and systems engineering, in the frame of both high level (A and pre-B phases) studies related to satellite operations and in the system definition and design phase.
Currently, SDAE supports mainly the following activities:

- mission and system requirements definition and management;
- operations modelling;
- functional static and dynamic modelling;
- behavioural modelling;
- models parametrisation with operational and performance attributes derived from mission and / or system requirements;
- executable simulation and statistical evaluation of simulation results.

2. SDAE MAIN PRINCIPLES

SDAE tool is based on a layered modelling approach, depicted in figure 1.

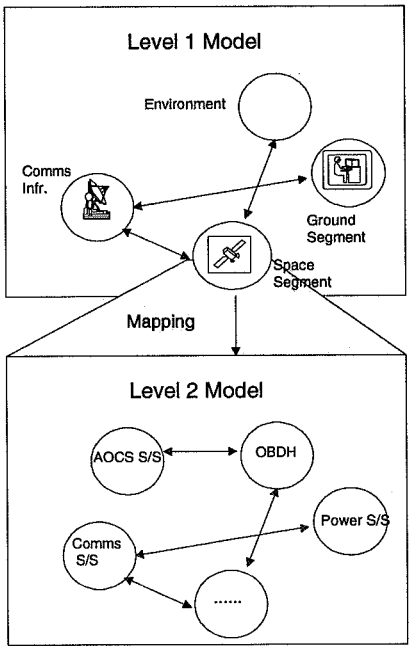


Figure 1: The layered Modelling Approach.

Each hierarchical layer is constituted by a set of models which structure and organise system information within well defined entities.
The scope and the purpose of the modelling activities vary according with the level of details of the system description.

On top layer, the entities managed by the tool are the main mission elements (physical or logical), such as the flight element(s) and its supporting ground facilities, or the spacecraft environment as well.
Entities can be functionally described as objects, in all those static and dynamic aspects which are of particular interest for the engineer in order to analyse a specific problem for the mission.
At this stage modelling supports initial mission analysis and operations concept definition activities, such as selection of mission support infrastructure, assessment of operational strategies and derivation of related design requirements and constraints.
A core modelling functionality enables the definition of dynamic relationships between objects (in terms of e.g. data exchange, events or dynamic modification of model parameters which affect objects behaviour).
Lower level models can be progressively defined for more specific analyses (e.g. command and control concept definition, budget analyses, element conceptual design and trade-off's).
The utilisation of a unique descriptive methodology at all the levels of details enables a straightforward traceability among the different modelling layers.
At bottom level, the tool can support the definition and description of end-to-end functional architecture models for the mission elements and their sub-components.
Any object at any level can be customised with characteristic parameters and reused in different contexts, even though at high level it constitutes only a partial view of the described element.
The execution of interactive simulations is therefore supported by a set of configurable library modules, including environmental models such as e.g. drag models and orbital propagators.
Simulation input parameters can be derived directly from associated requirements, as well as output parameters can be source for lower level requirements through dedicated derivation rules.

3. SDAE DESCRIPTION

SDAE tool provides the capability to build and execute dynamic operational, functional and behavioural models of a system, associating model parameters to mission or system requirements.
A high level architecture of the SDAE is provided in figure 2. Dotted lines in the figure show functionalities which are presently under development or test.

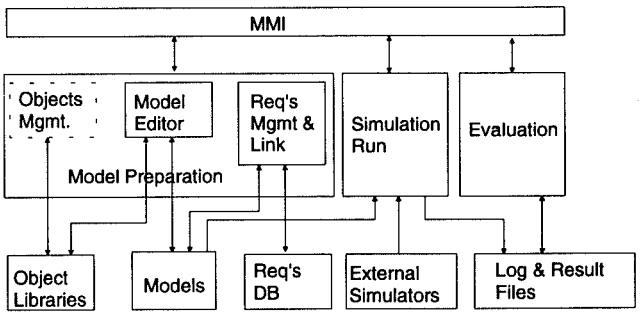


Figure 2: High Level SDAE Architecture.

The SDAE is constituted by three separate environments:
- Model Preparation;
- Simulation Run;
- Evaluation.

3.1 MODEL PREPARATION

Models are generated by means of:
- an object management facility (under development) for the static definition of basic model entities and their characterisation by means of a set of variables;
- a model editor facility for the end-to-end description of objects dynamic behaviour and relationships or interfaces;
- a requirements management and link facility for the models parametrisation with numeric parameters derived from mission or system requirements.

The model objects descriptions can be stored within object libraries and reused.
Models can also be interfaced at design time with external application specific simulation

libraries, with which they exchange data and status at run-time, providing in this way a realistic scenario for the simulation.

The **Model Editor** realises the core modelling functionality.

Such an editor is based on a Petri Nets-like synthax, and exploits a dedicated extension of Petri Nets methodology.

The editor enables the model dynamic specification through:

- a core state-transition network with deterministic and /or stochastic transitions;
- a predicates editor, which supports the definition of network predicates (conditions and actions) by means of a dedicated simulation language, and enables the model link with external simulation libraries.

The **Requirements Management and Link** facility enables the mission / system requirements handling, through:

- a requirements database editor;
- a linker between model variables and numeric requirements parameters, with possibility to specify input and output links, together with derivation rules for derived parameters;

The model preparation environment also enables the generation of ad-hoc panels for simulation monitor and control.

An example of SDAE preparation environment display output is provided in Figure 3.

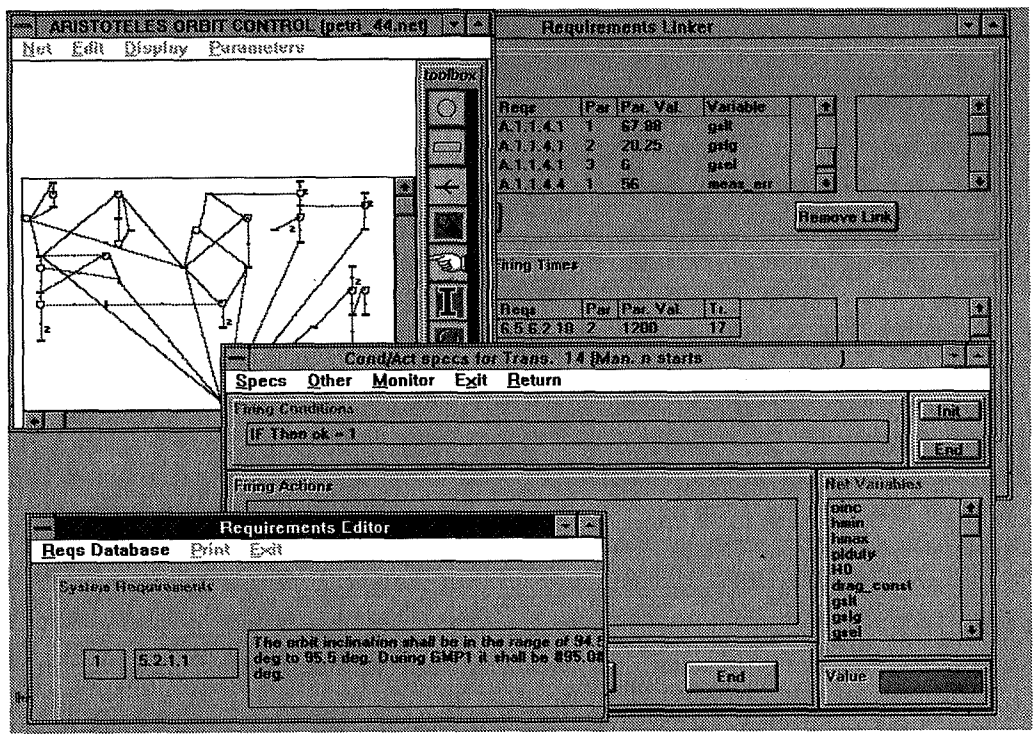


Figure 3: SDAE Model Preparation Environment.

3.2 SIMULATION RUN

Once the model has been generated, a simulation can be executed by means of the Simulation engine of the tool.

The simulation execution environment allows:

- initialisation of simulation parameters (e.g. duration, step) and variables;
- three different modes of simulation:
  - batch (the model works stand-alone with user interface);

- step by step (the model stops in case of firing conflicts in order to highlight decision branches in system behaviour);
- debugging (the user decides which transition shall fire, among those enabled, in order to experiment predefined behavioural paths);

- capability to stop, continue or restart a simulation with the same or different initial conditions;
- user interaction in batch mode, by means of monitoring and controlling the model through customised control panels defined at design time;
- simulation history log;
- on-line display of simulation statistics.

During the simulation, the run module executes the model syntax, interfacing with external simulation software.

The capability of defining firing conditions for the network transitions enables the implementation of priorities, in case the modelled process is fully deterministic, i.e. no resource conflict between concurrent functions is allowed.

The definition of transitions associated actions enables the parametrisation of network tokens, modelling in this way the availability of different kind of resources within the system.

Examples of simulation execution environment display outputs are shown in Figures 4 and 5.

The shown examples reflect different simulation and design objectives, as pertaining to different stages of system life cycle.

The application shown in Figure 4 has been developed within ESA/Dornier ARISTOTELES Phase A and Pre-B studies.

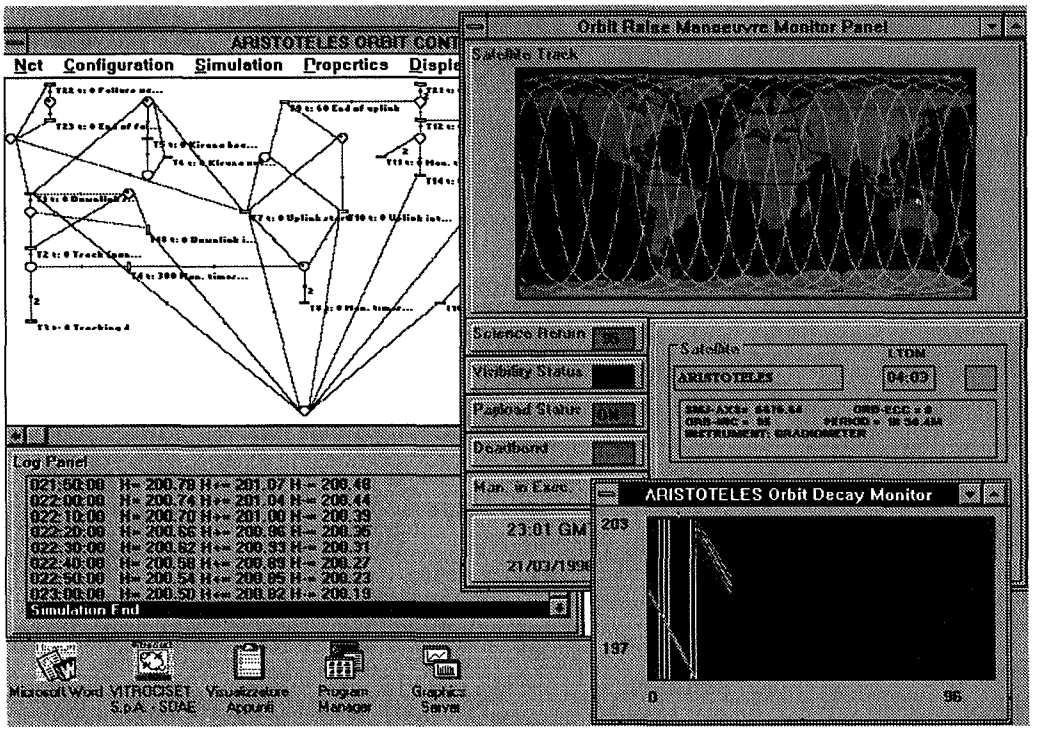


Figure 4: Simulation output example: ARISTOTELES ORM Analysis.

It constitutes the modelling of a spacecraft operational process, the Orbit Raise Manoeuvres (ORM) execution process, which involves ground, spacecraft and environmental functions.

The overall objective of the study was the definition of an optimum strategy for satellite tracking and ORM execution, identifying the impacts of the selected strategy onto the flight element and ground segment architecture.

In particular the following topics were addressed by the study:

- define the on board autonomy level, working on the flexibility of the mission;
- identify a safe orbit maintenance manoeuvre sequence;
- ensure required scientific return from the system operations viewpoint;
- identify the interrelationship of chosen coverage, link budget and memory budget with the selected operational strategy;
- validate the sequence of events in the operational scenario;
- analyse consequences of failure on the chosen design (e.g. redundancy philosophy).

Figure 4 shows:

- the model of ORM process within the Simulation Run Environment display screen;
- the ORM monitor panel, including an orbital propagator (external module) outputs and significant simulation variables monitoring;
- the Altitude display panel with an atmospheric drag model (external module) output;
- the log display of satellite contacts with Kiruna Ground Station, as computed by the orbital propagator.

The execution of the ORM process model for different initial conditions and environmental conditions (contact failures scenario) has enabled the selection and validation of an operations strategy, which satisfied all the system requirements in the defined worst case conditions.

The model has also been exploited as a breadboard of the process under study, deriving and verifying quantitative parameters determining the sensitivity of the strategy (and therefore strategy failure conditions) to the variation of any of the parameters of the model, like e.g. the spacecraft decay rate or the altitude determination errors, with respect to the reference values.

A wide number of statistical results about the process under study has been derived, as the time distribution of manoeuvres intervals and of manoeuvres size, the deadband utilisation figure, the scientific return distribution.

Finally, concrete impacts on the space and ground architecture have been identified on the basis of simulation results, especially with respect to On Board Data Handling System (in terms e.g. of definition of autonomous functions, sizing of mass memory required for manoeuvres parameters storage) and Ground Station architecture (e.g. need for a dedicated ground station, which has been derived as an "a posteriori" constraint for successful exploitation of ORM strategy).

The application shown in Figure 5 has been developed in the frame of ESA/SAT CONTROL Hermes Board Observability Breadboard (BOB) software project.

The BOB is a spacecraft simulator which models the generation and downlink of Hermes telemetry, with the scope limited to Guidance, Navigation and Piloting (GNP) functions.

The objective of the BOB is to provide a mean (breadboard) for the definition of an optimum telemetry strategy, and the verification of how this strategy copes with spacecraft observability requirements.

In this context, VITROCISET has been responsible for the definition and development of the on board Telemetry Generation Assembly simulator, which reproduces the generation of CCSDS telemetry packets on the basis of on board events and operator directives, and their delivery to Communications subsystem for downlink.

The Telemetry Generator Assembly (TGA) was designed with the SDAE simulation support.

A behavioural model of the assembly was generated and executed, in order to validate system behaviour w.r.t. specifications, to experiment different implementation solutions and to derive performance objectives for the software modules in order to cope with system requirements.

The model was able of fully reproducing the system behaviour, including partial modelling of hardware equipment (disk driver, buffers).

As an example, the model reproduced the following characteristics:

- packet generation directives acceptance and rejection policy (including input data format

and parameters check and consistency check with current packet generation status) and related timing;
- directives processing operations;
- directives scheduling policy (e.g. insertion/deletion/update of schedule items, schedule execution tasks "jumping" in case of critical delays) and related timing;
- internal synchronisation and priorities (e.g. enabling / disabling of packet playback on the basis of schedule status, blocking and non blocking operations, internal overrides);
- packet generation policy (e.g. handling of measurement variations occurred during the generation of a packet, generation policy of supercommutated packets).

The model accepted as an input a timeline of telemetry generation directives, and enabled the operator interaction by means of issuing at any time new directives for the model. The output of the model was a list of generated packets, with:
- packet generation and delivery times;
- list of included measurements and related values.

The time resolution of the simulation was chosen of 1 millisecond.

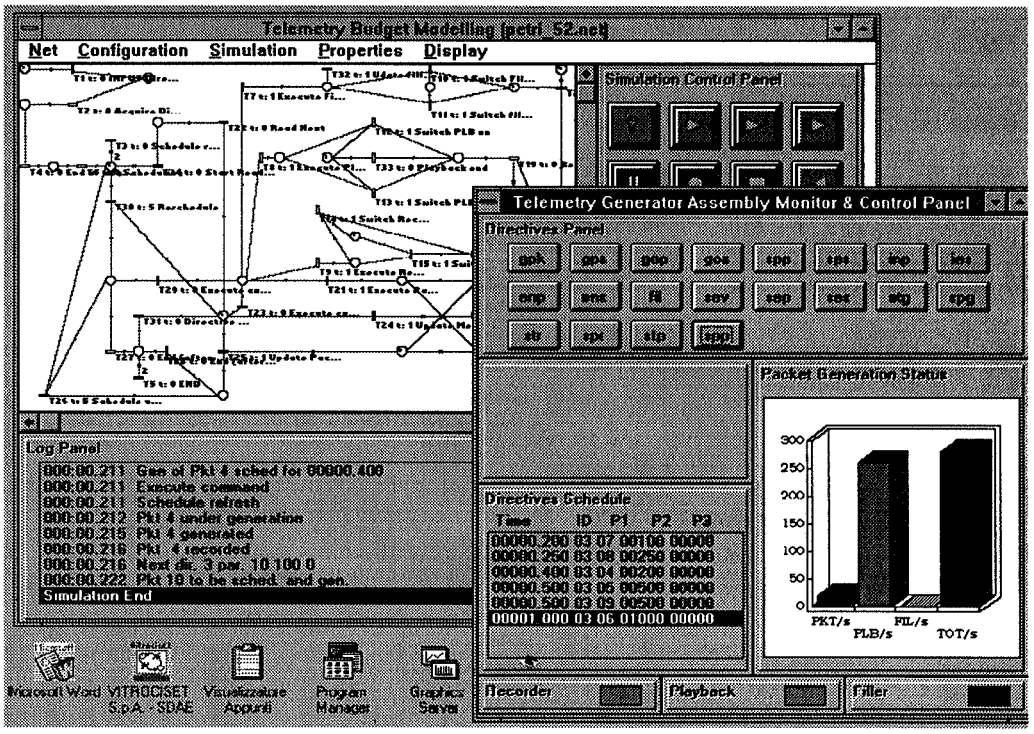


Figure 5: Simulation output example: BOB TGA Architectural Design.

Figure 5 shows:
- the process model within the Simulation Run Environment display screen;
- the test operator monitor and control panel, including:
  - directives panel for the generation of telemetry generation directives by the operator;
  - packet generation status monitoring panel;
  - current system schedule;
  - status of the main system functions.

The execution of the TGA behavioural model provided the designer with a lot of information on the system. In particular different scheduling policies and packet generation policies have been tested before selecting the one which optimised system functioning under nominal and peak load conditions.

Even though the model was at behavioural level, inferences on system performances have been

derived by setting and changing maximum allowed times for software tasks execution, and deriving in this way objectives to be pursued in single functions implementation in order to meet the overall system performances.
In depth analysis of deadlock conditions has been performed, by means of identifying and quantifying the relationship between the input data rate and the system response, which under critical conditions is characterised by a degradation in performances due to the skipping of packet generation tasks in order to avoid propagation of delay with respect to the schedule.
In addition, the system response under different modes of functioning (e.g. recorder, playback, filler activated / deactivated with a predefined rate) allowed the determination of packet generation rate achievable in the different modes, deriving in this way differentiated constraints for packet generation function.
Finally, the partial modelling of some significant time consuming hardware functions (access to disk, input/output operations) enabled the assessment of limits imposed by the hardware onto system performances.

3.3 EVALUATION.

After the simulation run, the log file is processed by an **Evaluation** environment, which computes and displays the main network statistics, i.e. for each transition:

- overall number of firings;
- minimum, average and maximum time between two successive firings.

The environment also supports the generation of customised graphical reports by means of interface with standard Windows facilities and the processing of the log file, providing statistical figures of predefined network parameters and variables (e.g. distribution of parameters values across the simulation).
An example of Evaluation Environment screen layout is provided in Figure 6, representing the ARISTOTELES ORM process model simulation statistics.

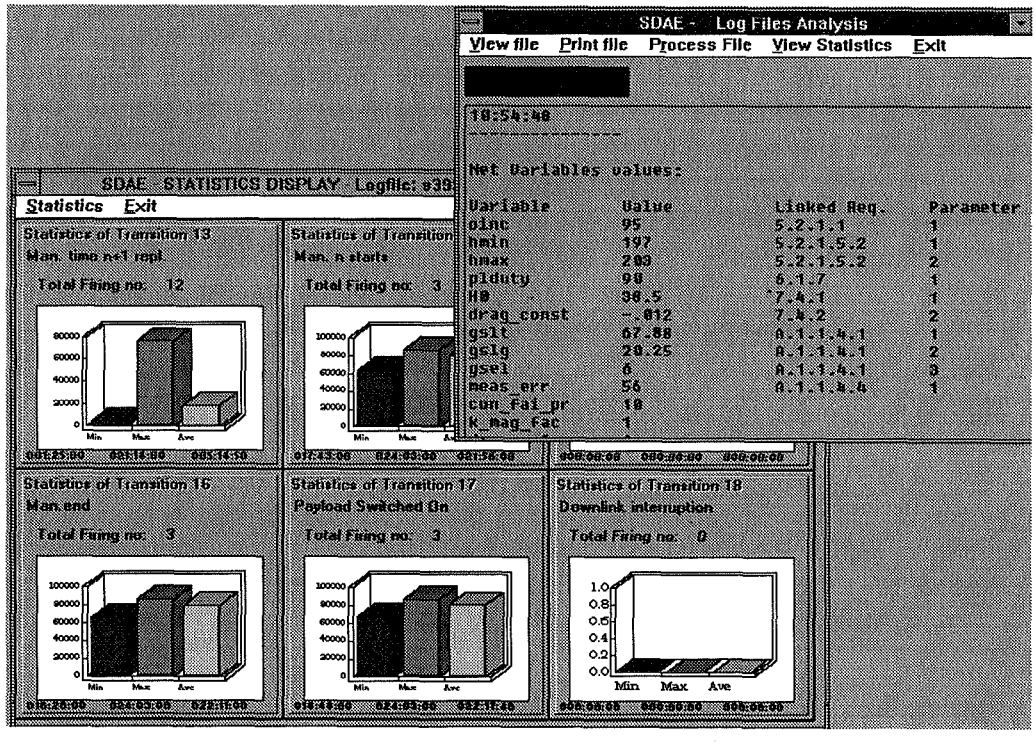


Figure 6: Evaluation Environment output example: ARISTOTELES ORM Analysis.

4. CONCLUSIONS

SDAE prototype implementation has been originated with the purpose of investigating the system engineering process of a modern space system in the first phases of its life cycle.
In particular, the ultimate objectives of the tool were:

1. to provide an efficient breadboard for testing "on the job", within limited implementation costs and effort, methodologies aimed at:
   - ensuring a harmonic and consistent growth of system information in this phase;
   - empowering system analysis and validation capabilities, especially for highly automated or non procedural systems.
2. to derive requirements for methodologies assessment and refinement, on the basis of concrete engineering needs outcoming from the tool application experience.

SDAE application has resulted to effectively support both system analysis and conceptual design, lowering the engineering effort for the execution of operations analyses and architectural trade-off's and providing, by means of simulation, significant support to operations and system concepts validation capabilities.
In particular the following characteristics of the prototype have been found of particular interest, especially in comparison with engineering tools available on the market:

- the flexibility of modelling methodology, which enables the easy generation and maintenance of "on purpose" models, without constraining the engineer to rigorous top-down approaches, but at the same time providing capabilities for system information consistency keeping;
- the adequacy of modelling and simulation tools to non procedural, event drive systems;
- the reusability of model objects and simulation modules;
- the "live dialog" capability of system models with mission and system requirements parameters through numeric data exchange and derivation rules, which highly enhance ability to manage, control and validate system information.

Those positive outcomes suggested the prosecution of the internally funded SDAE prototyping activity, which currently is being performed in the direction of both:

- improvement of tool modelling powerfulness and engineering support scope;
- increase of tool application experience, through the investigation of new application areas, such as communications and ground data control and distribution systems.

REFERENCES

1. Agerwala T. 1979. Putting Petri Nets to work. Computer, Dec. '89, 85-94
2. CISET. ARISTOTELES Phase Pre-B Study: Satellite Autonomy. Ref. ARI-TN-CI-001, Issue 1.1, April 3rd, 1992
3. E. Barro & F. Rossi. An Application of Timed Petri Nets to Operations Analysis: the ARISTOTELES Autonomy Concept. In *Proc. ESA Symp. "Ground Data Systems for Spacecraft Control"*, Darmstadt, FRG, ESA SP-308, 317-322.
4. SAT CONTROL. BOB Software and Hardware Architecture Description. H-NT-01210-0286-SATC, Issue V0, September 8th, 1992.
5. E. Barro, A. Del Bufalo & F. Rossi. Operational Characterisation of Requirements and Early Validation Environment for High Demanding Space Systems. In *Proc. NASA 2nd Symp. "Ground Data Systems for Space Mission Operations"*, Pasadena, California, USA, November 16-20 1992, 845-850.

N95-17559

111144

354219

SEQ_POINTER: NEXT GENERATION, PLANETARY SPACECRAFT REMOTE SENSING SCIENCE OBSERVATION DESIGN TOOL

Jeffrey S. Boyer[1]
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109
USA
Phone: (818) 354-3944 Fax: (818) 393-5074

ABSTRACT

Since Mariner, NASA-JPL planetary missions have been supported by ground software to plan and design remote sensing science observations. The software used by the science and sequence designers to plan and design observations has evolved with mission and technological advances. The original program, PEGASIS (Mariners 4, 6, and 7), was re-engineered as POGASIS (Mariner 9, Viking, and Mariner 10), and again later as POINTER[2] (Voyager and Galileo). Each of these programs were developed under technological, political, and fiscal constraints which limited their adaptability to other missions and spacecraft designs.

Implementation of a multi-mission tool, SEQ_POINTER, under the auspices of the JPL Multi-mission Operations Systems Office (MOSO) is in progress. This version has been designed to address the limitations experienced on previous versions as they were being adapted to a new mission and spacecraft. The tool has been modularly designed with subroutine interface structures to support interchangeable celestial body and spacecraft definition models. The computational and graphics modules have also been designed to interface with data collected from previous spacecraft, or on-going observations, which describe the surface of each target body. These enhancements make SEQ_POINTER a candidate for low-cost mission usage, when a remote sensing science observation design capability is required.

The current and planned capabilities of the tool will be discussed. The presentation will also include a 5-10 minute video presentation demonstrating the capabilities of a proto-Cassini Project version that was adapted to test the tool.

The work described in this abstract was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration.

Keywords: remote sensing science observation, adaptable tool, interchangeable models, digital terrain map-defined celestial body

INTRODUCTION

POINTER provides functionality analogous to a professional photographer's process of preparing for and taking photographs. POINTER supports this process for a remote robotic photographer that has no control over the environment where it has been sent to gather images and other data of the surrounding phenomenon. The functions which are similar to the photographer's process define the foundation of POINTER. These foundation functions are listed and illustrated in Figure 1. In SEQ_POINTER, the functions have been designed and implemented for multiple missions. The mission specific capabilities are incorporated via a process called adaptation.

---

1. Cognizant Development Engineer (CDE)
2. Planetary Observation INstrument Targeting and Encounter Reconnaissance

PAGE 914 INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

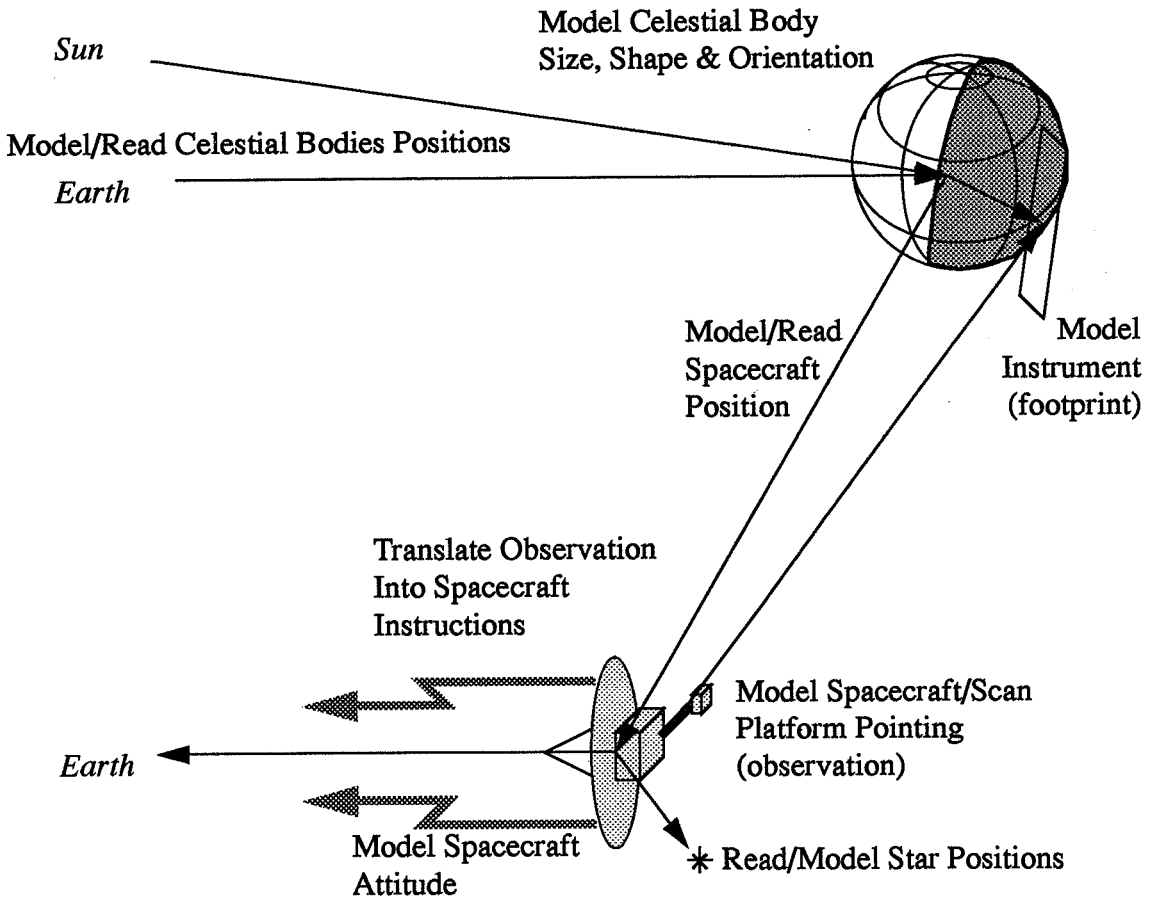


Figure 1. POINTER Foundation Functions

CURRENT CAPABILITIES

The foundation functions are augmented by capabilities which allow SEQ_POINTER to fit within AMMOS[3], the multi-mission operations support system being developed by MOSO. The tool capabilities with respect to AMMOS are illustrated in Figure 2. The primary capability is the interface with the Sequence file. The Sequence file contains spacecraft instructions and ground software directions in the form of requests. Requests perform remote sensing as well as fields and particles science observations and engineering activities during mission operations. The remaining capabilities are the interfaces with the Spacecraft & Celestial Body Ephemerides file and the Spacecraft Clock file. The Spacecraft & Celestial Body Ephemerides file(s) lump together spacecraft, planetary, and satellite ephemerides (currently NAIF[4]) and star catalog(s). The Spacecraft Clock file provides spacecraft clock adjustment data referenced by the tool.

The upper portion of Figure 2 illustrates the primary SEQ_POINTER operator displays, the operator interface and observation design depiction. The operator interface (left) is an X Window System/Motif application. It provides the operator with capabilities to manipulate observation design instructions and to perform a simulation of spacecraft execution instructions which are graphically depicted (on the right by the Project module). Figure 3 contains images illustrating sample menus (top) and a sequence component (bottom) from the operator interface. The observation

3. Advanced Multi-Mission Operations System
4. Navigation Ancillary Information Facility

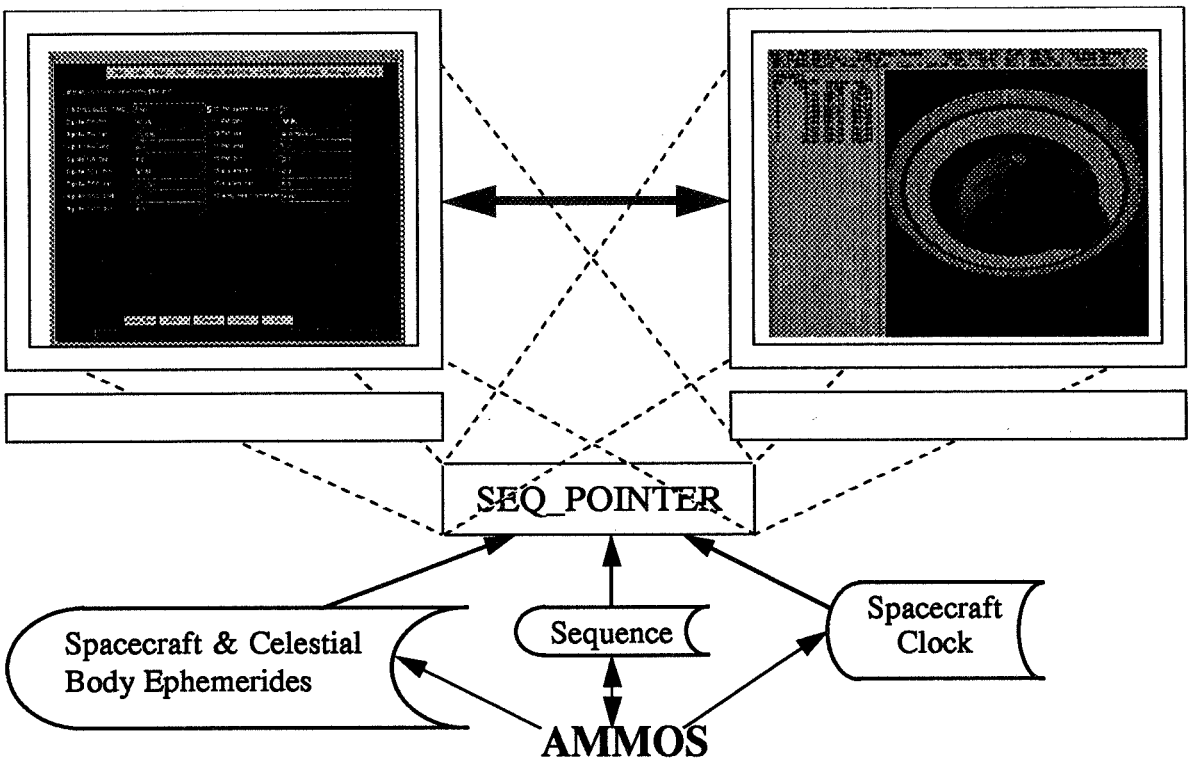


Figure 2. SEQ_POINTER in the MOSO AMMOS

graphically depicted is a perspective projection of Saturn with events (footprints) from a Cassini Project instrument. Figure 4 contains a sample image of the same observation from a different vantage point than depicted in Figure 2. The implemented module in the Project module family is an X-Window System/Motif/PHIGS+[5] application. The depiction of the target body is data driven, based-on a PHIGS data structure that models the surface of the body. The data structure can be derived from a variety of sources: an oblate-spheroid body shape algorithm or the same algorithm with an electronic version of a USGS[6] Albedo Image file. The Saturnian rings are modeled in the same fashion. However, the data structure for the rings is created at run-time from ring system constants read by the tool.

ARCHITECTURE

To facilitate a mission adaptable tool, SEQ_POINTER has been organized around the concepts of modular executable programs (module families) and interchangeable models. The tool comprises three module family groups: infrastructure programs, observation design utility programs, and observation activity programs. The groups and some of the constituent module families are illustrated in Figure 5. The infrastructure group consists of module families which contribute the underlying data flow architecture for the tool: Operator Interface, Activity Design, Modeler, Position, Project, Present, and Targeting Update. A description of each module appears in Table 1.

The design utility and observation activity program groups consist of activity and command modules, the sequence components of an observation. The design utility program group currently consists of two module families, Solar System Body/Surface Point Trajectory and Stellar Position. These modules are used to produce geometric and photometric data the operator analyzes for

5. Programmer's Hierarchical Interactive Graphics System (ANSI-Computer Graphics)
6. United States Geological Survey

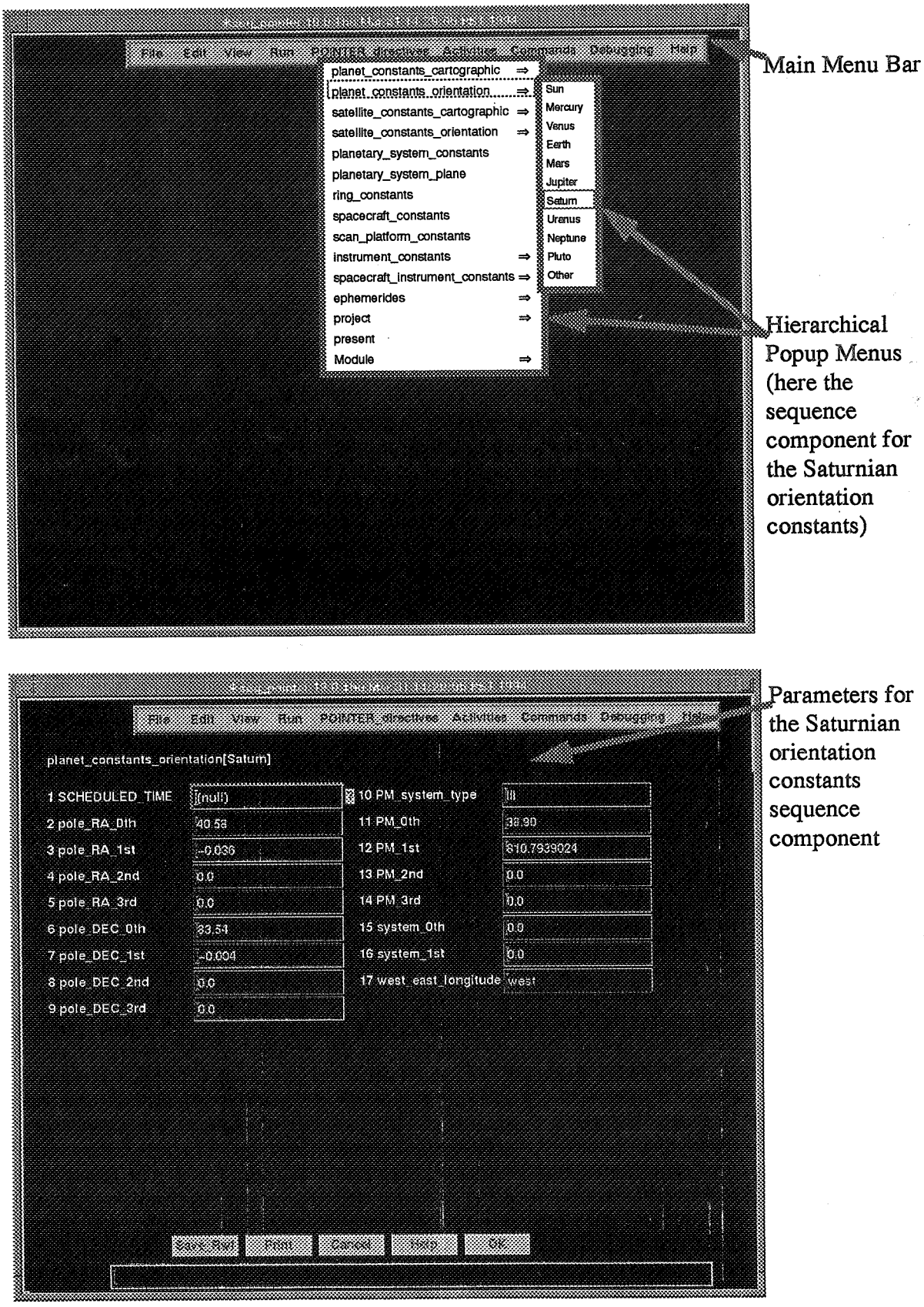


Figure 3. Operator Interface Module - Sample Windows

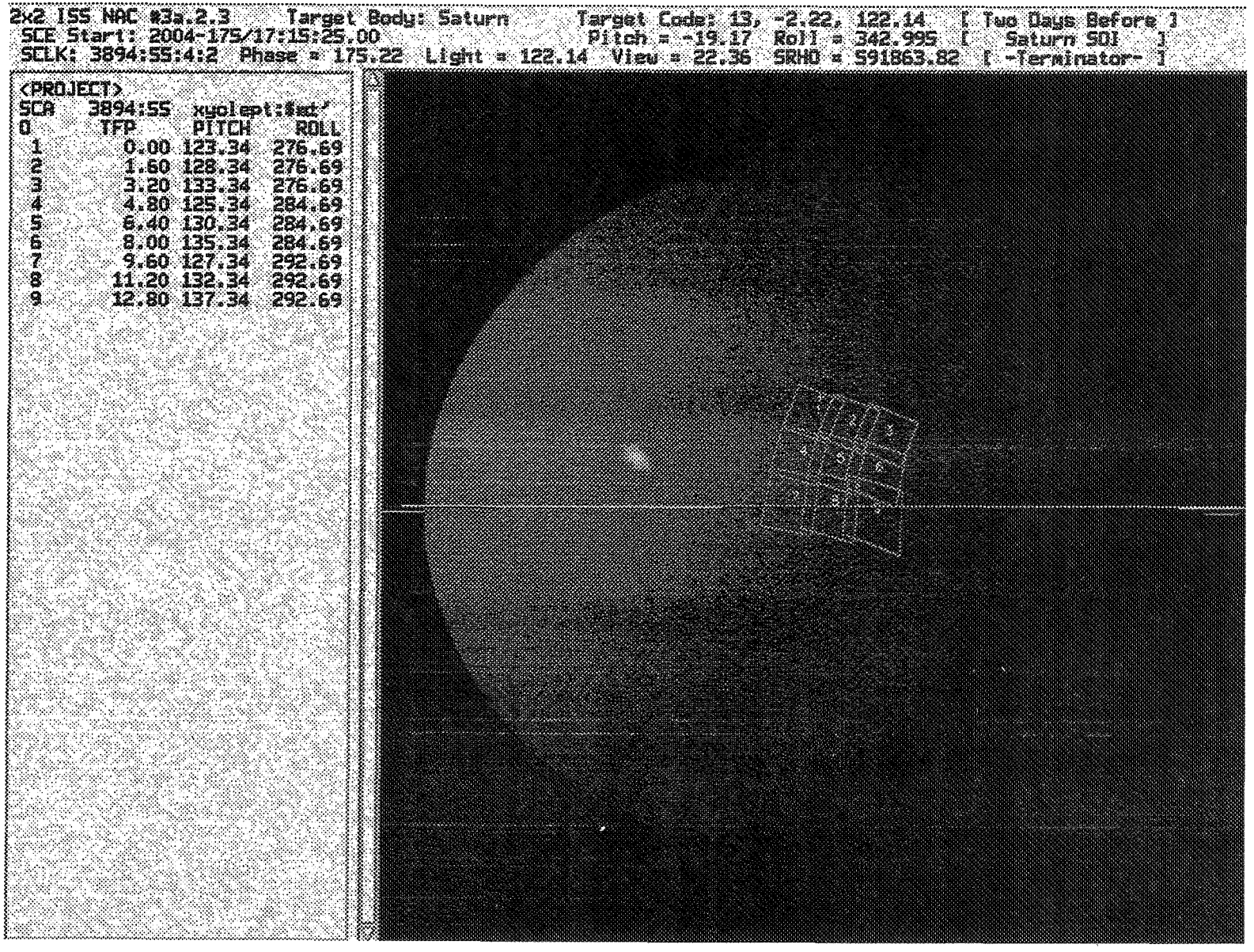


Figure 4. Project Module - Sample Window of an Observation Design Depiction

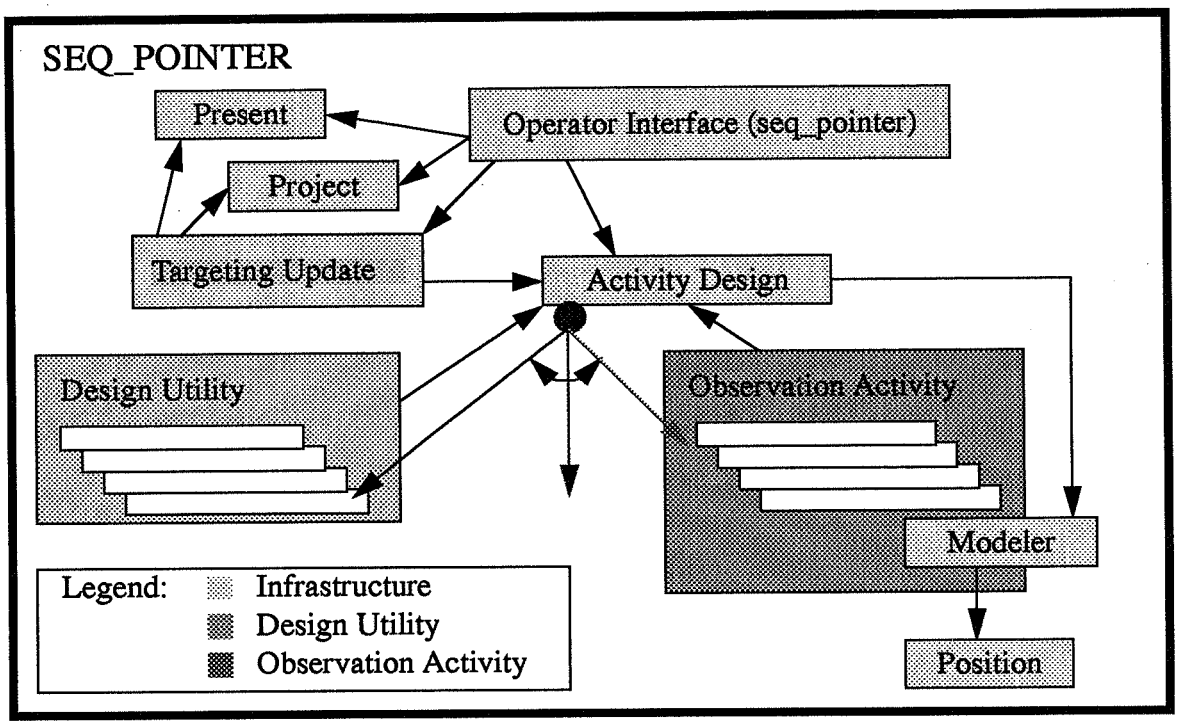


Figure 5. SEQ_POINTER Module Family Architecture

Module Family	Functional Description
Operator Interface	interactive operator and sequence file-request interface
Activity Design	processes observation activities through module families "expanding" activities to the resulting commands
Modeler	calculates spacecraft/scan platform and instrument(s) events from the commands resulting from expansion of the mission dependent activity modules and formats the event data for output
Position	calculates celestial body and spacecraft position data from internal and operator-supplied data or external ephemeris file(s)
Project	graphically depicts the observation events (footprints)
Present	reads the output event data file loading a Lotus 1-2-3 spreadsheet where charts illustrating event data can be output
Targeting Update	batch sequence file processor for updating all observations in the sequence for the latest ephemeris data

Table 1. SEQ_POINTER Infrastructure Module Family Descriptions

designing desired observations. The observation activity program group consists of the module families for all levels of sequence components. The sequence components are expanded to commands and later modeled as the events of an observation.

The tool can be adapted to a new spacecraft because the architecture segregates the mission and spacecraft dependencies. Adaptation is designed into the tool through function modularization and the concept of interchangeable models. The segregation of mission and spacecraft dependencies into independent and dependent module families is illustrated in Figure 6. The independent

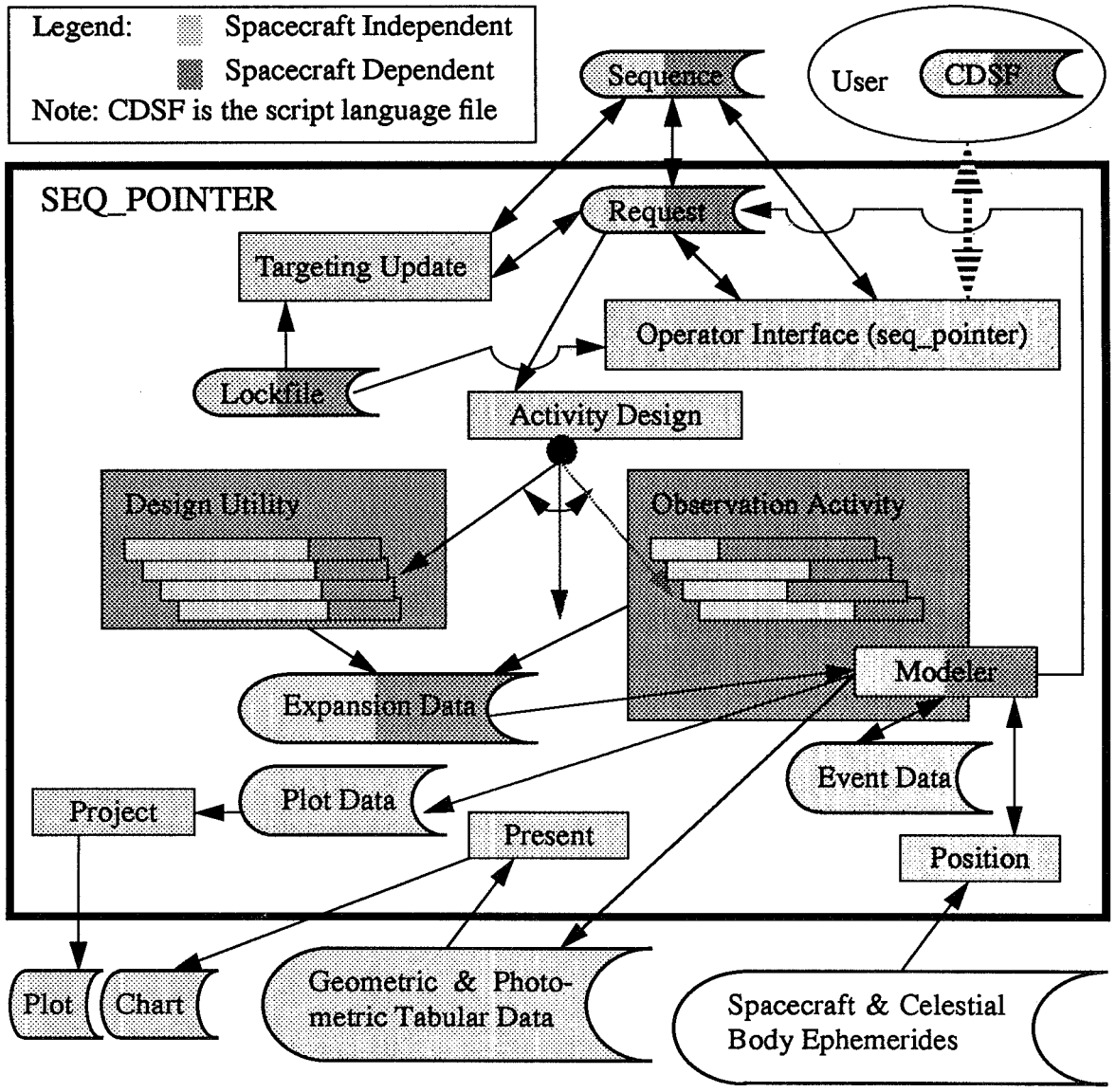


Figure 6. SEQ_POINTER Module-Data Flow

module families read dependent data file(s) to incorporate mission and spacecraft information.

Illustrating interchangeable models, the dependent module families in the design utility and observation activity groups contain both independent and dependent sections. These modules are designed around generic drivers which call plug-in models written in C language functions. The calling and return interfaces are defined for each model family. The model family instance contains or retrieves any model-unique data necessary to calculate the return data.

For example, the interfaces for the celestial body position model family are: as input, the reference (i.e., Sun) and subject (i.e., Saturn) body identifications and the time of the position and, as

output, the position and velocity vectors of the subject body relative to the reference body. The interfaces are the same whether the ephemeris data is interpreted from a conic element set, NAIF ephemeris data, or Navigation Team data. Each model instance gathers the data necessary to define the vector set at the input time. For the conic element model, it calculates the vector set from the conic element set and the necessary celestial body constants. For the NAIF or Navigation models, it interpolates the vector set using the ephemeris file readers.

ADAPTATION

Adaptation of SEQ_POINTER for a mission and spacecraft is performed manually. An adaptation utility program provided by another AMMOS tool is planned to be updated, enhanced, and delivered in the future. The following adaptation steps are performed after capability definition to create the mission and spacecraft module suite for the mission specific version of the tool:

1) identification of the necessary models and modules for the mission to be adapted,

2) identification of which existing mission independent models in the model families library satisfactorily provide the necessary capabilities,

3) modification of existing mission independent models in the model families library which must be altered to provide the necessary capabilities,

4) design and creation of new models which must be added to complete provision of the necessary capabilities,

5) design and creation of the sequence components which define spacecraft instructions and translation of the components into a SEQ_POINTER specific file format (Lockfile), and

6) compilation of the mission version models and modules to create the executable module suite.

FUTURE CAPABILITIES

Enhancements to SEQ_POINTER consist of items which were not incorporated during previous development cycles due to technological or resource inadequacies, and items which result from evolution of the mission operations concept. The changes are taking the operations concept from a centralized system using experienced MOS operators to a distributed system where the primary users are scientists and their representatives.

Additions to address the changing environment include enhancements to make the tool more usable by a broader user population and closer association with the spacecraft flight software operation algorithms. Development of a user interface which provides direct graphical manipulation of observation events which are reverse-translated into spacecraft instructions has been proposed. One delayed capability would allow observation design with an irregularly shaped celestial body (e.g., asteroid). A new body surface family model would be developed to access a celestial body digital terrain map for instrument footprint calculations. Also, a new PHIGS data structure translation utility would be included which reads the digital terrain map and produces the data that is used to graphically depict the celestial body with the observation instrument footprint events.

REFERENCE

W. I. McLaughlin, M.J. Deutsch, L. J. Miller, D. M.Wolff, and S. J. Zawacki (6-9 January 1992). Cost, Capability, and Risk for Planetary Operations. AIAA-92-0600, *AIAA 30th Aerospace Sciences Meeting*, Reno, NV.

Acknowledgements - While the author has sole responsibility for the contents of this paper, the development of SEQ_POINTER was a team effort. I wish to extend my thanks to the following personnel who contributed their talents and energies to the implementation of SEQ_POINTER, Johnny E. Freeman and Lawrence N. Seeley of JPL and Shawn M. Veloso and Steve Wong of TELOS.

N95-17560

111145

354222

P-6

KNOWLEDGE-BASED CRITIQUING OF GRAPHICAL USER INTERFACES WITH CHIMES*

Jianping Jiang
Elizabeth D. Murphy
Leslie E. Carter

CTA INCORPORATED
6116 Executive Blvd. Suite 800
Rockville, MD 20852, USA

Walter F. Truszkowski

NASA-Goddard Space Flight Center
Code 522.3
Greenbelt, MD 20771, USA

ABSTRACT

CHIMES is a critiquing tool that automates the process of checking graphical user interface (GUI) designs for compliance with human factors design guidelines and toolkit style guides. The current prototype identifies instances of non-compliance and presents problem statements, advice, and tips to the GUI designer. Changes requested by the designer are made automatically, and the revised GUI is re-evaluated. A case study conducted at NASA-Goddard showed that CHIMES has the potential for dramatically reducing the time formerly spent in hands-on consistency checking. Capabilities recently added to CHIMES include exception handling and rule building. CHIMES is intended for use prior to usability testing as a means, for example, of catching and correcting syntactic inconsistencies in a large user interface.

1. INTRODUCTION

With continuing support from the National Aeronautics and Space Administration (NASA, Code O), the evolution of the CHIMES methodology and toolset has taken place in a series of research and prototyping cycles. The goal has always been to improve the usability of user interfaces developed at the NASA-Goddard Space Flight Center (GSFC) by providing user-interface designers with an automated design-evaluation capability. Recent prototypes have focused on implementing the CHIMES concept of knowledge-based compliance checking.

*For further information contact: Walter F. Truszkowski, Code 522.3, NASA-Goddard Space Flight Center, Greenbelt, MD 20771 U.S.A. (301)286-8821, FAX: (301)286-1768, Email: truszkow@kong.gsfc.nasa.gov.

Available user-interface design software provides designers with many useful capabilities, with the notable exception of any capability to evaluate the "look and feel" of a graphical user interface. Such interfaces are often evaluated for compliance with human factors guidelines or corporate style-guide requirements. Evaluation is typically done by time-consuming, manual review and usability testing. Taking steps to speed up the evaluation process, the present CHIMES prototype is capable of evaluating the look of single and multiple display screens that include alphanumerics, color, and graphics. The full CHIMES concept encompasses rule-based evaluation of user-interface behavior.

CHIMES is intended for use by GUI designers prior to formal usability testing, as a means of cleaning up a GUI and improving consistency from screen to screen. Rules in the knowledge base critique the design, and an advice generator offers advice, warnings, and tips to the designer. Explication of the CHIMES knowledge base and critiquing process is the primary purpose of this paper.

2. OVERVIEW OF CHIMES DATA FLOW

Figure 1 provides a conceptual overview of the flow of data during a CHIMES evaluation. Moving from left to right on the figure, the resource file representing a GUI design is acquired by CHIMES and transformed into an intermediate representation, which is transferred to the knowledge base. The acquired GUI design is then submitted to analysis and evaluation by the user- selected rule set. Products of the analysis include problem statements ("critiques"), advice, and suggested modifications. User-selected modifications

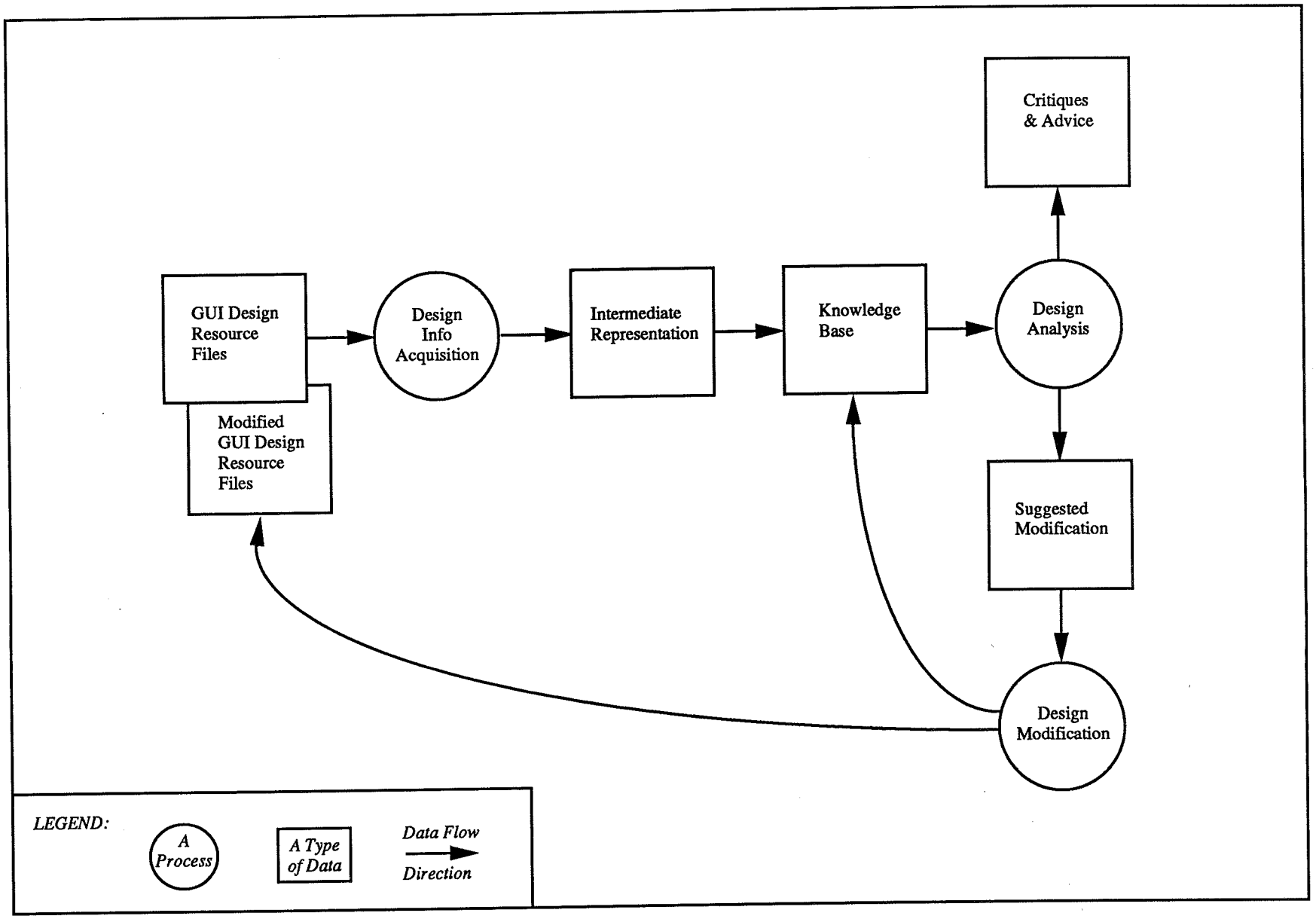


Figure 1: CHIMES Data Flow

are made automatically by CHIMES and sent to the knowledge base for re-evaluation. The resource file representing the GUI design is also automatically updated.

The remainder of the paper focuses on the contents of the knowledge base, describes the critiquing process, presents a case study, and discusses plans for enhancing CHIMES.

3. KNOWLEDGE BASE

The knowledge base stores a representation of the design to be evaluated as well as the rules that encode the heuristics for design evaluation. Each rule in the CHIMES knowledge base can be considered a critic[1]. Key components of the knowledge base include the qualitative and quantitative heuristics for evaluating the graphical design and use of color in a single display screen and, for multiple panels, heuristics on design consistency. The knowledge base is implemented in CLIPS[2].

Graphical Display Heuristics. CHIMES uses guidelines from the OSF/Motif[1] Level One Certification Checklist[8] and from the human factors literature to evaluate the "look" of single and multiple display panels. The CHIMES approach allows compliance checking of requirements and guidelines not included in the OSF/Motif defaults. For example, the number of type sizes and number of fonts per screen, as well as text justification and use of highlighting, can be checked for compliance with human factors recommendations.

Color Heuristics. The key human factors recommendation on color is that it should be used for functional purposes, not simply to decorate the screen. Functional purposes include attracting attention to critical data objects, communicating organization, indicating status, and establishing relationships between distant items[6]. To assist GUI designers in the effective use of color, CHIMES not only suggests appropriate colors but also incorporates its suggestions with the designer's functional purposes for using color and provides remedies for misuse of color.

Color heuristics implemented in the most recent prototype permit CHIMES to evaluate the consistency of color usage across multiple panels. The tool checks the consistency of both foreground and background colors; offers alternatives to the original color combination; allows the designer to preview different color combinations; and permits automatic modification of colors when the user finishes making changes.

The following are a few of the color heuristics applied in a CHIMES evaluation[5]:

- Pale foreground colors should not be used on a very bright saturated green background because of the resulting very low contrast.
- The same background color should be used for both a panel and its items unless there is a functional, user-task related reason for using different colors.
- Some background colors are not recommended for use with certain foreground colors because of the resulting color interference.

These heuristics are implemented in dozens of highly specific rules. Once the detected colors have been evaluated, CHIMES gives specific advice to improve color contrast and legibility.

Consistency-Checking Heuristics. Consistency is one of the primary human factors principles of screen design. Consistency of object location and screen behavior supports the end user's development of expectations about where to find common controls and of how the GUI will respond to user input. In general, an interface that reliably meets end-user expectations supports more efficient human performance as compared to an interface that violates end-user expectations.

As a basis for checking the internal design consistency of multiple panels or screens, the CHIMES knowledge base contains a set of rules on which there is general agreement in the human factors literature. When departures from consistency are warranted in the context of user's tasks[3], CHIMES is capable of handling exceptions.

The following are a few of the consistency-checking heuristics implemented in the CHIMES knowledge base[5]:

- The typographic elements of data items which serve the same type of function in a design are consistent within and across panels, unless there is a functional or

[1] Motif is a trademark of the Open Software Foundation, Inc.

user-task related reason for using different typographic elements.

- The background color of panels in a design is consistent across panels, unless there is a functional or user-task related reason for using different colors.
- The shadowing of pushbuttons is consistent within and across panels unless there is a functional or user-task related reason for using different shadowing thicknesses.

Although checking the consistency of location of displayed objects presents difficult technical problems, CHIMES is capable of checking the placement of the menubar. The current criterion for menubar placement is that recommended by the OSF/Motif guidelines[8]: "at the top edge of the application, just below the title area of the window frame." In the full CHIMES concept, other GUI style guides can be encoded as sets of rules in the knowledge base and applied upon user selection.

4. CRITIQUING PROCESS

The CHIMES heuristics are represented as CLIPS rules. A CLIPS rule has two parts: a conditional part and an action part. The conditional part describes the CLIPS data-memory configuration for which the rule is appropriate. (The GUI design to be evaluated is represented as facts in the CLIPS data memory.) The action part of a rule specifies the instructions to be executed when the conditional part of the rule is satisfied.

The CLIPS inference engine is the executor that determines which heuristics should be used by selecting and then executing the appropriate rule. Three steps are involved in selecting and executing rules: 1) match rules; 2) select-rules; and 3) execute rules. In the first step, match-rules, the inference engine finds all of the rules that are satisfied by the current contents of data memory according to the inference engine's comparison algorithms. The matched rules are potential candidates for execution. The second step, select-rules, applies some selection strategy to determine which rules will actually be executed. The last step, execute-rules, fires the rules previously selected.

Using the CLIPS inference engine and representing the GUI design as CLIPS facts allows the representation of heuristics as rules to match specific design patterns. For example, the rule "check-background-color-accord-pnl" represents a way to check item background inconsistency. Once the heuristics are modeled as rules, the CLIPS inference engine uses the rules to critique the GUI design that has been acquired by CHIMES.

4. CASE STUDY

As a preliminary test of CHIMES' ability to detect human factors problems in a user-interface design, we applied CHIMES to a real- world software application known as the Request Oriented Scheduling Engine (ROSE). Developed by NASA-Goddard, ROSE was designed to meet the needs of mission planners and spacecraft operators in a satellite ground-control environment[10].

The evaluation of the ROSE user interface was designed to meet two goals: 1) to identify human factors issues in need of resolution by the ROSE developers; and 2) to study how CHIMES can assist a GUI designer in catching and correcting human factors problems. For comparative purposes. we conducted both a CHIMES evaluation and a heuristic (manual) evaluation[4].

CHIMES Evaluation of the ROSE User Interface. The CHIMES evaluation took less than 10 minutes and detected three problems related to the use of fonts and typographic elements. ROSE used more than the three fonts permitted by a conservative rule in the CHIMES knowledge base. Contrary to the convention of using normal style fonts for menu options, ROSE used an italic font for options in pull-down menus. This use of italics made ROSE inconsistent with other OSF/Motif applications. CHIMES also detected typographic inconsistencies across widgets in ROSE. Several labels for the same kind of button had been implemented in mixed case, while others were in all upper case.

Heuristic Evaluation of the ROSE User Interface Three evaluators conducted the heuristic evaluation. (Two were human factors professionals who specialize in user-interface design; the third was an experienced designer of GUIs.) They spent a total of 12 person hours reviewing the ROSE documentation and on-line demonstrations. The heuristic evaluation found additional problems that CHIMES was not able to detect because of current limitations in its knowledge base.

To detect some of the problems found by the evaluators, CHIMES would need knowledge of user-

interface behavior. For example, any attempt to access the ROSE help facility caused the system to crash because this facility had not yet been implemented, although a help icon was displayed on some screens. CHIMES did not detect this problem because its current knowledge base encompasses only the look, but not the behavior of buttons. The full CHIMES concept includes evaluation of user-interface behavior.

The human evaluators found problems in screen layout that CHIMES was not able to detect. In some instances, interface elements were not grouped to aid the user's understanding of their interrelationships. Further, the heuristic evaluation found that certain panel overlays obscured useful information. To detect problems of this kind, CHIMES would need semantic capabilities beyond its current scope. For example, CHIMES would need knowledge of user goals and information requirements in order to suggest alternative layouts.

A particularly difficult issue for an automated evaluation is the absence of information that should be, but is not, displayed. For example, the human evaluators noted a general lack of user guidance (i.e., instructions displayed on the screen to aid the user in navigating through the ROSE user interface). Fairly sophisticated capabilities would be needed for CHIMES to detect the absence of user guidance or other missing information.

Similarly, advanced semantic capabilities would be needed to detect redundant information. The heuristic evaluation found, for example, a redundancy in panel titles, and the evaluator recommended simplifying the user interface by removing the redundancy.

Problems of appropriate widget selection, identified by a human evaluator, pose a significant challenge to CHIMES or any automated user-interface evaluation tool. For example, five pull-down menus were lined up horizontally to perform a task that should be performed by a menubar. Although CHIMES can detect the misplacement of a menubar, it cannot currently assess the appropriateness of the widgets selected by the user-interface designer.

As highlighted in the case study, the capabilities and limitations of CHIMES make it a useful tool to aid the user-interface designer, but not one that will replace usability testing. In the realm of user-interface syntax, CHIMES can reliably detect both inconsistent design elements and non-compliance with style guidelines. With syntactic issues cleared up prior to usability testing, such testing can then concentrate on semantic issues that affect end-user performance and satisfaction.

5. CURRENT AND FUTURE DIRECTIONS

The existing CHIMES prototype reads and evaluates GUIs created in TAE Plus[9]. Although TAE Plus supports CHIMES development, it limits the designs that CHIMES can evaluate. To make CHIMES a useful tool to GUI designers who do not use TAE Plus, we are developing an interface to OSF/Motif's user interface language (UIL)[7], which will allow CHIMES to evaluate any Motif-based design.

We are also currently developing a capability to allow CHIMES users to customize the knowledge base. We have demonstrated that CHIMES can work with a knowledge base containing several sets of rules. Switching from one set of rules to another does not require recompiling. Further, we have demonstrated that a rule can be modified through the CHIMES user interface and that the modified rule can be sent back to the knowledge base for execution. Now we are developing a capability to allow CHIMES users to set up new guidelines by customizing existing guidelines. A new guideline can later be loaded into the CHIMES knowledge base for evaluating GUI designs.

Other plans for the future call for research into possible uses for CHIMES as an intelligent agent and for experimental evaluation of the effects of CHIMES capabilities on the performance of user-interface designers.

References

[1] G. Fischer, A. C. Lemke, T. Mastaglio, and A. I. Morch. Using critics to empower users. In *Proceedings of CHI'90 Human Factors in Computing Systems*, pages 337–347, N.Y., 1990. ACM/SIGCHI.

[2] J. C. Giarrantano. *CLIPS User's Guide (Version 5.0)*. NASA Lyndon B. Johnson Space Center, Houston, TX, January 1991.

[3] J. Grudin. The case against user interface consistency. *Communications of the ACM*, 32:1164–1173, 1989.

[4] J. Jiang, L. E. Carter, E. D. Murphy, S. C. Bailin, and W. F. Truszkowski. Automated evaluation of graphical user interfaces (GUIs) using CHIMES. In *2nd Mid-Atlantic Human Factors Conference*, pages 57–63, Fairfax, VA, 1994. George Mason University.

[5] J. Jiang, E. D. Murphy, and L. E. Carter. Computer-human interaction models (CHIMES), revision 3. Technical Report DTSL-94-008, NASA-Goddard Space Flight Center, Greenbelt, MD, 1994.

[6] D. J. Mayhew. *Principles and guidelines in software user interface design.* Prentice-Hall, Englewood Cliffs, NJ, 1992.

[7] Open Software Foundation. *OSF/Motif Programmer's guide (Revision 1.1).* Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[8] Open Software Foundation. *OSF/Motif style guide (Revision 1.2).* Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[9] M. R. Szczur. Transportable applications environment (TAE) plus user interface designer workbench. In *Proceedings of CHI'92 Human Factors in Computing Systems*, pages 231–232, New York, May 1992. ACM/SIGCHI.

[10] T. Weldon and J. Odubiyi. Users guide for the request oriented scheduling engine (ROSE). Technical Report DSTL-89-021; CSC/TM-90/6108, NASA-Goddard Space Flight Center, 1990.

N95-17561

111146

354223

SEQ_REVIEW: A Tool for Reviewing and Checking Spacecraft Sequences

Pierre F. Maldague
Mekki El-Boushi
Thomas J. Starbird
Steven J. Zawacki

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099

ABSTRACT

A key component of JPL's strategy to make space missions faster, better and cheaper is the Advanced Multi-Mission Operations System (AMMOS), a ground software intensive system currently in use and in further development. AMMOS intends to eliminate the cost of re-engineering a ground system for each new JPL mission. This paper discusses SEQ_REVIEW, a component of AMMOS that was designed to facilitate and automate the task of reviewing and checking spacecraft sequences.

SEQ_REVIEW is a smart browser for inspecting files created by other sequence generation tools in the AMMOS system. It can parse sequence-related files according to a computer-readable version of a "Software Interface Specification" (SIS), which is a standard document for defining file formats. It lets users display one or several linked files and check simple constraints using a Basic-like "Little Language".

SEQ_REVIEW represents the first application of the Quality Function Deployment (QFD) method to sequence software development at JPL. The paper will show how the requirements for SEQ_REVIEW were defined and converted into a design based on object-oriented principles. The process starts with interviews of potential users, a small but diverse group that spans multiple disciplines and "cultures". It continues with the development of QFD matrices that relate product functions and characteristics to user-demanded qualities. These matrices are then turned into a formal Software Requirements Document (SRD). The process concludes with the design phase, in which the CRC (Class, Responsibility, Collaboration) approach was used to convert requirements into a blueprint for the final product.

THE UPLINK PROCESS

The multi-mission environment in which SEQ_REVIEW is intended to operate is fairly complex. This Section introduces the basic elements of the uplink process and explains where SEQ_REVIEW fits in that process.

Sequence Generation

The ultimate goal of the uplink process is to allow ground operations personnel to control the spacecraft by sending it radio signals that the spacecraft can receive, decode and store in its memory. The decoded information usually consists of commands that are to be executed in a precise sequence at specified times. We will refer to these commands as "spacecraft commands", and to a set of such commands sent to the spacecraft as a whole as an "on-board sequence".

Much of the uplink process is concerned with the planning, generation and verification of on-board sequences. This process can involve many people: mission scientists interested in planetary data request new observations;

engineers concerned about the capability, health and safety of the spacecraft issue maintenance requests; mission planners try to accommodate requests into a realistic schedule; sequence engineers translate high-level requests into detailed instructions that will cause the spacecraft to perform the required tasks; and finally, the flight team must check the detailed sequence against all flight rules, possibly including rules that were added at the last minute to compensate for equipment not operating at specification or software bugs aboard the spacecraft.

Analogy with Programming

The process just described resembles that of generating executable code for an ordinary computer, an analogy that will be used extensively in this paper. The spacecraft and its sequence are analogous to a microprocessor and its machine instructions. The process of planning and generating a sequence is similar to the task of designing and implementing software. Just as software engineers would find it impossible to do their job using machine code, sequence engineers find it useful to work not with the on-board sequence itself, but with a human-readable version of it that is similar to an assembly language program.

Of course our analogy between a spacecraft and a microprocessor is not perfect. Modern spacecraft have considerable processing power at their disposal, so that spacecraft commands are usually much more complex than typical microprocessor instructions. This complexity is reflected in the large number of arguments required by many commands. In spite of this, the analogy between spacecraft commands and assembly code remains valid in the sense that spacecraft commands are expressed in a special-purpose language that is hard to understand unless one is familiar with the architecture of the spacecraft.

Translating Requests into Commands: SEQ_GEN

Programming efficiency can be increased dramatically when using a high-level language instead of assembly code. The tool that makes this possible is the compiler, which translates high-level code into assembly code. Sequence engineers also find that programming sequences directly is prohibitively difficult, and that time can be saved by expressing commands as high-level "Requests" instead of low-level "Commands". Something similar to a compiler is now needed to translate the former into the latter. In the AMMOS system, this role is assumed by SEQ_GEN, a program that expands requests into sequences of commands. The figure on the following page shows the similarities between the conventional code development process and the uplink process.

Since SEQ_GEN is a multi-mission tool, it must obtain mission-specific information from external files. This is unlike most compilers, which are hard-coded around the syntax of a specific language. A second difference with compilers is that SEQ_GEN defines and maintains an internal model of the spacecraft. The mission-specific files required by SEQ_GEN therefore need to describe the spacecraft model as well as the basic commands and their effect on the model. Other mission-specific files used by SEQ_GEN define high-level "activity types", which are analogous to subroutines, and flight rules, which are formulated in terms of the spacecraft model (see Ref. 1 for more details on the operation of SEQ_GEN).

SEQ_GEN generates two basic output files. The first file is the Spacecraft Sequence File, which is an ASCII representation of the actual on-board sequence. This file is an input to another program, SEQ_TRAN, which converts ASCII mnemonics into binary code, links the program, and performs necessary memory management and packetization tasks. The second file is the Predicted Event File (PEF), which shows in time-ordered fashion the complete sequence of commands, ground events, and optionally the status of the internal spacecraft model that is predicted to result from the Request File. In the following, we focus on the PEF.

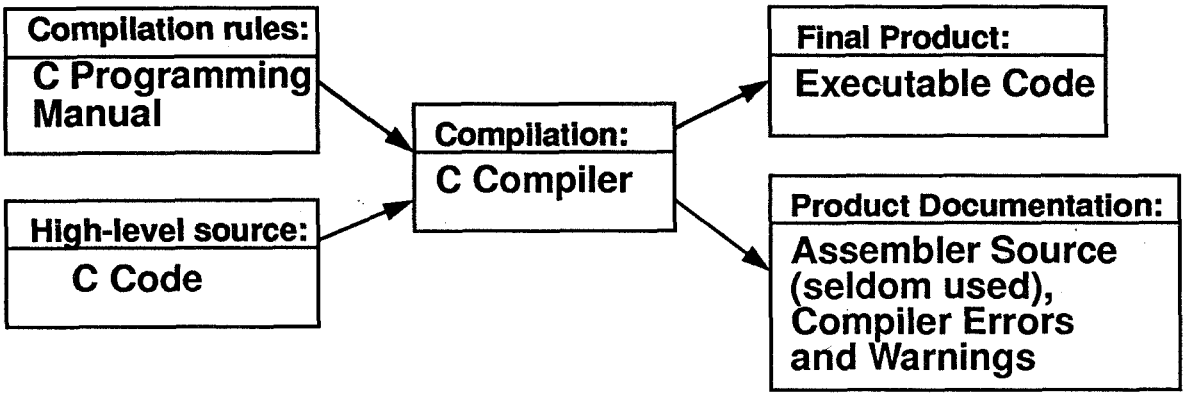


Fig. 1: SOFTWARE DEVELOPMENT PROCESS

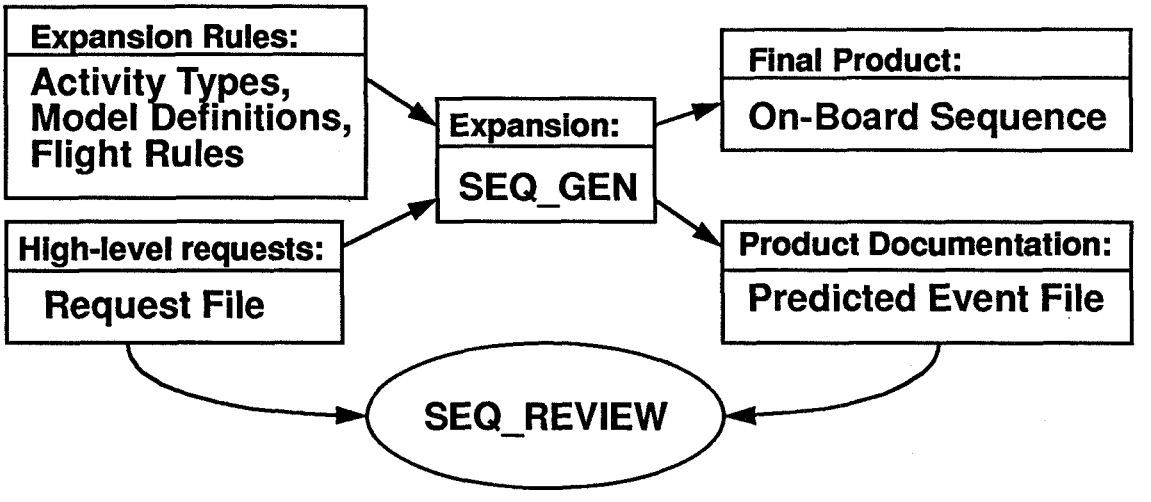


Fig. 2: SEQUENCE GENERATION PROCESS

Checking the Sequence: SEQ_REVIEW

Testing conventional software is a straightforward procedure: the worst that can happen is that the program "crashes" under the benevolent supervision of the operating system. In space exploration, however, sequence engineers do not have the luxury of trying again: the sequence HAS to work the first time. Simulation tools such as those incorporated into SEQ_GEN provide valuable help in validating sequences. However, the final arbiter of a sequence's validity is the sequence engineer and other flight team members who review it.

The main difficulty in checking a sequence is to zero in on the information that is pertinent to a single flight rule or constraint. The documentation provided by SEQ_GEN in the form of an event file is quite extensive, but that makes it hard to read. Traditionally, sequence checkers have used a variety of ad hoc methods to deal with this complexity:

- manual inspection of computer printouts
- BASIC and C programs that "strip" event files of unwanted information
- UNIX "awk" scripts for reformatting event files

The purpose of SEQ_REVIEW is to offer sequence engineers and other sequence reviewers an alternative, multi-mission package that is easy to use, adapt, port and maintain.

THE REQUIREMENTS PHASE

The SEQ_REVIEW software requirements document (SRD) was based on the TQM tool Quality Function Deployment (QFD), which we briefly outline here. A more detailed account of our QFD approach will be found in Ref. 2.

The QFD Approach

The emphasis in the QFD approach is on customer requirements and on how to ensure that these requirements are reflected, i. e., "deployed", through the design process. The first step in the process as implemented here was to collect information from potential users of the software through interviews. Responses to the interviews were then analyzed by a Committee with representatives from user, developer and systems engineering groups. The primary goal of this first step was to come up with three basic lists:

- Demanded Qualities, which express what the user wants to be able to do with the program. Example: easy to strip and reformat a PEF. All of these Qualities were taken from user responses. The Committee organized them into 6 broad categories such as "Sequence Validation" and "Ease of Use", and then into additional sub-categories such as "find stimuli of violations" and "filter and re-order fields".
- Quality Characteristics, which express in a quantitative manner how users and developers would rate the SEQ_REVIEW product against other methods for achieving the same task. Examples: "check one constraint in at most 5 lines of SEQ_REVIEW 'Little Language' code"; "keep the program to 18,000 or fewer lines of code".
- Functions. These are program features which will allow the product to meet customer requirements. Most of these were requested by users directly ("Perform time conversion on request"); a few were provided by developers.

A questionnaire was then circulated, asking users to rank the Demanded Qualities in order of importance. The responses were used to compute an average score for each one of the Demanded Qualities. Listed at the top were:

- "Easy to Strip and Reformat a File"
- "Draw Timelines"
- "Reduce the Amount to Read"
- "Allow Annotations"
- "Work with Event Files"

Some of the least important Qualities were "Correlate Event and Request Files" and "Work with Spacecraft Sequence Files." Clearly, our users were mostly interested in making event files easier to read.

In the next step of our QFD implementation, these user-assigned priorities were propagated through a set of "correlation matrices" that relate the users' Demanded Qualities to factors that the developers can influence through their design, primarily Quality Characteristics and Functions. These matrices specify whether for any given Demanded Quality/ Quality Characteristic or Demanded Quality/ Function pair, the correlation between the two members of the pair is (i) nonexistent, (ii) weak, (iii) moderate or (iv) strong.

Based on these matrices, we used a QFD software package to compute scores for each Quality Characteristic and for each Function. These scores were then used to prioritize the development process as well as the overall objectives for the product. The highest-priority items were

- provide users with a rule definition language (the "Little Language")
- provide a graphic interface that lets users specify a rule in under 5 minutes
- design the "Little Language" so that users can formulate a rule in 5 statements or less
- adapt existing timeline generation software from other programs (such as SEQ_GEN)

Some of the less important characteristics were "Ability to add a feature in one week or less", and "Keep the code to 18,000 lines or less". Clearly, the emphasis was on providing users with simple ways to express rules and on providing timeline capabilities without re-inventing the wheel.

Generating Requirements

Since the QFD methodology does not prescribe a specific method for generating requirement documents, we had to come up with our own. Our first attempt consisted in translating the correlation matrices for Functions and Quality Characteristics into plain English. The Functions were used primarily to explain the method used to meet the requirements, while the Quality Characteristics were used primarily to state testable objectives for the finished product.

This first approach was rejected because the resulting requirements document was hard to read. The problem was that our lists of Qualities and Functions did a good job of summarizing user requirements, but did not provide the reader with much of a feel for the functionality of the SEQ_REVIEW product.

Our second, more successful approach was to realize that the task of stating our requirements was going to be a lot simpler if we first carried out a couple of "pre-design" steps prior to writing requirements:

(i) design a tentative Graphical User Interface (GUI). This would give us a chance to organize user-demanded features in a logical manner. It was also decided to implement this preliminary design in Visual BASIC and make it available to potential users for feedback.

(ii) show a concrete example of a Little Language (LL) and explain how it relates to the desired functionality of SEQ_REVIEW. This step actually required little effort since a LL was already developed as part of the prototyping effort (see the next Section). While this LL didn't meet all the requirements, it is close enough to provide the reader with a sense of how the product would operate.

While these tasks delayed the SEQ_REVIEW SRD somewhat, we felt that the overall schedule would not be adversely impacted. First, additional up-front work would make design and implementation easier later on. Second, our tentative GUI could be turned very easily into the first Section of the SEQ_REVIEW User's Guide, again saving us time later on. Finally, we felt that making our tentative GUI available to users early on would contribute significantly to the ultimate success of SEQ_REVIEW.

The software requirements for SEQ_REVIEW were strongly influenced by two parallel efforts that took place in the summer of 1993.

Prototyping Activity

First, prototypes were built to demonstrate the feasibility of SEQ_REVIEW. These prototypes established a firm basis for the following concepts:

1. zero in on useful information by letting the user specify patterns and searches in a simple, intuitive way
2. translate sequence files into text files suitable for input into spreadsheet programs such as Lotus 1-2-3
3. express rules and constraints easily by writing simple programs in a Little Language designed to handle the type of information found in sequence files
4. reformat sequence files by letting users specify records of interest and fields of interest within these records, using either simple pattern definitions or the Little Language
5. build on previous experience by saving search patterns and simple algorithms so they can be reused in future review sessions
6. allow the program to read arbitrary (within reason) text files by specifying the file format on-line, as opposed to re-

compiling a new version of the software featuring new hard-coded file formats

Second, a Quality Function Deployment (QFD) Committee was formed. This Committee included representatives from potential users of SEQ_REVIEW as well as software developers. The Committee used the QFD methodology to identify desired features and qualities that the SEQ_REVIEW product should exhibit. How this work was used to establish the present requirements was described in the previous paragraphs.

User Interface

Since the primary purpose of SEQ_REVIEW is to display sequence file information to the user, it is anticipated that most users will want to interact with the program through a Graphical User Interface (GUI) similar to that used by many text editors. This should be qualified in two ways:

- a small but significant minority of potential SEQ_REVIEW users requested the ability to control the program through a command-line interface, as opposed to clicking on buttons and pull-down menus;
- SEQ_REVIEW needs to support "batch-mode" operation, in which a pre-defined set of commands is fed to the program from a command file. In this mode, SEQ_REVIEW acts as a "filter", e. g. to identify violations of rules not yet implemented in SEQ_GEN.

To accommodate these requirements, SEQ_REVIEW will be provided in two forms: interactive and batch. The interactive version will be GUI-based. In addition to the usual menu bar and push-button, the GUI will feature a special window for command-line input. Every SEQ_REVIEW function will be accessible as a command line as well as through menu selections. "Menu accelerators" will also be provided; these are short, user-definable keystroke combinations that can be used as a substitute for menu selections.

The batch version of SEQ_REVIEW will not display anything to the user and will accept commands from "standard input", which can be either the user's keyboard or a text file specified to UNIX as a source of redirected input. The only use of the batch mode version will be to create output (text) files that can be read by the user or scanned automatically to detect rule violations. It is anticipated that this version of SEQ_REVIEW will be used in highly automated, Operations-type throughput-critical environments.

The figure on the next page shows our preliminary design for a top-level menu of SEQ_REVIEW that satisfies user-demanded qualities and functions. When the user first activates the program, only the top (highlighted) line of each menu is visible; these lines form the "Menu Bar" at the top of the SEQ_REVIEW screen. The expanded menus shown in the figure appear when the user clicks on the corresponding menu title in the Menu Bar.

THE DESIGN PHASE

The method used to design SEQ_REVIEW is essentially the Class/Responsibility/Collaboration (CRC) approach described by Wirfs-Brock et al. (Ref. 3), with the following modifications/adaptations:

(M1)the starting point of the design is the SRD, which concentrates almost exclusively on the user's perspective of the program. The requirements do not address how the program is supposed to accomplish the various tasks.

(M2)SEQ_REVIEW will rely on the MOTIF toolkit for all graphics. Because MOTIF has its own class definitions, there is potential conflict with internal SEQ_REVIEW classes. This problem is not really discussed in Ref. 3.

(M3)a specific methodology was adopted early on to deal with the fact that SEQ_REVIEW needs to be delivered in two flavors, GUI and batch. The decision was that the two programs would share the same object structure, and that MOTIF, X Toolkit and X Window calls

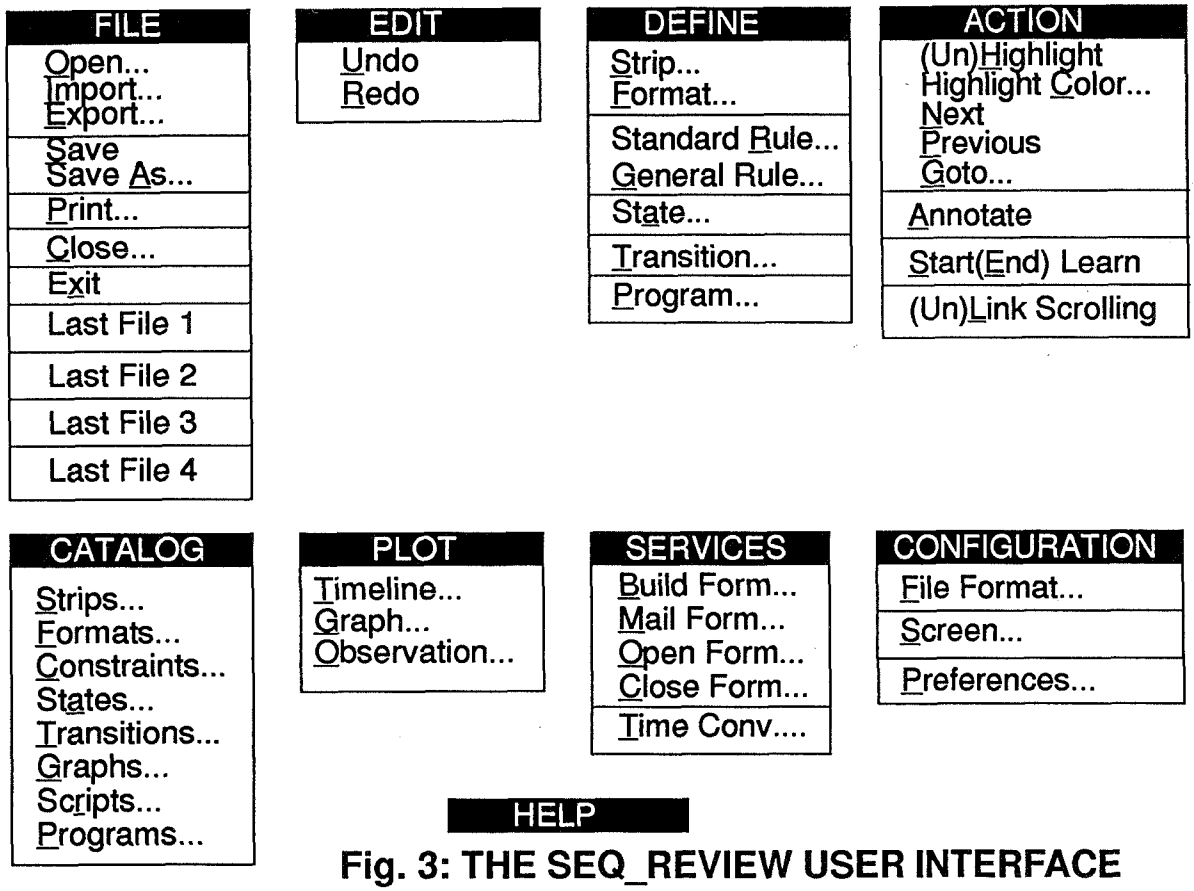


Fig. 3: THE SEQ_REVIEW USER INTERFACE

would be "dummied up" in the batch version.

(M4)we decided to use a fair amount of "vertical inheritance" in our design, as opposed to the Wirfs-Brock strategy which emphasizes "horizontal inheritance".

The starting point of our design was an index of keywords extracted from the SRD. The index was then edited into a table of "SRD objects", to be used as a first step towards designing the classes of SEQ_REVIEW.

As a result of (M1), however, we found that the SRD was not "rich enough" as a source of objects when it came to describing the inner workings of the program. In particular, it was difficult to write scenarios that went beyond the user interface. We then decided to use the scenarios as a source of objects, rather than as a means to check the validity of the design. This is of course dangerous, since many design decisions could be made inadvertently while writing scenarios. We avoided this problem by keeping the scenarios as simple, "down to Earth" as possible and subjecting them to frequent scrutiny.

After writing five or six scenarios and looking at the objects that would be necessary to support them, it became clear that objects fell into well-defined classes, and that these classes should be organized into hierarchies using the inheritance scheme. The resulting classes provided our first "draft" of the design.

A "shell" program, featuring all these classes but only some of their responsibilities, was implemented in C++. This was done to validate our design and to make sure that the C++ compiler would not object to our inheritance scheme. We learned the following lessons:

- Our design is compatible with the C++ compilers we are using.
- Inheritance, which had been the focus of our class-building effort, is only part of the story. It became clear that classes had a definite "personality" and that classes with similar personalities should be grouped in separate subsystems.

This naturally led to the next phase in the design: organizing classes into subsystems. The need for this was made more pressing by the requirement phrased in (M2) and (M3) above: we need a clear description of how MOTIF is to be interfaced to the rest of the system.

In the next step of the design, we built two more prototypes. The first one was a refinement of the earlier "shell". Although this new version was still only a shell, it was able to print in indented, scenario-like style what it was doing. It also provided a rudimentary user interface which demonstrated how the menu structure and the callback philosophy of the GUI version could be brought into the batch version of SEQ_REVIEW.

The second of these prototypes consisted of a MOTIF implementation of the "Define Strip" panel of the SEQ_REVIEW user interface. This is probably the most complex graphic object in the GUI. The prototype therefore demonstrated the feasibility of our approach and helped focus the discussion of how the GUI and batch versions of SEQ_REVIEW would coexist.

As a result of all this prototyping activity, we gained the confidence necessary to organize our preliminary classes into well-defined subsystems. We feel that our subsystem design is robust enough that it will survive any last-minute change to the class definitions, and we therefore look at our subsystem descriptions as the central part of our design.

CONCLUSION

SEQ_REVIEW is a tool that will facilitate the task of reviewing the various text files associated with spacecraft sequences. The requirements for SEQ_REVIEW were derived from interviews of potential customers. These interviews were converted into a requirements document using the QFD approach. Requirements were then translated into a high-level design using an object-oriented methodology. The overall process was facilitated by the use of numerous prototypes. Multi-mission aspects were built into the requirements from the start.

ACKNOWLEDGMENTS

We extend our thanks to our many colleagues who contributed their time and insight, and in particular to the flight team members who helped phrase the requirements for SEQ_REVIEW. Special thanks to Todd Bayer, Vickere Blackwell, Carlos Carrion, Julia Henricks, Tim Kaufman, Bob Kerr, Chuck Klose, Bill Nelson, Brian Paczkowski, Steve Peters and Bruce Waggoner for their thoughtful comments, and to Jose Salcedo for sharing his knowledge of the sequence generation process.

SEQ_REVIEW is currently under development at the Jet Propulsion Laboratory, California Institute of Technology, under contract to the National Aeronautics and Space Administration.

REFERENCES

1. Salcedo, Jose, & Starbird, Thomas (1994, Nov.). *SEQ_GEN: A Comprehensive Multimission Sequencing System*, Space Ops '94 (These Proceedings).

2. ELBoushi, M. , Zawacki, S., & Domb., E. (1994, June). *Towards Better Object Oriented Software Designs With Quality Function Deployment*, Transactions from The Sixth Symposium on Quality Function Deployment, Novi, Michigan.

3. Wirfs-Brock, Rebecca, Wilkerson, Brian, & Wiener, Lauren (1990). *Designing Object-Oriented Software*, Prentice-Hall.

N95- 17562

11147

SIMPLIFYING OPERATIONS WITH AN UPLINK/DOWNLINK INTEGRATION TOOLKIT

354225

P-7

SUSAN MURPHY (Contact)
KEVIN MILLER
ANA MARIA GUERRERO
CHESTER JOE
JOHN LOUIE
CHRISTINE AGUILERA

Operation Engineering Lab
Jet Propulsion Laboratory, MS 301-345
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109-8099

ABSTRACT

The Operations Engineering Lab (OEL) at JPL has developed a simple, generic toolkit to integrate the uplink/downlink processes, (often called *closing the loop*,), in JPL's Multimission Ground Data System. This toolkit provides capabilities for integrating telemetry verification points with predicted spacecraft commands and ground events in the Mission Sequence Of Events (SOE) document. In the JPL ground data system, the uplink processing functions and the downlink processing functions are separate subsystems that are not well integrated because of the nature of planetary missions with large one-way light times for spacecraft-to-ground communication. Our new closed-loop monitoring tool allows an analyst or mission controller to view and save uplink commands and ground events with their corresponding downlinked telemetry values regardless of the delay in downlink telemetry and without requiring real-time intervention by the user.

An SOE document is a time-ordered list of all the planned ground and spacecraft events, including all commands, sequence loads, ground events, significant mission activities, spacecraft status, and resource allocations. The SOE document is generated by expansion and integration of spacecraft sequence files, ground station allocations, navigation files, and other ground event files. This SOE generation process has been automated within the OEL and includes a graphical, object-oriented SOE editor and real-time viewing tool running under X/Motif. The SOE toolkit was used as the framework for the integrated implementation.

The SOE is used by flight engineers to coordinate their operations tasks, serving as a predict data set in ground operations and mission control. The closed-loop SOE toolkit allows simple, automated integration of predicted uplink events with correlated telemetry points in a single SOE document for on-screen viewing and archiving. It automatically interfaces with existing real-time or non real-time sources of information, to display actual values from the telemetry data stream.

This toolkit was designed to greatly simplify the user's ability to access and view telemetry data, and also provide a means to view this data in the context of the commands and ground events that are used to interpret it. A closed-loop system can prove especially useful in small missions with limited resources requiring automated monitoring tools. This paper will discuss the toolkit implementation, including design trade-offs and future plans for enhancing the automated capabilities.

INTRODUCTION

The Operations Engineering Lab (OEL) at NASA's Jet Propulsion Laboratory has developed a simple, generic toolkit that

integrates uplink events with downlink telemetry information, (often called *closing the loop*,). This toolkit provides capabilities for integrating telemetry verification points and ground monitoring information with planned spacecraft commands and ground events in the Mission Sequence Of Events (SOE) schedule. In the existing SOE for planetary missions, each spacecraft command item has a descriptive text field that contains a list of related engineering telemetry parameters. These parameters are monitored by mission controllers using a data monitor tool that processes and displays the downlink telemetry stream. However, the relevant downlink telemetry for command verification may not arrive for hours or even days after the commands have been sent because of the large one-way light times and limited contact periods of some planetary missions. The closed-loop system will integrate these tasks by interfacing the SOE with the real-time or non-real-time telemetry data streams and automatically append appropriate channel values and limit checks with command and ground event items. Our new closed-loop monitoring tool allows an analyst or mission controller to browse and archive uplink commands and ground events with their corresponding downlinked telemetry values regardless of the delay in downlink telemetry and without requiring real-time intervention by the user. Figure 1 shows a sample SOE with integrated telemetry channel information.

BACKGROUND

An SOE document is a time-ordered list of all the planned ground and spacecraft events, including all commands, sequence loads, ground events, significant mission activities, status and resource allocations. The SOE document is generated by the multimission control team by expansion and integration of sequence and ground files. This SOE generation process has been automated and includes a graphical, object-oriented SOE editor and viewing tool running under X/Motif. The SOE is used by the mission controllers and spacecraft and instrument engineers to coordinate their operations tasks, serving as a predict data set in ground operations and mission control.

APPROACH

The Operations Engineering Lab (OEL) proposed a research task to design and implement a toolkit that allows simple, automated integration of predicted uplink events with actual downlink telemetry in a single SOE document for viewing and archiving. This integrated SOE serves as the basis for a closed-loop monitoring toolkit that can automatically interface with existing real-time or non-real-time sources of information and display only selected values from the telemetry data stream.

A significant research effort was in the design and implementation of the interprocess communication interfaces and interactive controls for retrieving and passing information from a variety of downlink processing applications to the SOE tool. A flexible approach was chosen to allow phasing of planned future enhancements, including expansion of the SOE capabilities for automated mission controller logs, telemetry logging, system test procedure execution, and automated command verification.

During the requirements analysis, the JPL Multimission Control Team (MCT) indicated that the closed-loop monitoring system should include automated log keeping capabilities for mission controllers in order to include their real-time logs in the sequence of events. In the current operations environment, the MCT logs real-time information on specific uplink and downlink events on hand-written forms. The MCT cannot meet an electronic logging requirement without automation tools - manually typing a log report would be impossible during intense periods. Although the SOE tool can currently be used to enter comments and act as a logging tool, the mission controller logging requirements were very broad and not clearly defined enough to implement within an already-developed tool such as the SOE editor. Thus, we decided to implement a separate tool for automated logging that would integrate manual log inputs, predicted events input files (such as the SOE), real-time broadcast data, and output from other

downlink telemetry processing and monitoring applications.

OEL LOGGING TOOL

The OEL Logging Tool (OLOG) provides automated and manual logging of predicted and actual mission events in a graphical easy-to-use format. The user interface and pull-down menus are completely configurable by an individual end-user to meet mission-specific needs. An initialization file can be read on startup that customizes menu options and defaults. The OEL Logging Tool is shown in Figure 2.

The OEL Log Tool provides capabilities for manual entries to the log, either by allowing a user to select items off pre-defined menus or entering text manually into the entry area. The pre-defined menus and default values can be customized for various teams by creating an initialization file. The OEL Log Tool is also designed to interface in real-time with external input sources (downlink telemetry, monitor data, interprocess messages, predicts and actuals files), thus providing capabilities for automated log entries. The tool is designed to allow automated and manual log entries to occur concurrently with automatic ordering based on time tags. The current tool implementation includes several mechanisms for communicating with downlink processes based the MGDS custom Data Transport Subsystem (DTS) services. We have also implemented a telemetry data processing program that captures and processes data from the real-time telemetry data stream or spooler files, and passes log messages into the OEL Log Tool.

INTERPROCESS COMMUNICATION

The OEL Log Tool uses the MGDS DTS services to implement message passing capabilities. The tool connects to a broadcast channel or virtual circuit if a channel option is chosen. While processing input events, the tool continually checks for pending messages.

Several programs and routines have been written to implement real-time downlink data processing functions and cooperative message passing mechanisms. These programs provide a flexible approach to process various input data sources, which are then reformatted as log messages to be sent to the OEL Log Tool.

- A program that uses a DTS broadcast channel or virtual circuit for sending time-stamped messages. The OEL Log Tool cooperates by receiving and processing the messages for input into the log in time order.

- A generic routine that uses the MGDS Data Transport functions (DTS) to connect to a broadcast channel or virtual circuit for sending time-stamped messages and a corresponding OEL Log Tool routine that creates a broadcast channel or virtual circuit to receive messages.

- A program that reads the output from an existing real-time Smart Alarm Tool. It processes the output and then sends log messages to the OEL Log Tool.

- A program that reads MGDS downlinked telemetry data from a virtual circuit, real-time broadcast channel, a bytestream file, and/or a spooler file using the DTS functions. It processes the data by parsing telemetry headers and checking for selected data types which are then extracted, processed, and passed to the OEL Log Tool. It uses DTS services to send log messages to the OEL Log Tool. This program provides a model for building additional programs for processing data from real-time telemetry streams or from data output by other applications and then generating log messages to be integrated with the predict information in the log tool.

Figure 3 shows the OEL Log Tool and its telemetry processing interfaces.

CLOSED-LOOP SOE TOOL

The new closed-loop SOE tool is designed to allow integration of uplink predict events with actual downlink telemetry values.

Ground event and spacecraft command items in the SOE have associated telemetry parameters that can be used to verify the execution of each command or ground event or to establish the state of the spacecraft or ground system. These parameters are now interleaved with the commands and ground events in the closed-loop SOE, for easy access and viewing by the analyst. The closed-loop monitoring system can automatically interface with existing real-time sources of information, to capture and display actual values from the telemetry data stream. The graphical SOE viewing tool allows a user to highlight events of interest with a mouse on the screen and mark them for automatic alarm notification in real-time. This tool also allows a user to run the SOE in demand mode, specifying any time desired, and to search or step through the document for events of interest.

The following capabilities have been incorporated into the SOE tool to implement real-time, closed-loop monitoring capabilities:

- *Real-Time Capabilities*: A scroll bar has been added to allow for scrolling during real-time viewing. Real-time viewing can be accessed and controlled from a menu. It is possible to step through each event under user control or to run the SOE in real-time or in playback mode. The real-time control capabilities are based on an interrupt algorithm.
- *Channel Objects*: A new item format has been defined for the SOE to allow creation of a telemetry channel object. Each channel object is defined by a description field and is associated with the uplink predict item preceding it. There can be multiple channel objects associated with a single uplink event. The uplink/downlink command/channel integration is based on cross-referenced information within the telemetry and command dictionaries.
- *Channel Options*: A new option button has been implemented that allows a user to optionally display channel records in the SOE.
- *Downlink Interface*: Interprocess communication and data transport functions have been implemented to allow integration of the downlink telemetry stream.

Figure 4 shows the graphical closed-loop SOE tool.

BENEFITS

New missions are demanding electronic mission controller logs rather than the current hand-written reports. Without the automated log keeping capabilities in the new OEL Log Tool, a controller would spend most of their time manually typing in electronic log reports.

The closed-loop SOE and OEL Log Tool implementations have laid the groundwork for an advanced closed-loop monitoring system that can significantly reduce the need for operations teams to understand the complex set of processing and display tools in the existing JPL Multimission Ground Data System (MGDS). The current MGDS approach requires understanding multiple subsystems, and their often subtle interfaces, to allow an end-to-end processing of downlink data. There is also no current method to integrate or compare predict with actual values. The closed-loop SOE tool can greatly simplify a user's ability to access and view telemetry data, and provides a means to view this data in the context of the commands and predicted values that are used to interpret it. In this context, it is expected that significant cost savings can be realized from the productivity improvements that will be realized over the hundreds of current users of the SOE document and related uplink tools.

Some proposed areas of future work include expansion of the closed-loop SOE capabilities for adaptation to system test, dynamic alarms based on tolerances from predicted events, automated monitoring of spacecraft and ground system configurations at selected times, real-time timeline display of the SOE, generation of as-flown SOE schedules, and automated command verification. An advanced closed-loop monitoring system is essential to a more automated monitor and control system and to significant productivity improvements for the smaller missions of the future.

SEQUENCE OF EVENTS: YEAR-DAY OF YEAR --> 1993-280 GENERATED ON DAY OF YEAR = 317 SCROLL
S/C = 094 INPUT FILE NAME --> c2.fs.epef.4
SEQ = C2.FS OUTPUT FILE NAME --> c2.soe November 13, 1993 00:54:18 UTC

ITEM NO	GROUND TIME DDD HH:MM:SS	T B	ACTION	EVENT DESCRIPTION	DSN	COMMAND (M=RTC)	S/C EVENT TIME	TLM FMT
72	280 21:42:14	E		ENABLE ---> ROCKET ENGINE ASSEMBLY 13 TURN ON --> CATBED HEATER		PRR13E	275 21:29:43	
				P-0079 REA_CBH13_EN P-0962 REA_CBH13_TP 15 530 RED: 10 550 DEG_C				
73	280 21:42:14	E		ENABLE ---> ROCKET ENGINE ASSEMBLY 15 TURN ON --> CATBED HEATER		PRR15E	275 21:29:43	
				P-0081 REA_CBH15_EN P-0964 REA_CBH15_TP 15 530 RED: 10 550 DEG_C				
74	280 21:42:15	E		ENABLE ---> ROCKET ENGINE ASSEMBLY 10 TURN ON --> CATBED HEATER		PRR10E	275 21:29:44	
				P-0076 REA_CBH10_EN P-0959 REA_CBH10_TP 15 530 RED: 10 55 DEG_C				
75	280 21:42:15	E		ENABLE ---> ROCKET ENGINE ASSEMBLY 12 TURN ON --> CATBED HEATER		PRR12E	275 21:29:44	
				P-0078 REA_CBH12_EN P-0961 REA_CBH12_TP 15 530 RED: 10 550 DEG_C				
76	280 21:42:15	E		ENABLE ---> ROCKET ENGINE ASSEMBLY 14 TURN ON --> CATBED HEATER		PRR14E	275 21:29:44	
				P-0080 REA_CBH14_EN P-0963 REA_CBH14_TP 15 530 RED: 10 550 DEG_C				
77	280 21:42:15	E		ENABLE ---> ROCKET ENGINE ASSEMBLY 16 TURN ON --> CATBED HEATER		PRR16E	275 21:29:44	
				P-0082 REA_CBH16_EN P-0965 REA_CBH16_TP 15 530 RED: 10 550 DEG_C				

Figure 1 - Sequence of Events (SOE) Printout

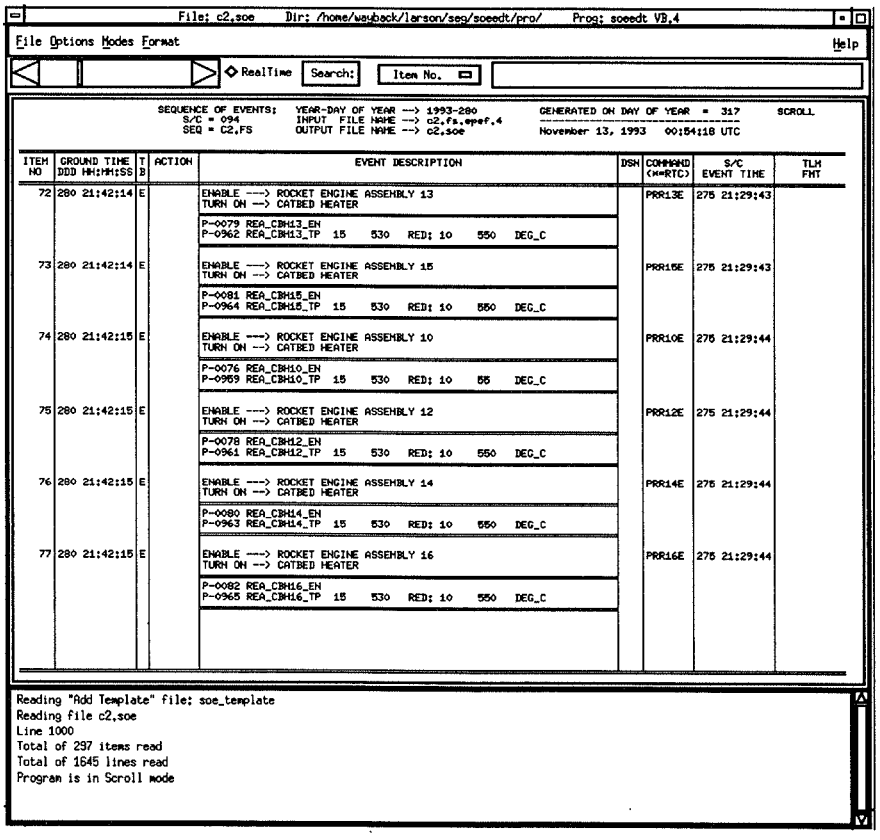


Figure 4 - Realtime Closed Loop SOE Tool

ORIGINAL PAGE IS OF POOR QUALITY

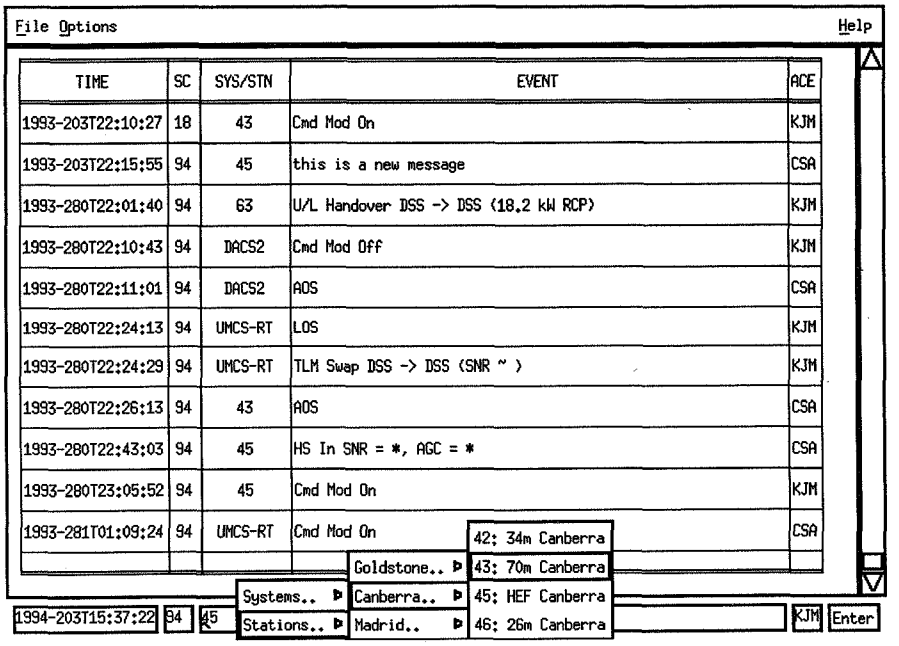


Figure 2 - OEL On-Line Logging Tool (OLOG)

Mission Control Team Operations Log

DOY: 211

Page 1
7/30/93

TIME	SC	SYS/STN	EVENT	ACE
1993-054T03:00:27	94	BCUI	001-00:00:00 GLLTISB received SFOC G/W query for routing tbl [054-03:00:27]	BC
1993-211T20:46:48	94	45	S/C 2-Way	MD
1993-211T20:48:38	94	TLM	Id: C307 -- SCP Telemetry SFDU # 25 DDP-ID = C307 SCP Telemetry 00c0 eb73 0079 1891 0be1 0019 0091 SCLK: 412158945 Segment: 25 EDF Status Word 0091 MISSION MODE D=0, A=2 EDF Digital Telemetry 68 58 10 7f 7f 40 48 ef ff 9a EDF Analog Telemetry 00 0a 6a 66 7b 01 3c b4 b3 11 46 7a 7b 00 b4 00 7a 78 74 7a 46 88 6e 3b e7 83 bc 83 SCP Status Word de84 from CONTROL SCP CV Word c006 SCP Telemetry c006 c2b0 0001 feec 0001 8c35 13c2 cf54 0eb9 f872 0019 0008 eed8 fffc fb1e ffff 0522 10d8 b061 50e3 0000 0000 ffe8 0067 0032 0000 0000 5898 49d4 c006 ce30 0001 feec 0001 89f0 122d d48e	TLM

Figure 3 - Sample OLOG Printout

ACKNOWLEDGMENTS

This work was done at the Jet Propulsion Laboratory, California Institute of Technology, under a contract from the National Aeronautics and Space Administration. We would like to acknowledge the work of the technical staff in the OEL and the JPL Mission Operations Teams for their enthusiasm and support.

111148 N95-17563

354227 p-8

ELISA, A DEMONSTRATOR ENVIRONMENT FOR INFORMATION SYSTEMS ARCHITECTURE DESIGN.

Chantal PANEM
CNES French Space Agency
18 Avenue Edouard Belin
31055 Toulouse Cedex- FRANCE
Tel.: (33) 61 28 26 72
email: panem@melies.cnes.fr

ABSTRACT

This papers describes an approach of reusability of software engineering technology in the area of ground space system design. System engineers have lots of needs similar to software developers ones: sharing of a common data base, capitalization of knowledge, definition of a common design process, communication between different technical domains. Moreover system designers need to simulate dynamically their system as earlier as possible. Software development environments, methods and tools now become operational and widely used. Their architecture is based on a unique object base, a set of common management services and they home a family of tools for each life cycle activity. Late 92, CNES decided to develop a demonstrative software environment supporting some system activities. The design of ground space data processing systems was chosen as the application domain. ELISA (Integrated Software Environment for Architectures Specification) was specified as a "demonstrator", i.e. a sufficient basis for demonstrations, evaluation and future operational enhancements. A process with three phases was implemented: system requirements definition, design of system architectures models and selection of physical architectures. Each phase is composed of several activities that can be performed in parallel, with the provision of Commercial Off the Shelves Tools. ELISA has been delivered to CNES in January 94, currently used for demonstrations and evaluations on real projects (e.g. SPOT4 Satellite Control Centre), it is on the way of new evolutions.

Keywords: PCTE (Portable Common Tool Environment), Satellite Control Centre, Ground segment, computer science, data processing, architecture, simulation, queueing networks.

This article starts by a presentation of the rationale for ELISA development, it describes the implemented life cycle, the workbench architecture and ends with first conclusions of the project.

FROM SOFTWARE ENGINEERING TO SYSTEM ENGINEERING...

After several years studying software engineering environments, mainly for the needs of the Hermes program, it appeared that they become operational and that any software project can find rather easily satisfying COTS environments, methods and associated tools.
On the other hand, in the area of system engineering, the lack of an approved, detailed and well-defined common design process, the variety of tools and the poorness of communication between them, increase difficulties when the project size grows.
The idea that the knowledge acquired in software engineering area could help the design of space systems was the starting point for this new orientation of our activities.

A REUSABLE TECHNOLOGY FOR "SYSTEM" DESIGN ?

Before trying to show how the technology was reused, lets explain it in few sentences.
Up-to-date software engineering environments are based on a so-called "integration platform" or "integrated project support environment (IPSE)" or "integration framework", in which a variety of tools are "plugged-in". The framework offers tools integration services, in three degrees: data integration via a repository, which role is to define, store and control all data needed by

PRECEDING PAGE BLANK NOT FILMED

PAGE 944 INTENTIONALLY BLANK

heterogeneous tools, control integration for communication between tools (interoperability) and presentation integration for uniform access to tools via the user interface. Some frameworks also provide process integration services for piloting of users activities according to a predefined life cycle. Two kind of services complete the environment to get a full "workbench": "horizontal" services, like documentation, configuration management, project management which are used in all the project phases, and "vertical services" which support individual life cycle activities by means of COTS tools (e.g. IDEF tool).
Such architecture did not seem limited to software engineering applications, but its adequation for system engineering needs had still to be proved.

ELISA, A FIRST STEP TOWARDS A WORKBENCH FOR DATA PROCESSING SYSTEM DESIGN

It was thus decided to develop a "demonstrator", i.e. a demonstrative environment based on an IPSE technology and supporting a coherent and consistent set of system activities. The chosen application domain was the design of data processing systems for ground space segment.

ELISA (Environnement Logiciel Intégré pour la Spécification d'Architectures informatiques/ Integrated Software Environment for Architectures Specification) users requirements specifications were produced in december 92. The objectives were:

- to demonstrate the benefits of software engineering IPSE frameworks in the system area,
- to increase CNES experience on three points: modeling of a design process, interface with a technical knowledge capitalization system and integration facilities,
- to show the interest of specific system tools and moreover of an integrated workbench with respect to isolated tools.

ELISA development was then reduced to the minimum set of services needed for demonstrations and evaluation but as a reusable basis for future enhancements. The constraint was to reuse as far as possible commercial tools and to limit specific developments.

THE ELISA REPOSITORY: a kernel for traceability and reusability.

ELISA is based on a PCTE repository (ECMA and Draft International ISO Standard). The repository is an object management system, which allows to define entities with the Entity-Relationship-Attributes model, to store them in a distributed way and to execute operations on them (calls to external tools). The ELISA data model has been defined in a modular way (thanks to PCTE) leading to an organised network of objects representing pertinent information for the user (functions, requirements, documents, architecture, equipments, simulation scenarios and results, etc..).

In ELISA, links between the objects represent either composition relationships, either trace relationships. Trace links are used to store implementation/ validation relations according to the process model steps: for example a computer linked to a performance requirement can mean that the computer implements it. Traceability allows the user to navigate directly between heterogeneous objects, to assess requirements coverage by architecture trade-offs and to analyse the impact of changes of a customer's requirement, or a system function. Through the trace matrixes produced by ELISA, the list of objects impacted by a modification is available at any step of the process.

The ELISA repository is also used as a "Technical Memory" storage, a place in the data model allows to capitalize information from old projects, feedback from previous studies, catalogues of hardware and software products and also architectural data from previous projects realised with ELISA.

The user can at any time consult and reuse a functional or physical architecture from this technical memory.

The definition of a complete, coherent and efficient data model is one of the tasks that require most reflexion, by the fact that it's the basis for tools integration and invocation. Extension to the data model is easy, but deletion or modification of data types seem more delicate once the environment is used by several projects.

WHICH ACTIVITIES DOES ELISA SUPPORT ?

Large projects are composed of a large number of complex and inter-related tasks. The initial work was to define the reduced life cycle that should implement ELISA, this comprises the definition of the activities that will be supported (WHY), their scheduling (WHEN), the persons who will perform them (WHO), the tools that will be used (HOW) and the products that should be available as inputs and outputs of the tasks (WHAT). This work appeared to be fundamental for the good achievement of the project.

The ELISA process model has three phases:
- System requirements definition,
- System architectural models design,
- Physical architectures assessment.

As it concerns a design process for early phases of a space system, flexibility is the major issue for an efficient assistance to users. The three phases are not purely sequential, but the user can complete them in an iterative way.
The EAST IPSE framework has been chosen, mainly for its ability to define, control and monitor any user defined process model, through the user interface. Different types of users have been defined (customer, architect, administrator and project manager). When a user starts the environment, he can activate tasks that have been assigned to him by the project manager. Starting a new project according to the ELISA process model become a mere operation.

PHASE I: DEFINITION OF SYSTEM REQUIREMENTS

The first task the designer deals with, is the capture of the requirements and constraints of the system. ELISA assists him in performing three activities: formalization of system and functional requirements, functional architecture analysis and definition of the logical sequencing of data processings.

Extract imposed requirements:
The first step is to deduce from input system specifications and customer interviews, the information which will be pertinent and/or constraining for the system architecture. These information are identified as requirements and can relate to several system aspects like performance, security, integrity, sizing, fault-tolerance.
With ELISA, requirements are managed by the LOTUS 123 spreadsheet tool, tables of requirements are created and filled-in by the user. Requirements are formalized by several attributes: an identifier, a textual description, a status (to be defined, hypothesis, computed, stable) and a value that can be the result of a formula computation from other requirements values.
Input specifications and interview notes can be stored in the repository if compatible with FrameMaker format, traceability links can be set towards them in order to keep the origin of design choices.

Analyse the functional architecture of the system:
The second step covers the analysis of the system functions. ELISA assists the user by the integration of the ASA tool supporting the IDEF0 methodology. The designer creates a functional model, edit it and refines the system functions in a hierarchical way, until obtaining a tree where leafs correspond to processes or software pieces. Each function is extracted and accessible in the repository as an object, automatically linked to its father and sons.

Tables of functional requirements can here also be attached to any function of the tree (node or leaf), for enabling the user to add details like performances, input and output data volumes or activation frequency.
ELISA ensures the consistency of the functions tree and the attached requirements tables; if some functions are renamed, moved or deleted in ASA tool, the corresponding objects in the repository are automatically changed; on the reverse, if the user deletes objects in the repository, he will receive inconsistency warnings.

Define logical sequencing scenarios:
Starting from functions and associated performance and data flows requirements, the designer usually defines a set of data processings and looks for their sequencing and synchronization constraints.
Functional analysis only gives a static view of the system which is not sufficient, the dynamic behavior is represented via "chronograms". Chronograms graphically express duration, start and end dates of each processing execution, in a given time. scale. Several chronograms are necessary to analyse nominal and critic paths of the system. This step allows to highlight possible parallelism, concurrency and synchronization constraints between processings. With the FrameMaker graphical toolbox, it is possible to create, edit as many chronograms as needed (exploitation chronograms, telemetry level n processing sequence..) and to link them to requirement tables and functions.
At any time, one can query the "technical memory" for estimation of some processing duration, by comparison with previous similar projects.

PHASE II: DESIGN OF MODELS OF THE DATA PROCESSING ARCHITECTURE

A key feature in system design is to predict system performances as soon as possible, in order to foresee system evolution ability, according to potential customer requests. System designers perform trade-offs between central processing, distributed, client/server or cluster architectures and have to propose the best one. Alternative solutions are often provided on designers experience basis or on hardware constraints. But few solutions are in fact really studied for a given project.

The objective of this phase is to come up with alternative models of the system hardware and software architecture, which all satisfy the requirements defined in the previous phase.

ELISA gives help in three inter-related activities:
- software architecture modeling,
- hardware architecture modeling and
- overall model validation.

The support to the whole phase relies on an integrated toolset for system modeling and performance evaluation: MODLINE. It is an open environment for modeling discrete event systems, developed and sold by SIMULOG (F). At the time the ELISA project started, no commercial tool with a satisfying high level user interface was available. MODARCH, a new tool has been added to MODLINE on CNES request, starting from an existing mock-up.
System designers are rarely familiar with formal technics (e.g. petri-nets) or with queueing networks, so they need to manipulate "macroscopic" and realistic components. With an ergonomic graphical object oriented interface, MODARCH let them manipulate and compose tasks, processors, networks, storage devices, terminals...
The tool relies on the queueing networks theory (QNAP2/ Queueing Network Analysis Package, from Inria and Bull F).
It must be kept in mind that the objective is not to monitor precisely the performances of a system, but to evaluate roughly the performances and sizing capacities of a future system. Most of its parameters and then results will be known in an approximative way, but in an acceptable margin, depending on the current project phase (A, B or C).

Model software architecture:

The activity consists of defining a model of the software application that answers to the functional architecture of phase I. For simplification purpose and demonstration of data sharing between integrated tools, ELISA implements a single concept: functions; this means that a function, a software, a processing or a task represents the same object in the environment.

From the function tree, the environment automatically extracts the leafs and generates the software tasks each time the user edits the MODARCH architecture.

ELISA again maintains consistency: the functional analysis may evolve, software architecture modifications will automatically follow, for example adding a function in ASA will add a task in MODARCH, but removing a function will generate a warning.

Automation is provided when possible, but the user is still free to work in inconsistent states, ELISA guides him in a predefined way but does not enforce him, at least he is warned.

The initial architectural model is composed of a set of independent tasks (names of leaf functions). The designer refines them by filling attributes (priority, memory) and activation conditions. The behavior of the tasks is defined with QNAP2 language and operations (read/ write in a storage, send message to other task, consume CPU, etc..). Four task types are provided: *sources* which allow to activate tasks by sending requests, *tasks* which execute some code on reception of requests, *in-out tasks* for modeling files, data bases, *exit* tasks for deletion of requests. This ends with an executable tasks network.

Model hardware architecture:

The architect now looks for a hardware configuration that satisfies the software application. Without leaving MODARCH, he selects components in an equipments data base (processors, storage devices, terminals, networks..). Each component is typed (a printer and a screen are of terminal type, the processor type includes workstations and mainframes), and its attributes can be instantiated with user defined values (memory, swap, CPU, rates...). The behavior of the hardware equipments is coded in QNAP2 and is hidden to the user, standard algorithms, systems (Unix) and communication protocols are available.

A major requirement towards MODARCH was the flexibility of the projection of software tasks to hardware equipments. With a simple graphical link, the user affects a task to a processor or to another. No user code is modified if the task is moved or if the storage device to which it sends write operations is attached to a remote machine.

The objective is indeed to analyse several solutions as easily as possible. The MODARCH user interface ensures consistency controls (a task can only be mapped on a processor, a source can be mapped on a terminal, an in-out task can be mapped to a terminal or a storage).

Each hardware and software component of a model can be parameterized (e.g. a CPU size, a message length, a task priority), the parameters values will be used for simulation purposes.

At any time, the user can call the other tools, in order to have all the system views in his screen. When he decides to close his model, the repository automatically imports the system components and the mapping links between software and hardware. New tasks will then be created and linked to the processor objects, existing ones (functions) will be updated with MODARCH information.

The user can add trace links between new objects and previous ones. The result is a graph of inter-related objects which allow direct navigation between activities (from a requirement table relating to function A to the description of the computer which runs the task A). The impact of a requirement evolution is thus immediately visible if the user lists the traced objects; the reverse is true, if a hardware equipment evolves, the user can control the impact on traced requirements or functions.

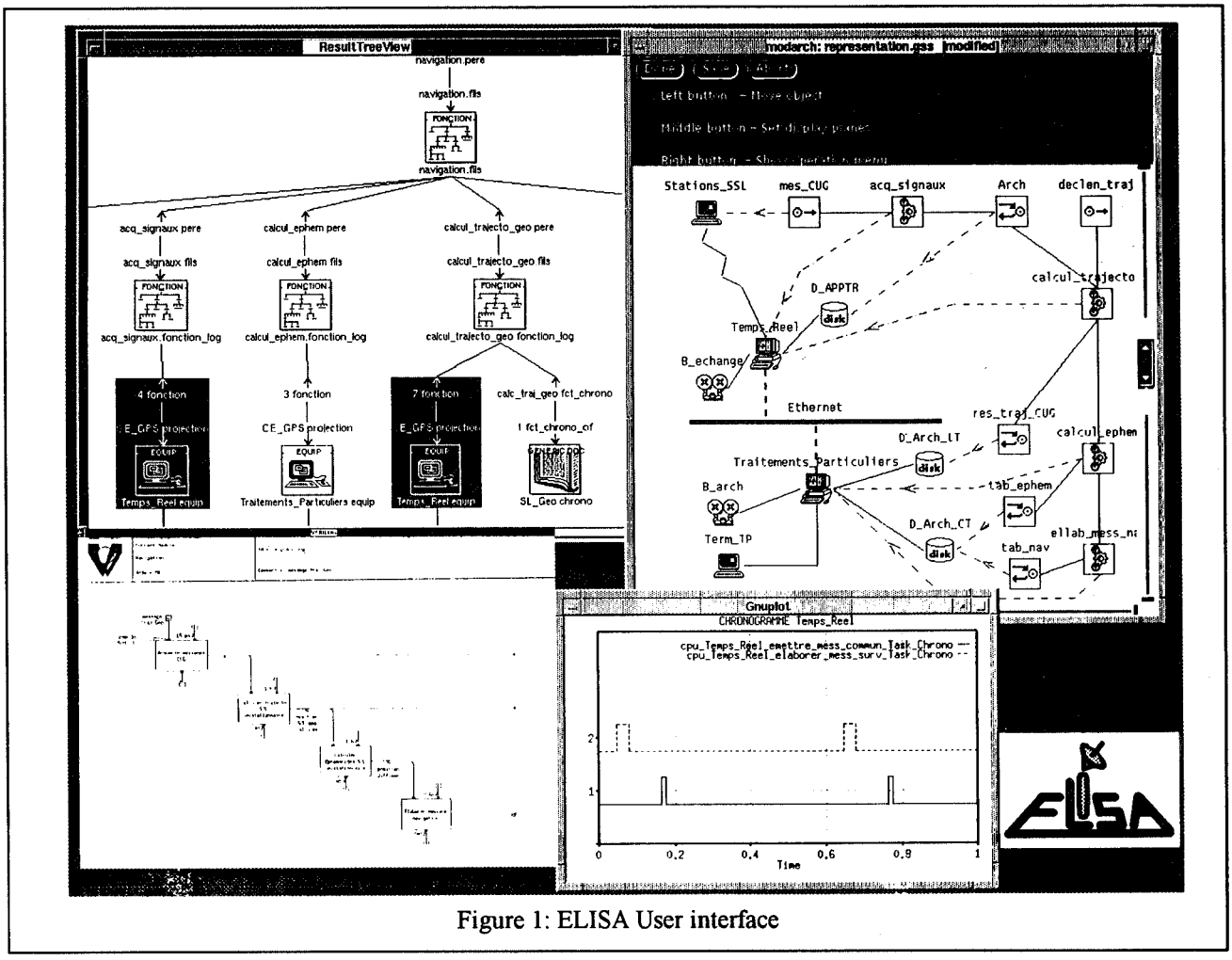


Figure 1: ELISA User interface

Validate the overall model:

The last activity consists of executing the simulation model and analyzing the results. The MODLINE toolset integration in ELISA is helpful.

Simulation scenarios (experiment plans) allow to define variation laws for the parameters values of the above model.

Via a single operation, MODLINE automatically checks, generates and compiles the code, executes and provides results. With a graphical results analyzer, the user can edit curvs or resources occupation chronograms. An animation tool allows to follow messages exchanges dynamically and then helps debugging of the model.

Each data is stored in the repository: scenarios, results, analysis drawings, links to its parent objects are set by the environment. Through the repository the user can compare output analysis chronograms and those issued in phase I, this assessment may help him choosing an architecture type or another.

The process is iterative, according to resources utilization analysis, the user can change his models until obtaining the better optimized one.

Figure 1 shows the ELISA user interface with views on the repository, ASA and MODARCH models and a chronogram.

PHASE III: ASSESSMENT OF PHYSICAL ARCHITECTURES

In the last phase, the designer chooses a real physical architecture after studying several implementations for each model (HP, SUN, IBM?). ELISA supports four activities:

- Selection of real equipments,
- Validation of physical architecture,
- Cost computation,
- Choice of physical architecture.

The tools are the same as above: MODARCH/ MODLINE, LOTUS 123 and Frame Maker.

Select real equipments. The architect selects existing physical equipments for each component of the chosen model. Specially, he chooses, via the MODARCH component base, a given workstation (e.g. SUN Sparc 10) or a given disk (e.g. Sundisk), and all their characteristics are updated. He may also refine the tasks behavior if necessary (e.g. Oracle for a data base model in phase II). When leaving the MODARCH tool, the repository is updated with new objects and links towards the origin model and functions.

Validate physical architecture. The validation of the physical architecture can be directly performed by using the scenarios of the source model. The user has to verify that the physical architecture still satisfies the system requirements.

Compute architecture cost. The objective is to provide an overall cost for the physical alternatives. By affecting a cost to all the components in MODARCH and with a LOTUS 123 integration, the user gets a table of costs he can complete and sum.

Propose an architecture. The end step is to give a proposition to the customer. With all information issued by previous steps (performances evaluation, requirements coverage, functional decomposition and costs estimation), the user can edit a Decision Justification Document with FrameMaker.

HORIZONTAL SERVICES

Documentation with Frame Maker.
Documentation is a very time consuming task, specially in early project phases. ELISA supports the user in composing and editing documents containing information produced by the tools. The "specification document" can be assembled in a semi-automated way: the IDEF0 graphics, the spreadsheet requirement tables, the chronogram drawings are imported in a synthetic document, from a predefined template. The user can yet complete and polish it before printing.

Configuration management.
The ELISA framework allows the user to manage versions of objects and to generate full or user-defined configurations of his project. Snapshots of his project will allow to stabilize versions of his work.

Administration tasks.
The administrator is responsible of the environment evolutions, of the repository management (save and restore operations), of feeding the technical memory, and of users and projects management.

ELISA ARCHITECTURE

The ELISA architecture is compliant with the one defined in the ECMA reference model [ECMA 91] and can be represented as in Figure 2.

ELISA runs on a Sun SparcStation 2, SunOS 4.1.3, Motif.
The PCTE repository is the Emeraude implementation.
ELISA has been developed by CISI S.A.

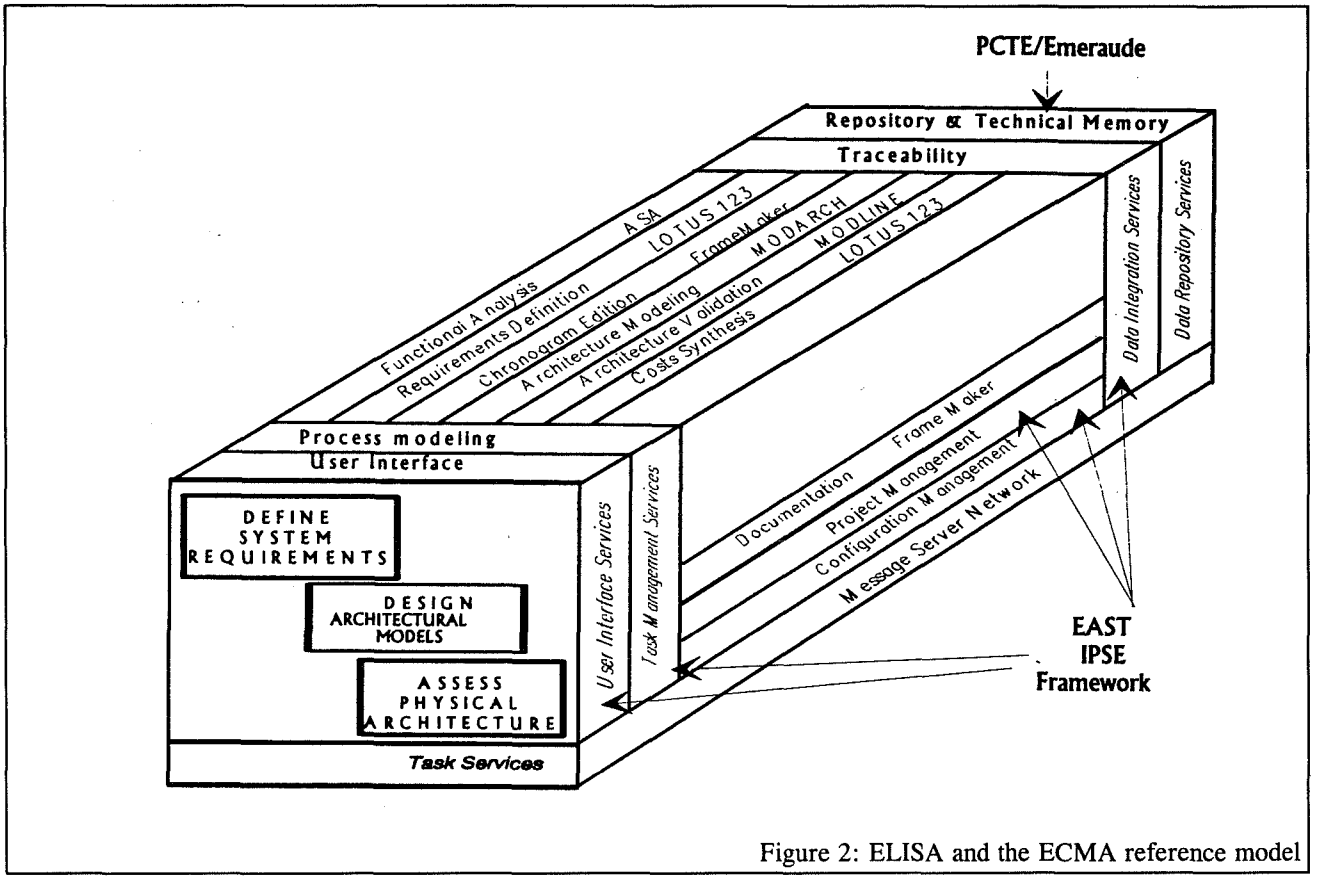


Figure 2: ELISA and the ECMA reference model

FIRST CONCLUSIONS

ELISA has been delivered in January 94. Architects have been trained to the demonstrator. Some demonstrations, at CNES and externally, have shown the public interest for the subject for the ELISA solution. Studies on system engineering environments currently raise in the european space and confirm our opinion.

The project allowed to complete our experience in tools integration, mainly in the impact of real data sharing between tools and the induced severe consistency checks.

The benefits of a system simulation tool are clear for the users. Moreover, with the ELISA environment, one can measure and better understand the benefits of integration like traceability, transparency of tools invocation, common services , and specially assistance to the generation of documentation.

ELISA has been delivered with an integrated real test case based on the CE-GPS project. It is currently being calibrated by Matra Marconi Space on the operational SPOT4 Satellite Control Center.

Some evolutions are going-on: porting on ECMA PCTE, moving to an operational environment, adding demonstrative features (multi-platform communications), integration of new tools version and process model enhancement.

References

[ECMA 91] ECMA, "A reference model for frameworks of software engineering environments", ECMA report n° TR/55.

ASA is a registred trademark of VERILOG
EAST is a registred trademark of SFGL
Emeraude is a registred trademark of GIE Emeraude
FrameMaker is a registred trademark of Frame Techn. Corp.
Lotus 123 is a registred trademark of Lotus Development Corp.
MODLINE is a registred trademark of SIMULOG
MOTIF is a registered trademark of OSF Inc.
SUN is a registred trademark of SUN Microsystems, Inc.
UNIX is a registred trademark of AT&T

111149 N95-17564

354230

9.8

Software Interface Verifier

Tomas J. Soderstrom, Laura A. Krall, Sharon A. Hope, Brian S. Zupke

Telos Corporation
320 N. Halstead, Suite 260
Pasadena, CA 91107
tomas.soderstrom@jpl.nasa.gov

Abstract - A Telos study of 40 recent subsystem deliveries into the DSN at JPL found software interface testing to be the single most expensive and error-prone activity, and the study team suggested creating an automated software interface test tool. The resulting Software Interface Verifier (SIV), which was funded by NASA/JPL and created by Telos, employed 92% software reuse to quickly create an initial version which incorporated early user feedback. SIV is now successfully used by developers for interface prototyping and unit testing, by test engineer for formal testing, and by end users for non-intrusive data flow tests in the operational environment. Metrics, including cost, are included. Lessons learned include the need for early user training. SIV is ported to many platforms and can be successfully used or tailored by other NASA groups.

1. Interface Testing History and Problem Statement

The Deep Space Network (DSN) Deep Space Communication Complex computer environment is highly distributed, with major functions allocated to subsystems. These subsystems are hosted in separate computers and communicate with each other and JPL via a LAN/WAN. All communications follow negotiated interface agreements which prescribe the communications protocols, data formats, and data ranges.

Over the past four years, JPL and Telos developers on the Telos DSN Task Contract fielded 40 subsystems into the DSN. Frequently, mission requirements forced subsystems to negotiate new interface agreements and to deliver asynchronously. The typical subsystem profile was:

- A telemetry, tracking, command, or supporting applications
- Communications and hardware intensive
- High reliability requirements
- 70K lines of C code, mostly realtime
- Six external LAN interfaces
- Development cost of $70K - $2M

The study team found interface testing to have been the most costly and error-prone activity. It proved nearly impossible to manually verify and all possible data ranges and data combinations for all interfaces during live tests. This was due primarily to excessive requirements for test equipment and test personnel in high demand. Consequently, interface errors sometimes were not detected until the subsystem was in operational use.

Metrics collected by the study team supported the high cost of testing. Typically, 3 - 10 attempts were necessary before the

average interface was successfully tested. End-to-end interface tests required from 5 - 12 personnel, and multiple tests were necessary. Programmers spent a total of 4 - 6 work months writing unplanned interface simulation code to support the test activity. In addition, they spent another 2- 4 work months per interface in creation and testing activities.

II. SIV Goals

The study also showed that overall testing accounted for a large part of the development effort of the 40 deliveries. This agreed with an Association of Computing Machinery study of seven large software projects, which found that 50% of the resources were spent on the overall test effort. The Telos study team estimated that a comprehensive, automated, reusable test tool could save 40% of the current interface costs. The team further found that 173 DSN interfaces could benefit from this tool within the subsequent five years.

What features would be needed in such a test tool? A literature search and interviews of personnel involved in testing found that the tool should:

- Understand DSN-specific protocols
- Be flexible and extensible, yet easy to use
- Test interfaces in an exhaustive but automated manner
- Provide both realtime visibility into the testing and off line results
- Be available in time to prototype interface agreements
- Support developers' unit testing
- Support test engineers' formal testing
- Support DSN end-users' application simulation and data flow testing

In addition, the test tool should combine three types of test tools and have the following specific capabilities:

1. *Generate Test Data*
   - Control data to the bit level
   - Produce static, variable, and predicted dynamic data
   - Simultaneously run in batch mode and interactively
   - Send single data blocks at specified times and intervals
   - Send data blocks or streams to multiple destinations

2. *Capture and Compare test data*
   - Specify which streams to capture and compare to expected results
   - Specify expected data values and ranges
   - View automatic comparison of test data to expected values both on- and off line
   - Mask out data which would not require an exact match

3. *Simulate the entire application*
   - Create, run, and repeat complete application scenarios for multiple interfaces of multiple subsystems
   - Interactively change the behavior of the simulated, scripted application
   - View online and printed detailed results

Telos proposed the Software Interface Verification (SIV) tool with all the above functionality. It was to be rapidly developed and fielded with increased functionality provided in two subsequent deliveries. SIV was funded by NASA/JPL and developed by Telos. The SIV provides all the functions

listed above listed above and summarized in Figure 1.

simulate an application session, such as sending the data from a typical Telemetry pass.

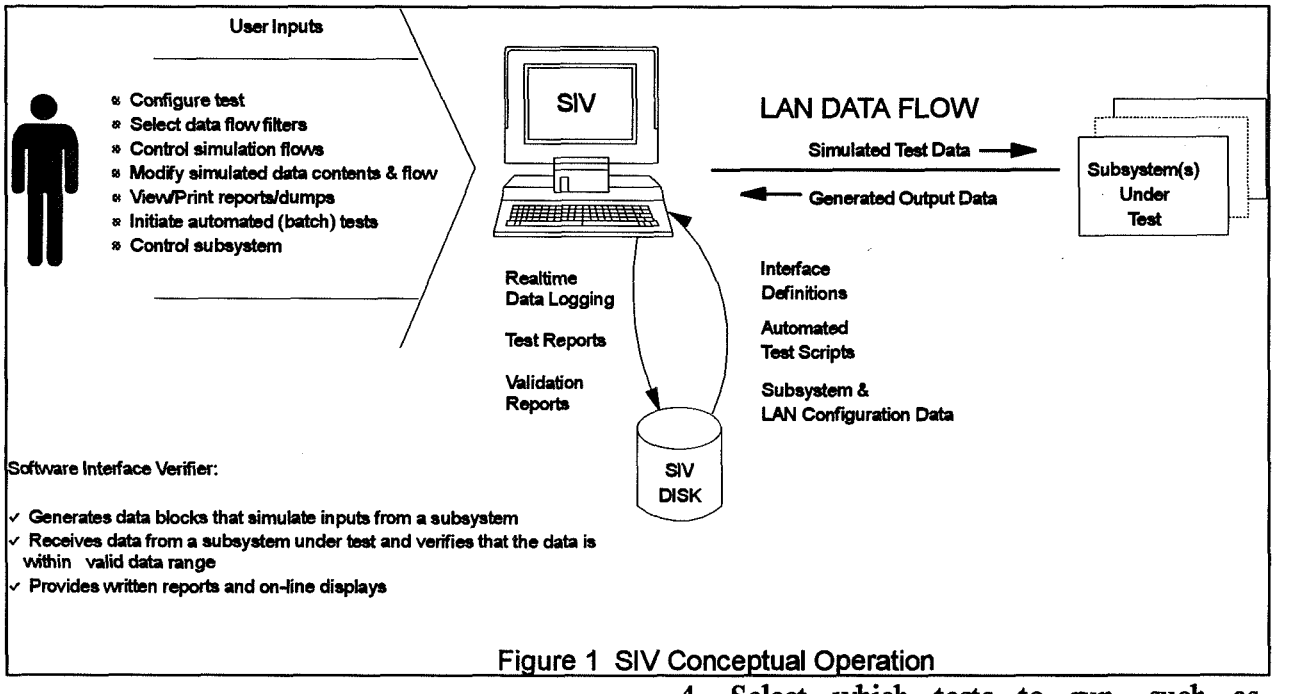


Figure 1 SIV Conceptual Operation

The following steps summarize the typical SIV user scenario.

1. Create an ASCII table describing the interface agreement (called a Rapid Interface Definition - RID). It contains interface definitions, including data types and minimum/maximum/expected values, incrementing values, etc. (In the next SIV version, this will be automatically created from the interface agreement. For now, it must be typed in once.)

2. Download the RIDs to SIV from LAN or floppy and select which RIDs to use via a type-in.

3. Select or create application simulation scripts, if desired. This will enable SIV to

4. Select which tests to run, such as generating test data, logging and comparing test data, and/or simulating entire applications.

5. Select which online displays to view (detailed data dumps, overall status monitoring, or none).

6. Start the tests and, as desired, interactively start/stop/modify the data flows via SIV type-ins.

7. When the test is complete, or manually terminated, print the test report or download it via LAN or floppy. Note that the tests can be set up to cycle indefinitely.

III. SIV Development

The SIV development team consisted of one technical lead who interfaced with the users plus one programmer and one half-time tester. The primary obstacles to be overcome were:

- Users' reluctance to use an unproven test tool
- Requirement to support multiple operating environments
- Limited budget
- Quick results needed to meet users' schedules

In order to meet the budget, time, and multiple operating environment constraints, the development team reused a working skeleton subsystem from the Multiuse Software reuse library, which had been previously created by Telos and had already been ported seven hardware/operating system platforms. In addition, existing test software from other development efforts was adapted for use within SIV.

To overcome the users' reluctance to learn and trust new test tools, the technical lead concentrated on frequent communication with potential users. This included electronic mail, phone calls, visits, demonstrations, and presentations. In addition, the team solicited feedback and carefully folded new user requirements into subsequent demonstrations. This convinced skeptical users by providing them continual visibility and input into SIV development progress and capabilities.

Although SIV was created as a DSN-specific test tool, it was developed in a layered fashion to facilitate later porting. This could include adding new protocols, porting SIV to new hardware/operating system platforms, changing the user interface, and adding/changing SIV functionality. Figure 2 describes the SIV software architecture and major functionality. For example, to incorporate a new, low-level LAN protocol, only the LAN Protocols module of Multi-use Software need change.

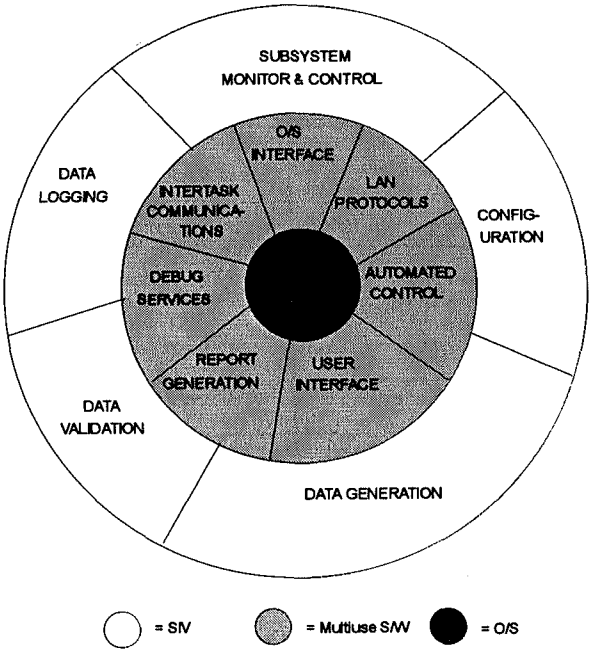


Figure 2. SIV Software Architecture

IV. SIV Results

SIV's primary goal was to reduce the cost of interface testing and the number of software interface errors in the DSN. To achieve this goal, skeptical users had to be convinced that using SIV would save them time. We originally hoped that cost savings due to SIV usage would exceed SIV total lifetime costs ($420K) during SIV's second year of use (1996).

As Figure 3 shows, the goal to obtain user acceptance was met with a wide margin. SIV was initially targeted for use by only 13 projects, or user groups, during the 1994-1999 time frame. However, within the first seven months of development, and one month after the first release, SIV had acquired 23 interested user groups, 10 of which have already used the SIV.

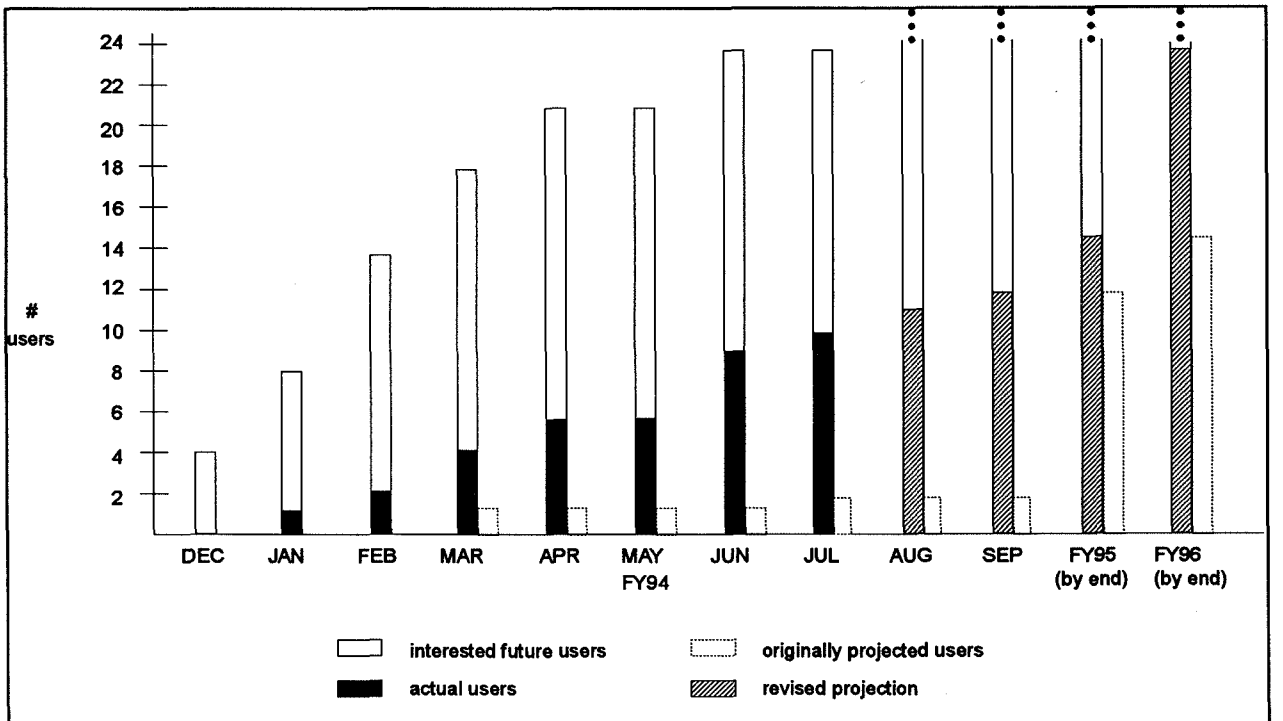


Figure 4 SIV User Interest and Involvement has Surpassed Original Goals

In these 23 user groups, there are now three distinct types of SIV users:

- Developers - use SIV to unit-test low-level interfaces in their development laboratories.
- Test engineers - use SIV to performance/stress test their applications at DSN's official test facility.
- End users - use SIV to simulate entire subsystems for data flow tests, for training, and for simulating hard-to-create error conditions at the official test facilities.

Metrics have been collected for three months: two months before official SIV release, and one month following the release of version 1. The metrics support the anticipated savings as well as ones not originally considered.

The following relates some specific user reports:

- The Metric and Pointing Assembly (MPA) group saved 50 development hours otherwise needed to write simulation code to test a new interface

which would not be available until well after MPA delivered.

- The Central Monitor and Control group reported saving 20 work hours because of SIV's ability to insert predicted errors in the interfaces. This would have otherwise taken several weeks and multiple 350-mile round-trips to the DSN station to induce the interface errors, test whether the assembly reacted correctly, and return to make needed software corrections.

- Multi-use Software saved 80 hours of dedicated Test Facility resources and associated travel by using the SIV in their development laboratory to identify and correct a complicated software anomaly.

So far, SIV users have detected and corrected the following types of errors in their applications, without the need for live tests: formatting errors, data range errors, routing problems, and errors due to misinterpretations of interface agreements.

The metrics listed in Table 1 represent three months of SIV usage by six user groups.

Let us tackle the difficult process of estimating cost savings achieved. Of the six user groups, an average of three user groups were concurrently using the SIV each month for a three month period. To estimate cost savings achieved, let us assume the DSN average development cost (including burden charges) to $67 per work hour and test facility usage to $200 per work hour (including support personnel, rent, hardware maintenance, etc.). These values when combined with the savings in the above table result in a total savings of $48.6K for the three months or $5.4K per user group per month. Applying the $5.4K to our projected Fiscal Year 1994 (FY94) and FY95 users (see Figure 3), results in a total cost savings of $216K for FY94 and $875K for FY95. This exceeds our originally projected cost savings of $51K for FY94 and $324K for FY95. In more general terms, this minimally translates into the developer having more time to work on other subsystem development areas. It also means more available test facility time to other users. Overall, SIV usage should significantly reduce the risk and cost of the typical DSN subsystem delivery.

Table 1. Initial SIV Metrics

SIV Usage	Metric
# Subsystems Interfaces Tested	11 interfaces
# Data Flows Tested	20 data flows
# Interface Definitions Generated	63 RIDs
# Code/Interface Errors Discovered	24 errors corrected
Est. Test Facility Time Saved	146 work hours saved
Est. Additional User Time Saved	190 work hours saved
Est. Simulation Code Time Saved	100 work hours saved
Est. SIV Learning Curve Total Cost	10.5 work hours invested

Additional savings due to automated testing using SIV include:

- Reduced amount of travel to -- and use of expensive -- Test Facility
- Faster turn-around times when testing within development labs--no need to wait for scheduled test times or personnel availability
- Costly simulation code need not be generated nor maintained

- Fewer end-to-end test resources required since data content and protocol routing can be pre-verified with SIV
- Automated regression tests can be run at computer speed

Although the initial SIV version has just been fielded, early results clearly indicate the value of automated testing and that SIV met its goals and will help test DSN interfaces at all levels. Developers, test engineers, and end users no longer have to be "sold" on using automated test tools such as SIV. The early results indicate that automated testing will continue to pay dividends.

V. Lessons Learned

What did we do right?

We solicited user acceptance. The SIV Technical Lead spent a considerable amount of time with skeptical users to learn their test and simulation needs and teach them SIV.

We held early and frequent demonstrations. These also allowed for design refinement and identification of new requirements. When acted upon, this was especially important as it created user acceptance.

We selected an experienced staff. The developers, who were experienced with the reused packages and testing in the DSN environment, experienced no learning curve.

We employed significant reuse. The completed SIV consists of 8% (or 8K lines) application-specific code and 92% reuse from Multiuse Software and adapted simulators and test software obtained from a reuse depository. Besides for helping speed up the SIV development, the reused software had been previously proven, extensively tested, and ported to seven platforms.

What could we have done better?

We should have allocated more schedule time to the demonstrations. Although invaluable for the eventual SIV progress, the cost of each demonstration was 3-5 work days to plan and hold plus 3 work days for user requirements change requests, follow-up, and action items.

We should have provided earlier user training. This would have lessened the drain on SIV personnel for user support which we under-estimated.

We should have held smaller training classes customized to the group's needs. This would have allowed more customized training to better enable the users to recognize and use the powers of simulation and automation that SIV possesses.

VI. Applicability For Other Groups

SIV can be successfully used on all large, distributed software development efforts where computers interface over a LAN. Although standards, such as the Distributed Computing Environment, and Abstract Syntax Notation, have great promise, they are often too late to immediately benefit current, large software environments. The SIV is a flexible test and simulation tool which can test other subsystems over a LAN. It can be easily adapted to use new custom or standard high- or low-level protocols.

SIV is written in C and currently runs on a Sun under the Solaris operating systems and on Modcomp's Unix work stations running

the Real/ix operating system. It can easily be adapted to run on all other platforms supported by Multiuse Software (PDOS, VxWorks, VADSWorks, and OS/2). It is currently being ported to run on Intel 80386 computers (and greater) running a shareware Unix variant called Linux. SIV is fully documented and available from Telos or JPL by request to the authors. We plan to implement TCP/IP during Fall/Winter 1994, which should make the SIV instantly usable by groups outside the DSN.

Acknowledgments

The work described in this paper was accomplished by Telos Corporation under contract to the Jet Propulsion Laboratory, California Institute of Technology and sponsored by the National Aeronautics and Space Administration.

We would like to thank Joseph Wackley, Roger Crowe, and Sheila Davis of JPL for their support of Multiuse Software and the Software Interface Verifier project and for their vision and relentless efforts to create an automated testing environment in the DSN.

Systems Development

3. Methods

** Presented in Poster Session*

N95- 17565

111150

354235

OOD/OOP Experience in the Science Operations Center part of the Ground System for X-ray Timing Explorer Mission

Abdur Rahim Choudhary

Hughes STX, Technology Applications Group, 7701 Greenbelt Road, Greenbelt, Md-20770, USA. (301-441-4229), rahim@rosserv.gsfc.nasa.gov

1.0 Introduction

The Science Operations Center (SOC) for the X-ray Timing Explorer (XTE) mission is an important component of the XTE ground system. Its mandate includes:

- Command and telemetry for the three XTE instruments, using CCSDS standards.
- Monitoring of the real-time science operations, reconfiguration of the experiment and the instruments, and real-time commanding to address the targets of opportunity (TOO) and alternate observations.
- Analysis, processing, and archival of the XTE telemetry, and the timely delivery of the data products to the principal investigator (PI) teams and the guest observers (GO).

The SOC has two major components: the science operations facility (SOF) that addresses the first two objectives stated above and the guest observer facility (GOF) that addresses the third. The SOF has subscribed to the object oriented design and implementation; while the GOF uses the traditional approach in order to take advantage of the existing software developed in support of previous missions.

This paper details the SOF development using the object oriented design (OOD), and its implementation using the object oriented programming (OOP) in C++ under Unix environment on client-server architecture using Sun workstations. It also illustrates how the object oriented (OO) and the traditional approaches coexist in SOF and GOF, the lessons learned, and how the OOD facilitated the distributed software development collaboratively by four different teams. Details are presented for the SOF system, its major subsystems, its interfaces with the rest of the XTE ground data system, and its design and implementation approaches.

2.0 Distributed Development

SOF development is distributed from following points of view:

- Development by a team with components distributed at Hughes STX and the three PI team locations at Goddard Space Flight Center (GSFC), University of California at san Diego (UCSD), and Massachusetts Institute of Technology (MIT). It also implies development under heterogeneous management structures, as each team component has its own management.
- Development on computer systems distributed at above team component locations, and internetworked using TCP/IP. This also includes development on heterogeneous types of machines.

PRECEDING PAGE BLANK NOT FILMED

PAGE 962 INTENTIONALLY BLANK

SOF development uses the incremental build approach, with builds roughly six months apart. It employs the philosophy that the system software will be so modularized that the modules can be developed by the components of the team that has best expertise for them. Thus the software development related to a particular instrument is allocated to the corresponding PI team. These include the instrument health and safety, instrument commands, instrument telemetry unpacking algorithms, and algorithms to construct physically meaningful data partitions from the telemetry.

The rest of the system development is performed by Hughes STX. This includes the overall system engineering, development of abstract classes and base classes, integration of the total software system, testing of the system and the subsystem components, and integration of the SOF with the rest of the XTE ground system. The overall responsibility for the SOF remains with Hughes STX. This includes coordination with the various teams, clear definition of the development interfaces, and meeting the software build schedules.

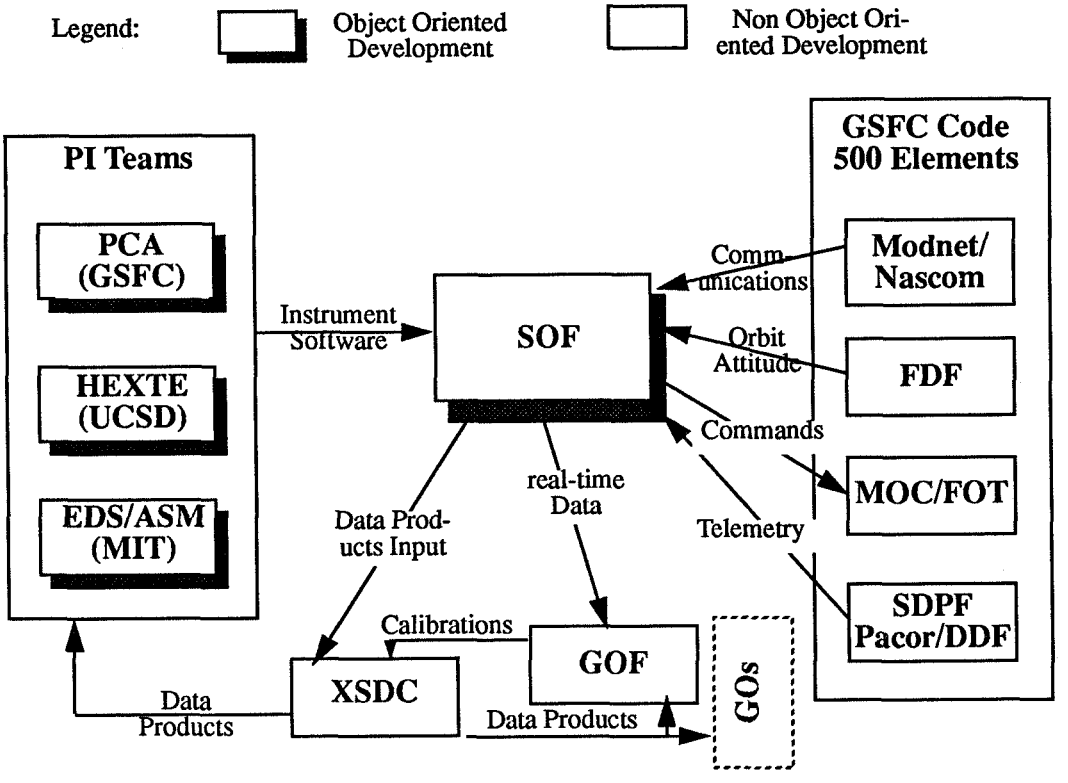


Figure 3-1: SOF interfaces and context within the XTE ground segment

3.0 SOF Context and Interfaces

Figure 3-1 shows the distributed parts of the SOF development effort together with the relationship of SOF with the rest of the XTE ground system. The SOF box shown in the center represents the net result of object oriented development by the PI teams and the Hughes STX. It has important interfaces with other ground system elements which are not object oriented. The GOF is not object oriented, but it needs to retrieve telemetry data

products from the SOF generated objects in order to generate the Flexible Image Transportation System (FITS) files. This interface is provided by data management subsystem of the SOF (see Fig. 4-1) that communicates with the XTE FITS Formatter software of the GOF using a set of data descriptors formulated according to a data description language (DDL) defined by the GOF for this purpose.

SubsystemConfig

This is the base class for the `DesiredConfig` and `PredictedConfig` classes.

Subclasses

`DesiredConfig`
`PredictedConfig`

Attributes

`RWCString configurationName;`
The configuration name.

`RWCString description;`
A descriptive string for the configuration.

Public Constructors

`SubsystemConfig ();`
Constructs a configuration with no description or configuration name.

`virtual ~SubsystemConfig ();`
Destructor.

Public Member Functions

`void setConfigurationName (const char* name);`
Sets the configuration name.

`const char* getConfigurationName() const;`
Returns a pointer to the configuration name.

`void setDescription (const char* name);`
Sets the descriptive text field.

`const char* getDescription() const;`
Returns a pointer to the description,

Virtual Member Functions

`virtual const char* getSubsystemName() const;`
Returns the name of the subsystem.

`virtual CommandScript* getCommandScript() const;`
Returns the command script.

`virtual TelemRate* getTelemRate (const Source& source) const;`
Returns a telemetry rate.

`virtual void printShort (ostream& ostr) const;`
Prints a description of the configuration.

`virtual void print (ostream& ostr) const;`
Prints a description of the configuration.

`virtual void printLong (ostream& ostr) const;`
Prints a description of the configuration.

Figure 3-2: An example of detailed class prototype from command generation subsystem.

The non Object Oriented interfaces are defined in traditional sense. All the data to be exchanged between SOF and an element of the ground system were identified; their formats were specified; the frequency and mode of each data transfer and the corresponding data volume was determined; and the standards to be adhered to were noted. A separate

ICD was concluded between SOF and the corresponding ground data element (as opposed to a single ICD between SOF and all other elements). This approach allowed the logistic complexities to be minimized and updates to these ICDs manageable by keeping the number of involved parties small.

The interfaces between the SOF and the components of the SOF to be developed by the PI teams were necessarily object oriented. The traditional methods for the interface treatment could not be employed in this case. To define the object oriented interfaces, first the class hierarchy was developed. The base classes were all allocated for development by the Hughes STX team. The subset of derived classes to be implemented by the PI teams were specified. The interfaces were defined in terms of the public member functions that these classes were required to support. As part of the interface definition, all such classes were prototyped; and those public member functions of each class were also prototyped upon which the other party depended for the implementation of their code. This set of prototype classes and public member functions were formulated early in the development and documented in an ICD. An example of such prototype class and its methods with their signatures is given in Fig. 3-2.

Separate ICDs were developed with each PI team. Further, the commonality between the ICDs with PI teams was explicitly acknowledged to facilitate their development, to avoid reinventing the parts already developed, and to manage the configuration of the common interfaces. This further facilitated the interface implementation, since the commonality explicitly formulated in the ICDs allowed the re-use of the corresponding software development approach among the PI teams.

4.0 Analysis and Design Approach

The book "Object-Oriented Modelling and Design" by Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenson, W. (Prentice Hall 1991) was used by the SOF team to follow the Object Modeling Technique (OMT) advocated by these authors. The following procedure was found useful and worked for the SOF team, even though the various steps described below were often concurrently analyzed and subsequently refined via iterations.

1. The SOF team started with the usual requirements analysis. The requirements are sourced from the customer, domain experts, and the users.
2. The requirements were allocated to a set of high level functions. These functions were grouped into the subsystems, shown in Fig. 4-1. A lead engineer was appointed for each subsystem. The analysis described below was performed on subsystem basis.
3. The nouns used in the requirements allocated to a subsystem were potential objects. After the redundancy was weeded out and the overlap between the objects was minimized, the team had a fairly good starting set of the objects.
4. The associations between the objects can be indicated by the verbs in the requirements definition. This led to some objects being identified as the class attributes. The dynamic modelling scenarios were used to identify the objects that potentially form the member functions.The initial objects set was thus grouped into a set of classes, their attributes, and member functions.

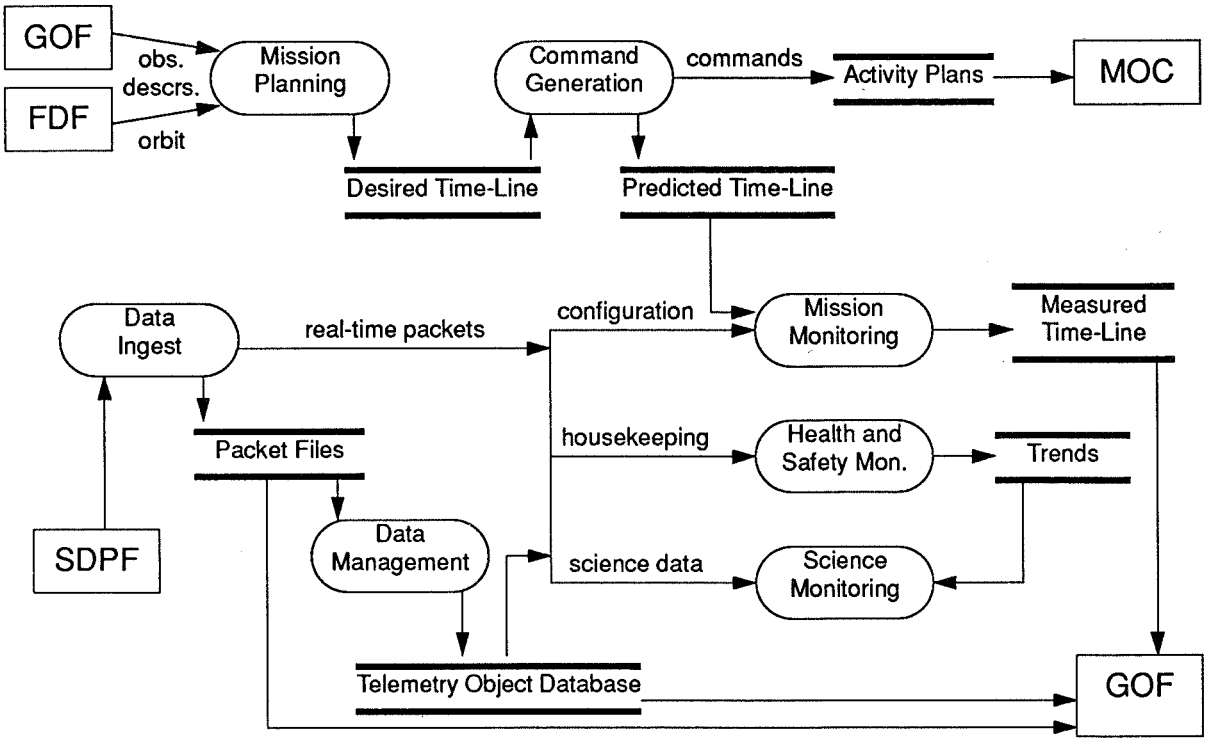


Figure 4-1: SOF software subsystems

5. A further analysis of these classes based on the bottom up and top down approaches was used to develop inheritance relationships between classes. The classes were then generalized to form the abstract classes; various specializations of which led to the derived classes. Some classes in each subsystem fell in the domain of expertise of the PI teams. Those were allocated for development by the PI teams. Such allocations however were not rigid so that they were reviewed as the design progressed and during the implementation phase of various builds.

Figure 4-2 shows an example of the object model for the command generation subsystem. The SOF design document has such object models for each subsystem and additional information as follows:

1. Subsystem introduction
2. Applicable requirements
3. Operating scenarios
4. Outstanding issues
5. Major design features
6. External interface
7. Subsystem interfaces
8. Subsystem object model
9. Subsystem class hierarchy
10. Detailed class design
11. Review comments and responses

The detailed class design is similar to the example presented in Fig. 3-2.

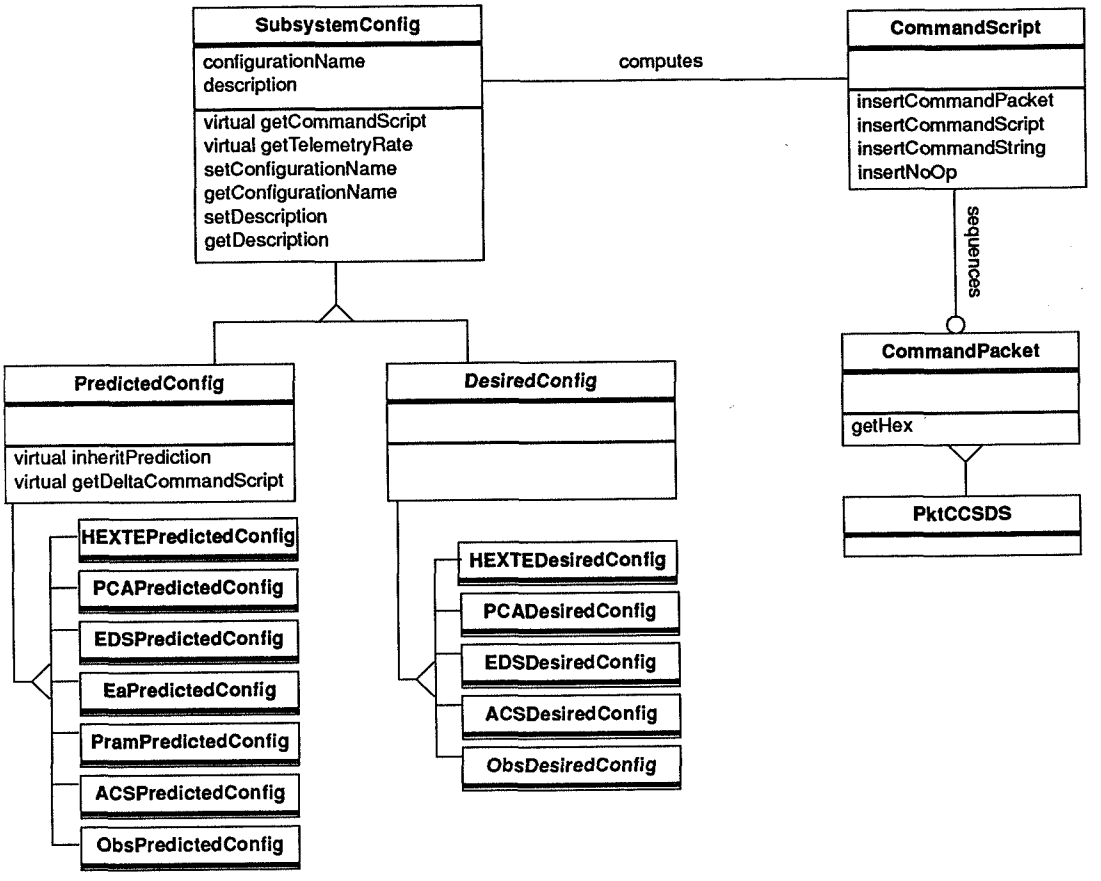


Figure 4-2: An example of Object Model from command generation subsystem

5.0 Development Environment

SOF decided for a client-server architecture using SunSparc workstations. However MIT wanted to use their existing DEC Ultrix workstations for their part of the SOF development. This meant that all development standards and the tools needed to be available on these two machines. To keep the development away from specific features of these two machines, a SGI IRIS Indigo was acquired to test that the software built on a third platform. The software development environment of the SOF are summarized in Table 5-1.

The internet connectivity between the computers on the four sites facilitated the distributed software development by the three PI teams and the Hughes STX. This allowed the developers to collaboratively debug problems on each others' computers using remote logon. It also allowed the periodic deliveries of the software and documentation from the PI teams to the Hughes STX for the SOF builds. Monthly meetings of all four components of the team were held. Other collaborations were ongoing using electronic mail. Each item in table 5-1 and all upgrades were discussed using these forums and kept in a standards document. Copies of all XTE SOF documents were available via an anonymous ftp account.

TABLE 5-1: Software development environment of SOF

Software Tool	Starting Version	Current Version	Comments
SunOS	4.1.3	4.1.3	Sun Operating System
Motif	1.1.4	1.2	GUI
ObjectCenter	2.0	2.0.6	C++ Debugger
CFront	3.0	3.0	AT&T C++ translator
Linpack/Lapack .h++	Linpack. 1.2	Lapack 1.0	Math.h++ and Maix.h++ supersets
xmgr	2.10	2.10	Oregon grad. inst. analysis package
RogueWave tools.h++	5.2	6.0	C++ library
RCS	5.6.0.1	5.6.0.1	Revision control system
TAE+	5.2	5.3	GUI Builder
X11	R5	R5	X Windows
FrameMaker	3.1x	4.0	Wordprocessor plus graphics
genman/genman++	2.0	2.0	Unix style man pages
GNU Make	3.68	3.70	make utility
Purify	2.1	2.1	check memory leaks/corruption
xteprob (home grown)	1.0	1.1	descrepancy tracking system

6.0 Object Persistence

Commercial object oriented data base management systems (OODBMS) were initially investigated for use in SOF. Ontos was selected for detailed evaluation. A pathfinder analysis showed that in the SOF context Ontos had several difficulties: presence of memory leaks, the performance limitations (SOF is required to ingest at an average rate of 64 kilo bits per second and a peak of one mega bits per second), and the fact that Ontos persistence mechanism required modifying those class library header files which must be persistent.

SOF's main data management needs are object persistence and persistent object retrieval. The more advanced features of an OODBMS such as sophisticated query capabilities or the transaction commit mechanisms are not required. The RogueWave (RW) Tools.h++ class library offers a means of making objects persistent. The UNIX file system together with the Dictionary classes in RW offer a means of accessing persistent objects; i.e. a way to simulate a global namespace. A prototype of the archival portion of the Ingest subsystem using Rogue Wave Tools.h++ was roughly ten times faster than the equivalent Ontos version. SOF therefore decided to develop internally the mechanisms it needed to satisfy many of its data management requirements.

7.0 Object Oriented Implementation

Some practical experiences during SOF implementation are presented in this section. The development was facilitated by early implementation of the object oriented interfaces. As can be seen from the example in Fig. 3-2, these interfaces were defined in terms of the method pro-

totypes in C++. The crucial parts of the code were therefore developed and scrutinized early in the process. As illustrated in Fig. 3-2, many of the interfaces were defined as virtual methods. This was very helpful in developing software with complete reliance on the PI teams for their instrument expertise and without the need for the Hughes STX engineers to also acquire such expertise. In fact the virtual method interfaces were often identically defined with each of the three PI teams; the specific instrument expertise were encapsulated in the way these interface methods were overridden by an individual PI team. At the same time the formulation presented a uniform interface to the Hughes STX engineers that were independent of the intricacies of the individual instrument subsystems. This approach is taken in many important instances including instrument configurations specification for command generation and mission monitoring, the telemetry unpacking to recover CCSDS packets, to assemble CCSDS packets into physically meaningful partitions, and to access that information from the persistent objects. This is a remarkable advantage of polymorphism in object oriented approach. The class hierarchy in such cases is illustrated in Fig. 4-2 for the case of instrument configurations. In this case the subclasses of PredictedConfig and DesiredConfig (except those for ACS and Obs) are developed by the PI teams while the rest are developed by the Hughes STX. The PI teams are free to derive their own sub-hierarchy.

The C++ templates were helpful. The real-time data ingest subsystem has a real-time server that passes CCSDS packets to the real-time clients. A real-time client template was developed that proved useful for the PI teams, the health and safety subsystem, the science monitoring subsystem, and the mission monitoring subsystem to write their own real-time clients.

The RW object oriented libraries of Tools.h++ proved very useful in saving the development effort on mundane things. The RW persistence and retrieval mechanism however was sometimes difficult for new developers to grasp.

8.0 Conclusion

Our OOD/OOP experience in SOF can be summarized as follows:

- Initial analysis and design activity took a while (the team was also passing through a learning phase); but the implementation proceeded pleasantly fast (couple of experienced C++ programmers later came on board, and the example of their work was helpful for the rest).
- Our decision not to use an OODBMS proved right.
- The COTS object oriented libraries saved SOF time and cost.
- Design changes due to the management decisions and requirements evolution were gracefully accommodated.
- The total SOF team is 11 persons, which is modest compared to similar past missions. XTE launch is scheduled for August 1995; the OOD/OOP approach has so far allowed SOF development on schedule and within cost.

MISSION OPERATIONS DEVELOPMENT: A STRUCTURED APPROACH

Michael Fatig
AlliedSignal Inc.

Paper Not Available

THE COOPERATIVE SATELLITE LEARNING PROJECT: SPACE MISSIONS SUPPORTING EDUCATION

Michael Fatig
AlliedSignal Inc.

Paper Not Available

PRECEDING PAGE BLANK NOT FILMED
PAGE 972 INTENTIONALLY BLANK

11151

N95-17566

354237

P-9

A Proven Approach for More Effective Software Development and Maintenance

Rose Pajerski
NASA Goddard Space Flight Center, Code 552
Beltsville, Maryland

Dana Hall
Science Applications International Corporation
McLean, Virginia

Craig Sinclair
Science Applications International Corporation
McLean, Virginia

Abstract

Modern space flight mission operations and associated ground data systems are increasingly dependent upon reliable, quality software. Critical functions such as command load preparation, health and status monitoring, communications link scheduling and conflict resolution, and transparent gateway protocol conversion are routinely performed by software. Given budget constraints and the ever-increasing capabilities of processor technology, the next generation of control centers and data systems will be even more dependent upon software across all aspects of performance. A key challenge now is to implement improved engineering, management, and assurance processes for the development and maintenance of that software; processes that cost less, yield higher quality products, and that self-correct for continual improvement evolution.

The NASA Goddard Space Flight Center has a unique experience base that can be readily tapped to help solve the software challenge. Over the past eighteen years, the Software Engineering Laboratory within the Code 500 Flight Dynamics Division has evolved a software development and maintenance methodology that accommodates the unique characteristics of an organization while optimizing and continually improving the organization's software capabilities. This methodology relies upon measurement, analysis, and feedback much analogous to that of control loop systems. It is an approach with a time-tested track record proven through repeated applications across a broad range of operational software development and maintenance projects.

This paper describes the software improvement methodology employed by the Software Engineering Laboratory, and how it has been exploited within the Flight Dynamics Division within GSFC Code 500. Examples of specific improvement in the software itself and its processes are presented to illustrate the effectiveness of the methodology. Finally, the initial findings are given when this methodology was applied across the mission operations and ground data systems software domains throughout Code 500.

Introduction

A recent analysis conducted by the NASA Software Engineering Program found that over 30% of the NASA Goddard Space Flight Center (GSFC) Code 500 civil servants and support contractors spend the majority of their time directly involved

PRECEDING PAGE BLANK NOT FILMED

PAGE 974 INTENTIONALLY BLANK

in the management, development, maintenance, and/or assurance of software (Reference 1). That represents over 1600 people out of the total of 5000 GSFC Code 500 civil service and support contractor community. Correspondingly, that same analysis found a tremendous investment in developed, operational software throughout the Mission Operations and Data Systems Directorate. Not including common off-the-shelf varieties of shrink-wrapped word processors, spreadsheets, and other typical tools, GSFC Code 500 is responsible today for some 21 million lines of operational code. This represents almost half of the 43 million lines of code currently operational throughout GSFC (Reference 2). Most of that is in one way or another involved in the preparation for, conduct of, or results analysis from spaceflight missions.

Given the importance of software in much of what the Mission Operations and Data Systems does and hopes to do, its not surprising that more attention is being paid to software; the tools and practices by which it is engineered; the management oversight by which it is coordinated and paid for; and the means by which products, tools, and know-how are disseminated and shared. The NASA Software Engineering Laboratory (SEL) and its software improvement methodology is a premier example of an attempt to understand the roles of software within GSFC Code 500 and to identify and promote practices that real experience shows are effective and beneficial.

Software Engineering Laboratory (SEL)

The Software Engineering Laboratory (SEL), located in the Flight Dynamics Division of GSFC Code 500, was developed to study the effectiveness of new software engineering technologies as part of the existing Code 500 software development projects. The SEL is a 300 person organization charged with producing operational flight dynamics software for each GSFC space mission, but it is also an organization that has intentionally and carefully for eighteen years experimented with languages, tools, and techniques to continually improve its software development and maintenance process. Within the SEL, every software project is considered to be an "experiment" where a new software technology is injected, its effectiveness measured, and if it proves useful the new technology is incorporated into the software development processes for the next project. The SEL organization works as a partner with the production organization's software developers to incrementally improve the software and its processes over time.

Figure 1 illustrates the three key components of the SEL environment that are critical to the success of improving an organization's software development process and software products. The first component is the development organization. This component is responsible for the software development of a real missions operations or ground data system application. This organization develops the software and documentation using the processes provided by the analysis organization. The development organization also provides software measurements, project characteristics, and lessons learned to the analysis organization.

The second component is the analysis organization. It uses the software measurements and project data to understand the developers' software and software process characteristics well enough to propose an improvement goal, analyze the effectiveness of the improvement, package the results, and feedback the result to the development organization for use in the current and future development efforts. The analysis component interacts with the development component to extract, examine, and compare the consequences of applying

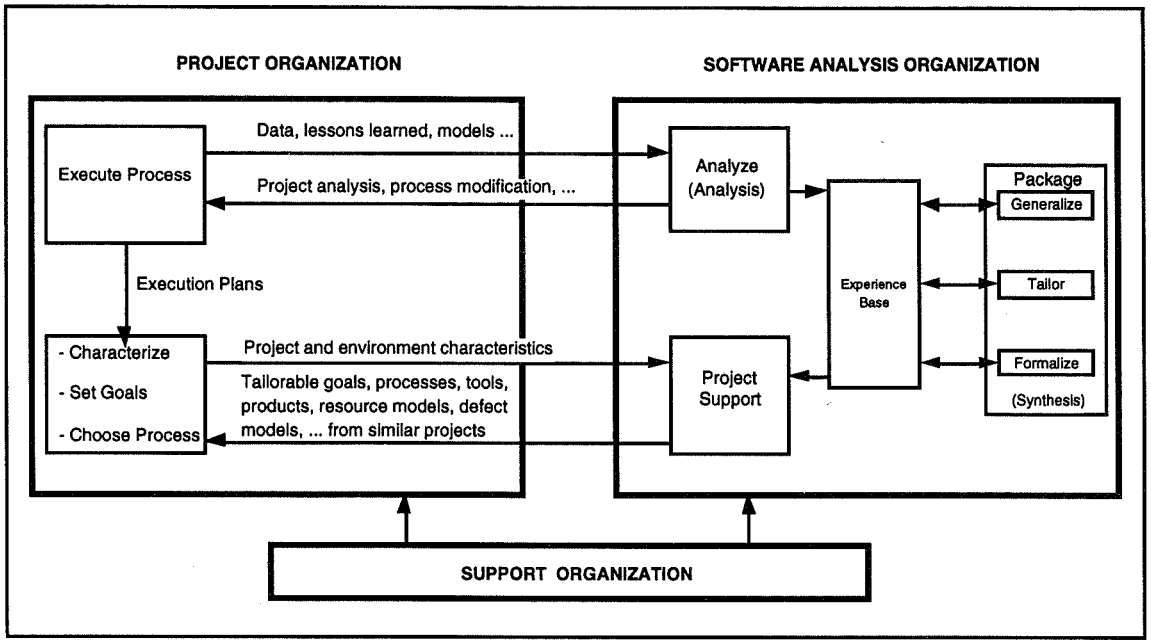


Figure 1: Components of the Software Engineering Laboratory Environment

specific methodologies, standards, and tools.

The third component is the support organization, which archives the information captured from the development and analysis organizations such as software measurement data, process models, training materials, and other documentation.

Over the last 18 years, the SEL has worked with more than 100 production projects where the software was used for mission operations and ground support of GSFC missions. In each of these, SEL analysts quantitatively assessed process changes on the developed software of real projects. These software projects ranged in size from 4 thousand lines of code to a million lines of code (Reference 3).

Software Improvement Methodology

The key distinguishing features of the SEL software improvement methodology are the following:

- Evolutionary not revolutionary
- Continuous
- Incremental
- Bottoms-up rather than top down
- Quantitative software measures
- Software experimenters work with software developers

These attributes are the key to success. Experience in many complex endeavors has shown that true process improvement takes time and commitment. The culture of an established organization must continually absorb and adapt to better ways of accomplishing its business. Experience has repeatedly shown that mandating a standard software engineering process from the top, for example, won't be accepted into an organization's culture. The people who comprise that organization must be part of the evolution of the process, the rules, and the techniques that they find work best for them in their particular environment.

The GSFC SEL software improvement strategy focuses around the simple three layer paradigm shown in Figure 2. This model recognizes that in-depth understanding must precede any attempt

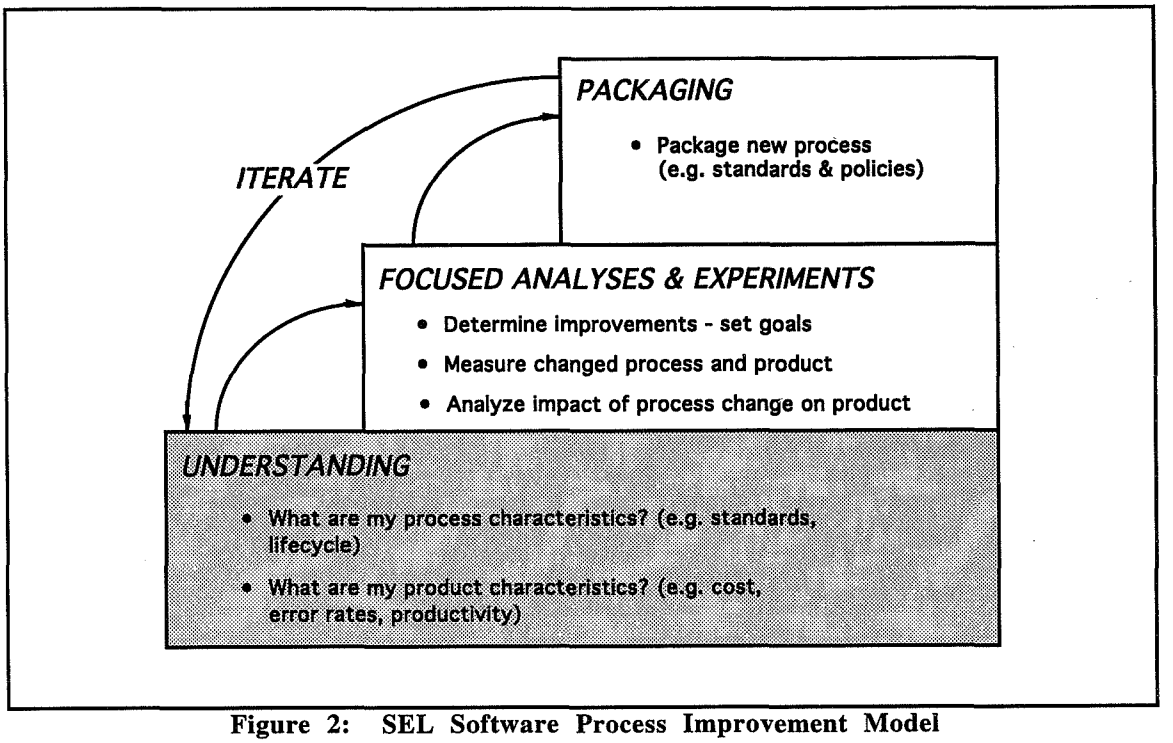


Figure 2: SEL Software Process Improvement Model

to improve. Detailed insight about the functions an organization performs, the dynamics of their interactions, the quality of their products, and the processes and tools they apply forms the basis for the second "layer" of the improvement model. This second layer benefits from the ongoing understanding activity to define focused, incremental improvement experiments. The term "experiment" is important, because change must be planned, instrumented, and compared. Many experiments may not prove helpful or at least not be beneficial in the ways or to the extent originally conceived. Further, the success of each incremental candidate improvement requires people "buy-in" which can only be gained through careful explanation and training, application, and results analysis. As stated above, true improvement takes time and is, by definition, bottoms-up. As improvements are shown to be helpful, they are packaged appropriately for ongoing use by the organization (the top layer shown in the figure). Usual examples of packaging are well-written user guidebooks and training materials. The state of the organization's business practice is thus altered. The packaged processes, tools, training, and guidance become that organization's software policies and standards. And those are effective policies and standards because they reflect what the organization really does (Reference 4).

Software Improvement Results

As an example of the application of collecting and analyzing of software error statistics at GSFC, the SEL collected data to determine the impact of the cleanroom approach on software error rates. Figure 3a shows the error rates of the baseline approach and two small (20-40 KSLOCs) development projects (Reference 5). Each time the cleanroom approach was used, the error rates decreased showing that the cleanroom approach had a positive effect on error detection rates and possibly should be adopted as part of the baselined development process for Code 552.

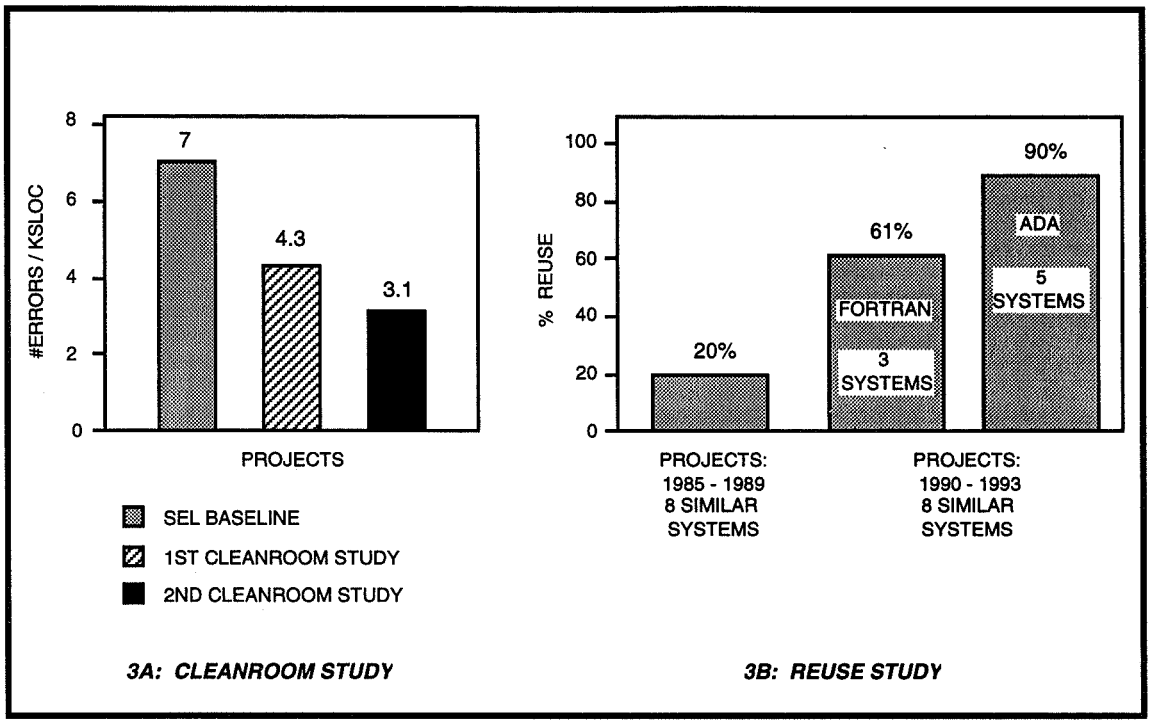


Figure 3: Cleanroom Error Rates and Object-Oriented Design Reuse Studies

Another effort focused on the use of object-oriented technology with the goal of increasing the reuse levels of software within the Flight Dynamics Division resulted in significant impacts as depicted in Figure 3b. The average level of code reuse increased from approximately 20% in the 1985-1989 time frame to over 61% reuse for FORTRAN projects and 90% reuse for Ada projects during the 1990-1994 period (Reference 3). Such improvements provide evidence of the benefits potentially derived from the application of evolving state-of-the-art software engineering practices such as object-oriented design.

Since the SEL has been monitoring and measuring the progress of GSFC mission operations and ground data software for over a eighteen years, the cumulative effect of the SEL software improvement technique on software error rates was analyzed. The software error rate is defined as the number of errors per thousand lines of code. The analysis shown in Figure 4 that the average software error detection rate decreased from 8 errors per KSLOC to 2 errors per KSLOC, a 75% decrease from 1977 to 1993. This type of information leads to well defined models and relationships of software parameters supporting improved management and control of future projects (Reference 3).

Improvement of the product within Code 552 (flight dynamics software) by changing and measuring results of software process changes on real projects has produced real gains in error rates, reuse, and productivity over the last 5 years. For a set of similar GSFC projects using this methodology since the late 1980s, the error rates decreased 75% (from 4 errors/KSLOC to 1 error/KSLOC), reuse increase from 20% reuse to 75% reuse, and the productivity increased 75% (from 440 staff-months to 110 staff-months) for an equivalent amount of software.

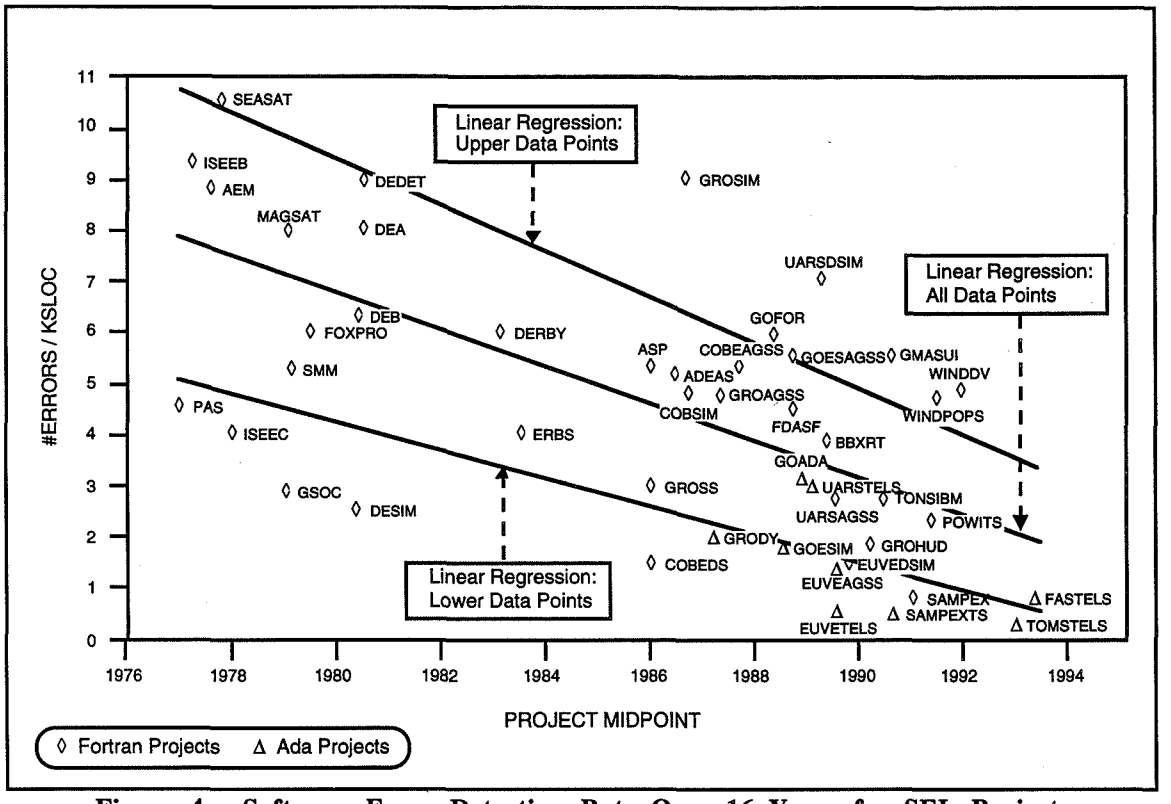


Figure 4: Software Error Detection Rate Over 16 Years for SEL Projects

Software Improvement Within GSFC Code 500

The basic methodology used by the SEL for flight dynamics software was applied to the Mission Operations and Data Systems Directorate as a whole. In accordance with the SEL software improvement model, a software "baseline" for Code 500 was established (Reference 1). The data for establishing an understanding of the Code 500 software and its development and maintenance processes was gathered via four integrated approaches: a comprehensive nine page survey, reviews of organizational and project data, informal roundtable discussions, and one-on-one interviews.

We sampled the GSFC Code 500 civil servant and support contractors by selecting respondents from organizations performing the majority of the software work. We then supplemented the insight gained through the surveys by reviewing software policies, standards, staffing data, project plans, and other organizational and project data. After initial review of the data, we conducted roundtable discussions with groups of respondents to refine the data and to obtain suggestions on how software processes might be improved. Finally, we conducted one-on-one interviews with selected managers to increase our confidence that our data was representative of the software work performed by that organization.

The published software baseline included both software product descriptions (amount of software, cost of software, software staffing, software error rates, etc.) and software process descriptions (standards used, metrics use, project volatility, development and test methodology use, training, etc.). This baseline data was then analyzed. Using the basic goals of decreasing cost, increasing productivity, and decreasing

error rates, specific areas for potential software improvement were identified. These areas were researched and a set of recommendations for improvement of the software product and processes within GSFC Code 500 were developed. Three of those recommendations are presented here.

Recommendations

Establishing our understanding baseline for GSFC 500 took time, patience, and careful analysis. We believe our understanding was sufficiently detailed and reasonably correct enough to move smartly into the second thrust of the improvement process; i.e., defining focused incremental improvements and experimenting with those improvements in controlled ways.

Three major areas that promise large near-term payoff for relatively small investments are:

1) Introduction of ongoing, continual software improvement into the culture of the Directorate

2) Establishment of an integrated software training program

3) Implementation of an effective software measurement program

These improvements can be accomplished for relatively little money and in a short time period because significant components of what are needed already exist.

Organizational Software Improvement Infrastructure

In order to implement and sustain any software improvement change across the Directorate, it is necessary to put in place a software improvement infrastructure throughout the organization. Upper management commitment and long term involvement is critical. So is the participation of everyone throughout the organization that has anything to do with software development and maintenance. People must be involved, have influence on, and help shape where their organization is going. Simply assigning another working group or holding an occasional meeting won't accomplish the software improvement goals. Improvement working teams or Software Process Groups must be established at all levels (Directorate, Division, and Branch). Both process improvement and software product improvement need to be emphasized. An initial task might be to develop a Code 500 approach (not necessarily standards) for the development and maintenance of software. With the help of improvement guidance such as the Software Measurement Guidebook (Reference 6), the Software Manager's Guidebook (Reference 7), SEL experience, and the materials from the Software Engineering Institute, this hierarchy of Software Process Groups could identify, define, and implement techniques designed to continually improve Code 500's software capabilities.

Software Training Program

GSFC Code 500 could benefit from an integrated software training program. Our findings indicate that software training tends to be focused on specific "hot" technologies as opposed to overall software process and development of personnel for key software positions. This goal could be accomplished with a bottoms-up approach by allowing project-level experiences to drive the content of the integrated training program. The needed disciplines are at minimum those of software project management, software requirements management, software contractor management, configuration management, quality assurance, and the software engineering life cycle. The courses must be consistent in approach, show the role of software measurement and feedback in the context of each discipline, and be

tightly integrated to the GSFC Code 500 approach to software development.

The curriculum will be most effective if each course has an overview version that is 3 to 4 hours in length and a full duration version (1 to perhaps 3 days depending upon the subject.) The overview version would be taken by everyone involved with software, but not directly responsible for that discipline area. For example, only software project managers and those people training to become such managers would take the full length software project management course. An effective enhancement to the basic training program would be on-line refresher modules accessible from any of the organization's networked workstations.

The basic elements of this training curriculum already exist at CSC, SAIC, and in the SEL at GSFC. This existing courseware and instructors can be tailored and enhanced and could be ready for use without a long delay or large additional investment.

Software Measurement Program

We found that little attention is given to software measurement in most GSFC organizations. Several contracts required metrics to be collected and forwarded to the government, but little or no analysis was being performed and even less in the way of improvement feedback into the actual projects. The consequence was the project and line management and staff had virtually no real insight about critical status indicators such as the number of errors in the delivered code, the amount of time any activity actually took, or how well the documentation matched the design, code, or testing.

Our recommendation is that GSFC Code 500 develop an effective, practical software metrics program to collect, analyze, and provide feedback for the following purposes:

- Continually decrease software cost
- Assist in the management of software projects
- Assure timely delivery of products
- Improve software reliability

Fortunately, practical solutions and experience are readily at hand. The SEL in GSFC Code 552 is one of the few nationally leading organizations that has proven, long term experience in the definition, analysis, and application of software metrics. The SEL-developed NASA Software Measurement Guidebook (Reference 6) will be released shortly. Code 500 should adopt a top level software measurement policy with the local organizations choosing their own specific goals for measurement, picking the minimum set of metrics needed to meet their goals, and performing metric analysis and feedback. The know-how in this Guidebook combined with an integrated training program are key improvement tools that are easily available.

Conclusions

The GSFC Mission Operations and Data Systems Directorate has successfully developed many millions of lines of code for ground systems of numerous spacecraft. Even so, our analysis of the ground and data software systems shows that are areas that could benefit from a sustainable, continuous software improvement program. The Software Engineering Laboratory is an example of this. In the past five years, the SEL saw a 75% increase in productivity and a 75% decrease in software error rates in flight dynamics projects. The SEL software improvement method of working directly with the development projects and using quantitative measures to test new software technologies can be applied throughout the GSFC Code 500 software domains.

The SEL approach of understanding, assessing, and packaging the assessment

results was applied to the Code 500 software domains in general. This study identified three areas in which GSFC Code 500 could enhance the success of their software development and maintenance projects: institute a Directorate-wide software improvement program, develop an integrated software training program, and develop a software measurement program. We believe that GSFC Code 500 has a superb opportunity to leverage the isolated experiences already existent in their organization to adopt a broad, experience-based software improvement program that could indeed be a model for both Government and industry.

References

1. McGarry, F. & Hall, D. (1994, June). *Profile of Software Within Code 500 at the Goddard Space Flight Center.* NASA/Code QE. NASA Software Engineering Report NASA-RPT-001.
2. McGarry, F., Hall, D. & Sinclair, C. (1994, June). *Profile of Software at the Goddard Space Flight Center.* NASA/Code QE. NASA Software Engineering Report NASA-RPT-002.
3. McGarry, F., Jeletic, K. (1993, December). *Process Improvement as an Investment: Measuring its Worth.* Proceedings of the Eighteenth Annual Software Engineering Workshop. SEL Report SEL-93-003.
4. Caldiera, G., et.al. (1993, December). *NASA Software Process Improvement Guidebook.* NASA/HQ/Code QE. NASA Software Engineering Report NASA-RPT-nnn. Draft.
5. Green,S.E., Pajerski, R. (1991, December). *Cleanroom Process Evolution in the SEL.* Proceedings of the Sixteenth Annual Software Engineering Workshop. SEL Report SEL-91-006.
6. Pajerski, R. & Bassman, M. (estimated 1994, July). *Software Measurement Guidebook.* NASA HQ/Code QE. NASA Software Engineering Report NASA-RPT-nnn. Draft.
7. McGarry, F., Waligora, S., Landis, L., et.al. (1990, November). *Manager's Guidebook for Software Development (Revision 1).* NASA/GSFC. Software Engineering Laboratory Report SEL-84-101.

N95-17567

111152

354239

XMM INSTRUMENT ON-BOARD SOFTWARE MAINTENANCE CONCEPT

p. 8

Mr. N. Peccia
European Space Operations Centre (ESOC),
Robert-Bosch-Str. 5, 64293 Darmstadt, Germany

Mr. F. Giannini
European Space Technology Centre (ESTEC),
Keplerlaan 1, 2200 AG, Noordwijk, The Netherlands

ABSTRACT

Whilst the pre-launch responsibility for the production, validation and maintenance of instrument on-board software traditionally lies with the experimenter, the post-launch maintenance has been the subject of ad hoc arrangements with the responsibility shared to different extent between the experimenter, ESTEC and ESOC.

This paper summarizes the overall design and development of the instruments on-board software for the XMM satellite, and describes the concept adopted for the maintenance of such software post-launch.

The paper will also outline the on-board software maintenance and validation facilities and the expected advantages to be gained by the proposed strategy.

Conclusions with respect to adequacy of this approach will be presented as well as recommendations for future instrument on-board software developments.

Keywords: **On-Board Software, System testing, Software life cycle, Software maintenance**

1. INTRODUCTION

In the last decade, the complexity of space missions has increased significantly due to the more demanding requirements on mission efficiency and quality of mission products. Such requirements could only be satisfied by designing intelligence on board for increased autonomous operation of the spacecraft and the instruments in its orbit.

On-board software and autonomy however have a significant impact in the design of the ground facilities for the support of the mission. Although instrument on-board software is designed, developed and tested following strict quality assurance procedures, experience of past and current missions show that the capability of reprogramming instrument on-board software from the ground is an essential requirement throughout the instrument lifetime.

Certain events during the instrument lifetime can create the necessity to modify the flight software. The causes of change in the on-board software are manifold :

- Change of specification (e.g. errors, essentially numerical, in the specification of thresholds, calibration, delays, etc.)
- Non conformance of the software with the original specifications (e.g hidden

PRECEDING PAGE BLANK NOT FILMED

PAGE 984 INTENTIONALLY BLANK

bugs not detected during testing on ground)

- On board hardware failure following which the instrument can only be recovered by reprogramming the on-board software. This event is the most likely reason for the necessity of maintenance activity because its unexpected nature renders it impractical to implement complete autonomy in the software with respect to such failures.

- Change in strategy in instrument operation (e.g. changes to improve capability or efficiency)

The complexity of instrument on-board software maintenance is directly related to the on-board software configuration.

Different approaches have been taken to instrument on-board software maintenance from mission to mission. The main variation has been the responsibility for actually making software corrections and implementing new on-board software requirements, which has in some cases been done by the experimenters and in other by ESTEC and/or ESOC.

The factors influencing the choice of a particular maintenance scheme are :

- availability of expertise

- availability of :

  a Software Development Environment (SDE) which contains CASE tools supporting on-board software lifecycle for development and change of the software, verification and validation, configuration management and documentation. The facilities also include cross-compilers, cross-debuggers and downloaders to compile, load and debug the instrument software from the host to the hardware target.

  a Software Validation Facility (SVF) for on ground testing and validation. The SVF provides facilities to emulate or simulate the hardware environment of the instrument on-board software. Facilities range from software simulators and emulators to replicas of instrument on-board hardware systems.

- duration of mission

2. XMM OVERVIEW

The X-Ray Multi-Mirror Mission (XMM) is a high throughput X-ray spectroscopy mission (photon energy range from 0.1 Kev to 10 Kev), which is the second cornerstone of the ESA long term scientific plan. The XMM is a facility type observatory open to the worldwide astronomical community. The scientific payload forms an integrated mutually complementary package optimised to fulfil the scientific aims of the mission and fully exploit the ESA supplied X-ray optics.

The XMM observatory will offer a major step forward in the field of X-ray astrophysics in the 21st Century. It is envisaged as a long duration facility class mission aimed at performing detailed imaging spectrophotometry of a wide variety of X-ray sources. The observatory will be placed in a 24 hour highly eccentric inclined orbit to allow uninterrupted observations up to 16 hours using the groundstation of Perth (Australia). The spacecraft consists of a service module which carries the payload module.

The scientific instruments are:

- European Photon Imaging Camera (EPIC)

  The XMM x-ray telescope consists of three separate co-aligned mirror modules, for each of which EPIC will provide an imaging x-ray focal camera. Each of these cameras will be mounted at the focus of the respective mirror modules. Two different types of Charge Coupled Devices (CCDs) will be used: one type based on p-n and the other on MOS technology.

- Reflection Grating Spectrometer (RGS)

  RGS features two independent instrument chains placed behind two of the three mirror modules. Each chain incorporates an array of reflection gratings which pick off roughly half of the X-ray light and deflects it to a strip of CCD detectors offset from the telescope focal plane. The remaining light passes undeflected through the grating stack where it can be utilised by other instruments located in the focal plane.

- Optical Monitor (OM)

  OM is a UV / optical telescope with two chromatically split channels, the blue channel and the red one. Both beams are transmitted to the CCDs detectors.

The XMM spacecraft will be operated in a continuous interactive mode from a Mission Operations Control Centre (MOC). XMM Science operations will be conducted from a Science Operations Centre (SOC) in close interaction with the MOC.

Considering

- the long duration of the mission (10 years)
- each experimenter has his own SDE/SVF
- distributed processor architectures are present in the payload
- different languages are used on the processors

the following sections describe how a different set of instrument on-board software maintenance and validation facilities could be assembled, which would allow the SOC, given the appropriate expertise, to assume the responsibility for instrument on-board software maintenance in the majority of the cases.

Additionally the following items will be addressed

- Design choices
- Inclusion of hardware-in-the-loop
- "worst case" testability
- exception handling in the software

The current baseline is that the Instruments Software will be maintained in the SOC with the Instrument Software Subsystem (ISS); however during the early phases of the XMM mission support from the instrument development teams will be available (e.g. for validation).

3. XMM INSTRUMENT SOFTWARE MODULES

This section describes briefly the various on-board software modules in the Instruments,

mostly on different hardware units (instruments contain more than one processor with a maintainable software module) :

In the **EPIC** experiment SW is present in :

a) EPIC Mos Data Handling Unit
b) EPIC Control and Recognition Unit
c) EPIC Pn Data Handling Unit
d) EPIC Pn Event Analyzer Unit
e) EPIC Pn Analogue Electronic Unit

In the **RGS** experiment SW is present in the RGS Digital Electronic Unit running on the following processors:

f) Instrument Controller Processor
g) Data Pre-Processor

In the **OM** Software is present in:

h) Instrument Controller Unit
i) Data Processing Unit

In the following the Software Modules will be indicated by the above letters. These modules are of different size and complexity, and they can be classified in 2 categories:

- Running on what is traditionally identified as the Instrument Controller (a,c,f,h)

- Running on secondary processor (b,d,e,g,i)

Regardless of the names used for the Units, we will call IC the units interfacing with the Spacecraft On Board Data Handling System (OBDH).

The Software Modules run on different type of processor:

- MIL-STD-1750A (a,c,f,g,h)
- HARRIS 80C86 (b,d,e)
- MOTOROLA 56001 (i)

and different language are used:

- Ada (a,c,f,h)
- C (b,d,e,i)
- Assembler 1750A (g)

Assembler is also used on other software modules which are on PROM and are not modifiable (e.g. boot/loader code for f and h).

4. XMM INSTRUMENT DEVELOPMENT ENVIRONMENTS (SDE)

The XMM instruments are being developed by different experimenters across Europe (United Kingdom, France, Italy, Belgien, Germany and Netherlands) with some collaboration from USA (RGS and OM instruments). The consequence is the use of different host machines, target processors, languages and tools among the instruments.

Figure 1 summarizes the Software Development Environments which are being used to develop the various instrument on board software modules.

5. XMM INSTRUMENT ON-BOARD SOFTWARE MAINTENANCE APPROACH

The following assumption are made:

- Instrument simulators will be available and they will be based on Instrument Controller (ICU) processor emulator running the on-board SW.

SW Module	Host Machine	Target Processor	Language	Tools
a , c	SUN	MA31750	Ada	TLD APSE, Debugger, Simulator, TEK V1750 Emulator
b, d, e	HP9000	80C86	C	HP AXLS C/80C86 Cross-compiler, Debugger, Emulator, Simulator
f, h, g	SUN Solaris 1.1	MA31750	Ada Assembler	Tartan Compiler / assembler, AdaScope debugger
i	SUN	56001	C	GCC.G56KCC OM Simulator

Figure 1; Instrument Software Development Environments

- All software modules need to be maintained
- Modification of the instrument on-board software cannot damage the instruments while the instrument is monitored from the ground.
- The Software delivered with the instrument flight model (FM) has been fully validated.

The purpose is to outline a coherent approach in the frame of a plan for the maintenance of the software on the various on-board processors and during the various relevant phases (development, commissioning and routine operations).

The trade-off between instrument on-board software maintenance at the XMM SOC on the one hand and maintenance via each experimenter on the other hand has been based on the following assessment criteria :

- Criticality of the on-board software, which covers an assessment of the impact an erroneous software modification might have on the performance of the instrument
- Software complexity versus availability of expertise, which addresses the degree of expertise needed for a specific software maintenance during the commissioning and the routine operations phases.
- Cost aspects which addresses
  - investment costs for hardware, software and documentation including installation at the SOC and training of personnel
  - operations costs at the SOC

6. DEVELOPMENT ENVIRONMENT

The set of activities involved in the maintenance of the XMM instrument flight software will be executed at the XMM Scientific Operations Centre (SOC).

In order to perform these activities the SOC will require a common SDE which will ease the maintenance activities and will limit the costs. The development environment for the Instrument Software will be composed of the total set of Software tools used by the developers of the Software modules.

All tools mentioned in the Figure 1 on section 4 will be available for modification of the instruments Software to ensure compatibility with the implemented instrument flight software.

Other tools will be used in the development of the Software (e.g. AdaNice HOOD tool for the Architectural design of the EPIC Data Handling Software), but the use of such tools is not considered necessary for the maintenance, due to the limited structural changes in the code during the maintenance phase.

The development environment will be hosted on the smallest set of computer needed to host all tools in a version equivalent to what used by the developers. At the moment a SUN SPARC and an HP9000 are needed to host all tools. In order to ensure that the compiled code produced by the SOC SDE is compatible with that flown during the mission it will be necessary to freeze the compilers version at the version delivered with the instrument Flight Model.

A configuration management tool should be added in order to keep track of changes. No configuration information prior to delivery will be used. Configuration management will be restarted with the Software as delivered for the launch.

Full documentation of the Software development will be available on paper as delivered by the developers. Electronic form of the documents might also be available, but no standard format has been mandated.

The following will be available:

- Source code of all instrument Software
- All "makefile" and any image generation procedure used by the developers

7. VALIDATION ENVIRONMENT

The validation environment will be different for the various type of processors used and the functionality of the Software module. The main driver of the proposed approach is the high investment and maintenance costs associated with a SVF based on an Engineering or spare Flight model.

7.1 INSTRUMENT CONTROLLER

For ICU software (modules a,c,f,h), the capability of the instrument simulators to run the Software will be exploited. This solution does not have the fidelity of the actual hardware, and therefore its adoption is associated with an element of risk. The level of risk is related to the degree to which the instrument on-board software is sensitive to the flight hardware performance (timing, I/O performance).

Other solution would be the "hardware-in-the-loop" design, based on commercially available VME cards and hosting the target processors. This approach was discarded due to higher costs because additional secondary processor hardware is needed.

The use of the instrument simulator as a validation tool has the advantage that it implicitly contains a realistic environment simulation and the means to easily vary this environment. The preparation for and conduct of validation tests is easier than for an EM (Engineering Model) or a spare FM (Flight Model) based system.

After the unit testing and software integration, the new executable image of the module will be first executed on the 1750 processor emulator for simple tests.

It will be then loaded on the instrument simulator, exercised by TC and stimulated by data files reproducing the Instrument data

flow. The data files used in these tests are available from unit tests or calibration tests.

This will allow to "partially" validate the Software before up-linking into the instrument. The settings of the instrument simulator will allow to simulate Hardware failure in the instrument. This strategy for test and integration causes some difficulties on real time embedded software, as follows :

- Emulators have limited facilities for exercising in real time and simultaneously monitor the embedded software
- It is often impossible to reproduce a test 100%
- It is very difficult to create a "worst case" test
- It is difficult to exercise exception handling in the software

The proposed approach is also based on the criticality of the Instrument Controller on-board software. Error in new images or software patches causes no damage to other instruments. A power on reset will bring the software back to the PROM reference. Full ground validation is not required because of criticality. The proposed partial validation is necessary due to the complexity of some software modules.

7.2 SECONDARY PROCESSORS

Modification to all other software modules will only be tested on the host computers and on the processor emulators (b,d,e,g,i) with data files generated by simulation software or collected during instrument testing. The impact of software modifications on the secondary processors is considered negligible. Additionally the XMM instrument simulator can not be used for validation because it does not emulate the secondary processors.

Anyhow, the relative simplicity of these Software modules makes them easier to test except for timing behaviour. Furthermore, the modification to these modules are more likely to be needed during the early phases of the operations, because of the possible difference of the actual data from the expected ones.

During the early phases of the operation (6 months), the instrument developer teams will modify the software using the Instrument Software Subsystem at the SOC; validation of the Software modules will be complemented by the possible use of the test equipment available at the experimenter premises.

After this period, if the experimenter facilities are not available, the Software will only be tested on the host computers; the last tests before declaring the Software operational, will be executed on the flying instrument using when possible the period of the orbit below 40,000 Km.

It is, however necessary, that it be demonstrated that changes to the instrument software do not adversely affect the performance of the system as a whole, either functionally or in the consumption of resources. This can be done by analysis (in the absence of a validation facility), or by demonstration.

The responsibility for demonstrating that any changes to instrument software will not adversely affect the system will lie with the XMM SOC during the routine operations phase.

8. CONCLUSIONS

In the XMM SOC project the instrument software maintenance problem is tackled by setting up a centralised software maintenance facility, the ISS (Instrument Software subsystem), which will take over the on-board software maintenance of all instruments when the commissioning phase is over.

In order to fulfil the requirements and responsibilities for such facility the XMM SOC requires :

- a common Software Development Environment (SDE) compatible with that used by the experimenters to produce the flight software and maintainable for the mission duration. This common SDE will ease the maintenance task and will limit the costs.
- delivery of the software module source code, the standards applied during development and testing, the test harnesses, test procedures and test results.
- a Software Validation Facility ,also maintainable throughout the mission, which consist of :
  - the instrument simulator running the instrument controller emulator
  - simple emulators of the secondary processors for testing and validation on the host computer

The final decision whether or not to implement an instrument software change resides with the XMM Project Scientist.

The proposed solution is based on the analyses of the criticality of the XMM instrument on-board Software as regards instrument performance, on the availability of expertise at the SOC during the various phases of the mission as well as on a cost estimation.

9. RECOMMENDATIONS

For missions with long lifetime, as XMM, ESA should take over post launch instrument on-board software maintenance. This will be more cost effective, since it involves only a marginal expansion of existing teams. It will also result in a better and more responsive service, will simplify the operational interfaces and will help continuity of expertise in the SOCs.

Standardisation of Software Development Environments should be managed / mandated early in the project in order to reduce the cost of the maintenance environment without penalising the instrument developers.

10. REFERENCE DOCUMENTS

1. Science Operations Centre Implementation Requirement Document (SIRD), PX-RS-0392
2. SOC Implementation Plan (SIP), XMM-SOC-PL-0100
3. Software Project Management Plan for EPIC Experiment, TL 10002 EPIC-LAB-PL-001 Issue 1, April 1994
4. XMM-OM Instrument Control Unit Software Development Environment, XMM-OM/MSSL/SP/0025.01 25/5/94
5. XMM-OM Instrument Control Unit Software Verification & Validation Plan, XMM-OM/MSSL/SP/0026.01 25/5/94
6. Digital Processing Unit Electronic Ground Support Equipment and Software Development Environment XMM-OM/PENN/SP/0005.1 24/5/94
7. DPU Software Validation Plan, XMM-OM/PEN/SP/0006.draft 3/5/94
8. Huygens On-Board Software Maintenance, DOPS-SMD-HUY-OBS-001, Issue 1, 17.02.94

N95-17568

111153

354241

INTEGRATION OF A SATELLITE GROUND SUPPORT SYSTEM BASED ON ANALYSIS OF THE SATELLITE GROUND SUPPORT DOMAIN

P- 8

R. D. Pendley, E. J. Scheidker, D. S. Levitt, C. R. Myers, and R. D. Werking

Computer Sciences Corporation
10110 Aerospace Road
Lanham-Seabrook, Maryland 20706

This analysis defines a complete set of ground support functions based on those practiced in real space flight operations during the on-orbit phase of a mission. These functions are mapped against ground support functions currently in use by NASA and DoD. Software components to provide these functions can be hosted on RISC-based work stations and integrated to provide a modular, integrated ground support system. Such modular systems can be configured to provide as much ground support functionality as desired. This approach to ground systems has been widely proposed and prototyped both by government institutions and commercial vendors. The combined set of ground support functions we describe can be used as a standard to evaluate candidate ground systems. This approach has also been used to develop a prototype of a modular, loosely-integrated ground support system, which is discussed briefly. A crucial benefit to a potential user is that all the components are flight-qualified, thus giving high confidence in their accuracy and reliability.

Introduction

The satellite ground support domain comprises all ground-based (as opposed to onboard) activities needed to operate an orbiting spacecraft, including the bus and payload. It does not include such activities as, for example, instrument data reduction from a scientific satellite, image production from a weather satellite, or message traffic management from a communications satellite; although the ground support domain does cover capturing and making available the data required by such end-user processes. This domain also includes the integration of payload plans and commands into the overall plan for mission support. The activities supported by functions in this domain also differ during the prelaunch, launch, early mission, on-orbit, and end-of-life phases of a mission. In this paper we undertake to define a complete set of spacecraft support functions that span the satellite ground support domain during on-orbit operations for one or more spacecraft.

The principal motivation for this analysis is the belief that satellite ground control systems, traditionally implemented on central processor systems based on mainframe or mini-computers, can be hosted on client-server or other architectures, based on high-performance work-stations linked in networks. Such systems have been proposed within government organizations such as NASA and the Defense Department, and by numerous commercial firms.

By looking at the functions covered by two of these proposed architectures and applying our own spaceflight support experience, we have derived a superset of functions that covers all the aspects of satellite flight support. This set of functions facilitates comparison among the numerous approaches to distributed, open-system architectures that have been proposed in the past four years. We also discuss a loosely integrated ground support system prototyped at CSC in an effort to understand how to move to a distributed, open-system architecture while taking maximum advantage of the enormous amount of existing flight-proven software developed for mainframe- and mini-computer-based ground systems.

Spaceflight Ground Support Functions

The ground support functions found in the two sources investigated for this paper are summarized in Table 1. The first column lists the functions summarized by A. R. Stottlemyer and his co-authors in a paper proposing distributed architectures for NASA ground systems

Table 1 - Two sets of satellite ground support functions

Stottlemyer *et al.*	DoD ISC HCI
1 Mission Design	1 Schedule Resources
1.1 Orbit requirements and design	2 Create Satellite Support Plan
1.2 Attitude requirements and design	3 Update Satellite Support Plan
2 Remove communications artifacts	4 Configure, Test, and Verify System
3 Spacecraft position and orientation	4.1 Verify Configuration
3.1 Orbit determination	4.2 Test End-to-end Configuration
3.2 Attitude determination	4.3 Configure for Operations
4 Analysis of spacecraft operations performance	5 Perform Satellite Support
4.1 Trend analysis	5.1 Acquisition of Signal
4.2 Command Response	5.2 Verify Tracking
5 Analysis of scientific instrument performance	5.3 Verify Correct Telemetry Stream
5.1 Data quality	5.4 Verify Frame Synchronization
5.2 Measurement quality	5.5 Verify Command Link
5.3 Calibration	5.6 Perform Planned Commanding
6 Operations planning	5.7 Verify Satellite State of Health
6.1 Spacecraft operations	5.8 Produce Output Products
6.2 Instrument operations	5.9 Complete and Verify Support Activities
6.3 Support environment operations	5.10 Log Activities
6.4 Supporting analysis	5.11 Terminate Pass
7 Spacecraft command and control	6 Deconfigure Resources
7.1 Command generation	6.1 Deconfigure Resources
7.2 Command validation	6.2 Verify Deconfiguration
7.3 Command issue	7 Orbit Data Collection and Verification
8 Scientific data analysis	7.1 Collect Orbit (Tracking) Data
8.1 Data preparation and management	7.2 Verify Data
8.2 Analysis algorithm management	8 Attitude Data Collection and Verification
8.3 Support for data access and manipulation	8.1 Collect Attitude Data
8.4 Product generation and distribution	8.2 Verify Data
9 Data acquisition and management	9 State of Health Data Collection
10 System resource management	9.1 Request State of Health Data
10.1 Physical resources	9.2 Collect State of Health Data
10.2 Operations staff	9.3 Process and Verify Data
11 Integration and test	10 Orbit Determination and Planning
	10.1 Predict Orbit
	10.2 Plan Orbit Maneuvers
	10.3 Maintain Orbit Model
	11 Attitude Determination and Planning
	11.1 Plan Attitude Determination
	11.2 Plan Attitude Maneuvers
	11.3 Maintain Attitude Model
	12 State of Health Determination and Planning
	12.1 Determine State of Health
	12.2 Plan State of Health Activities
	12.3 Maintain State of Health Model

(Stottlemyer *et al.*, 1993).The functions in the second column are taken from a Defense Department standard drafted by the Integrated Satellite Control (ISC) Human Computer Interface (HCI) Working Group (ISC HCI Working Group, 1993). NASA Goddard's Mission Operations

Directorate has also begun an extensive campaign to take advantage of workstation-based, distributed architectures for satellite ground support. However, this effort, called the Renaissance Initiative, is newly begun and it is therefore premature to include it in this analysis.

These two sets of ground support functions represent different views of satellite ground support. The Stottlemyer *et al.* paper was written primarily to explore the feasibility of system architectures and is not meant to be an exhaustive analysis of the ground support domain. Their paper nonetheless contains a list of eleven high-level ground support functions that we have broken into subcategories to facilitate comparison with other function sets. This architecture analysis was one of the drivers of the Renaissance initiative and in this analysis we use it as a snapshot of the NASA ground support function set.

The Defense Department function set is taken from an appendix of a standard drafted to define DoD's view of the optimum interface between humans and computers for satellite ground support. In writing this standard, these authors also found that they needed a generic set of satellite ground support functions, which appears in this appendix and which we have taken to represent a picture of DoD satellite ground support.

In defining our superset of ground support functions, we made the following assumptions:

- only on-orbit operations considered in this analysis
- payload (instruments, e.g.) operations and planning not included
- integration of payload commands and schedules received through external interface included
- no particular institutional organization assumed, but system resources can be physically separated

We created the superset of functions appearing in Table 2 on the next three pages by combining the two function sets in Table 1 and adding elements drawn from our own ground support experience. We have tried to generalize functions. For example, NASA places considerable importance on managing onboard flight recorders to maximize scientific data return. A more general function might be the optimum management of onboard resources, for which different operations teams might have varying goals such as maximum observation time or extended mission life. One purpose of our function 1.3.7, *integrate commands to form command load*, is to optimize the planned command load within such constraints.

To organize the listed functions, we set up the seven main categories and sixteen subcategories shown in the light grey areas of Table 2. These areas are collectors of identifiable functions, which are in turn mapped against the other function sets. To facilitate comparison with reference functions, we have mapped them into our categories, using broad interpretations. Note that Stottlemyer functions 1.1 and 1.2 are not included, because they are requirements definition, hence prelaunch and not part of the on-orbit phase. This arrangement can be modified by adding or deleting lower level functions. As we extend this analysis to other mission phases, such as launch or end-of life, it is reasonable to anticipate that the function set will need modification.

The major categories were chosen by analyzing the reference function sets and other models, seeking high-level function collectors that would span the entire domain of on-orbit flight operations and would be significant for all identifiable missions. These categories are discussed below.

Defining the *spacecraft state* (1) in terms of a physical model and its state representation is the basis of the spacecraft mission control systems developed by the Altair Aerospace Corporation (Wheal, 1993). We have called this part of the spacecraft state the *vehicle state* (1.1), defined by the collection of its telemetry values. To fully define the concept of the spacecraft state, we have added the concept of the *dynamic state* (1.2), reflecting the fundamental flight dynamics definition of state as a set of parameters defining

Table 2 - Superset of ground support functions mapped against previous sets

SPACECRAFT GROUND SUPPORT FUNCTIONS	Stottlemyer *et al.*	DoD ISG HCI
1 SPACECRAFT STATE		
1.1 Vehicle State		
1.1.1 Monitor Health and Safety	6, 6.1	5.8, 5.9
1.1.2 Monitor Payload Status	5, 5.1	5.8, 5.9
1.1.3 Verify Commands	4, 4.2	5.9
1.1.4 Monitor OBC Image		5.9
1.2 Dynamic State		
1.2.1 Real-time Orbit Determination	3, 3.1	5.9
1.2.2 Off-line Orbit Determination	3, 3.1	10.1
1.2.3 Verify Orbit Maneuvers	4, 4.2	5.9
1.2.4 Real-time Attitude Determination	3, 3.2	5.9
1.2.5 Off-line Attitude Determination	3, 3.2	11.1
1.2.6 Verify Attitude Maneuvers	4, 4.2	5.9
1.3 State Transitions		
1.3.1 Generate and Validate Commands to Alter Vehicle State	7, 7.1, 7.2	2, 3, 12.1
1.3.2 Generate and Validate Commands for Orbit Maneuver	7, 7.1, 7.2	2, 3, 10.2
1.3.3 Generate and Validate Commands for Attitude Maneuver	7, 7.1, 7.2	2, 3, 11.2
1.3.4 Generate and Validate Flight Software Change	7, 7.1, 7.2	2, 3, 12.1
1.3.5 Generate and Validate OBC Image Commands	7, 7.1, 7.2	2, 3, 12.1
1.3.6 Generate and Validate Payload Commands	7, 7.1, 7.2	2, 3, 12.1
1.3.7 Integrate Commands to Form Command Load	7	2, 3
1.3.8 Uplink Command Load	7, 7.3	5.6
1.3.9 Uplink Real-time Command	7, 7.3	5.6
2 MISSION AND SPACECRAFT OPERATIONS		
2.1 Planning and Scheduling		
2.1.1 Predict Orbit	6, 6.4	2, 3, 10.1
2.1.2 Predict Orbit Event	6, 6.4	2, 3
2.1.3 Plan Orbit Maneuver	6, 6.4	2, 3, 10.2
2.1.4 Predict Attitude	6, 6.4	2, 3, 11.1
2.1.5 Predict Attitude Event	6, 6.4	2, 3, 11.1
2.1.6 Plan Attitude Maneuver	6, 6.4	2, 3, 11.2
2.1.7 Plan Spacecraft Contact	6, 6.4	2, 3
2.1.8 Plan Vehicle Activity	6, 6.1	2, 3
2.1.9 Plan Payload Activity	6, 6.2	2, 3, 12.1
2.1.10 Plan Flight Software Change	6, 6.1	2, 3
2.1.11 Plan System Configuration	6, 6.3	1, 2, 3
2.1.12 Plan Remote Resource Activity	6, 6.3	1, 2, 3
2.1.13 Integrate and Optimize Plan and Schedule	6	2, 3
2.2 Logging and Reporting		
2.2.1 Log Events	8, 9	5.10
2.2.2 Access Logs	9	
2.2.3 Create Spacecraft State Report		
2.2.4 Generate Mission Operations Report		
2.2.5 Generate Communications Report		
2.2.6 Generate Data Management Report	8, 8.3, 9	

SPACECRAFT GROUND SUPPORT FUNCTIONS	Stottlemyer *et al.*	DoD ISG HCI
2.2.7 Generate System Operations Report		
2.2.8 Generate Calibration Report		
2.2.9 Generate Simulation Report		
2.2.10 Generate Integrated Reports and Digests		
3 SPACECRAFT COMMUNICATIONS		
3.1 Ground RF Support		
3.1.1 Predict Attitude-independent Contact Times	6, 6.4	2, 3
3.1.2 Predict Attitude-dependent Contact Times	6, 6.4	2, 3
3.1.3 Compute Acquisition Data	6, 6.4	2, 3
3.1.4 Model Antenna Position and Mask	6, 6.4	10.3
3.1.5 Acquire Signal		5.1
3.1.6 Terminate Contact		5.11
3.1.7 Determine Onboard Oscillator Frequency	2	
3.1.8 Perform Frequency Compensation	2	
3.1.9 Perform Range Correction	3.1	
3.2 Tracking Data Streams		
3.2.1 Capture Tracking Data	2, 9	5.2, 7.1
3.2.2 Verify Tracking Data	9	5.2, 7.2
3.2.3 Sort and Sequence Tracking Data	9	7.1
3.2.4 Check Tracking Data Quality	3.1, 9	7.2
3.3 Telemetry Data Streams		
3.3.1 Capture Real-time Telemetry Data	2, 9	5.3, 8.1, 9.2
3.3.2 Verify Real-time Telemetry Data	9	5.3, 8.2, 9.3
3.3.3 Sort and Sequence Real-time Telemetry Data	9	5.4, 8.1
3.3.4 Check Real-time Telemetry Data Quality	9	8.2
3.3.5 Capture Playback Telemetry Data	2, 8, 9	8.1, 9.2
3.3.6 Verify Playback Telemetry Data	8, 9	8.2, 9.3
3.3.7 Sort and Sequence Playback Telemetry Data	8, 9	8.1
3.3.8 Check Playback Telemetry Data Quality	3.2, 5, 5.1, 8	8.2
3.4 Command Streams		
3.4.1 Verify Command Link	2	5.5
3.4.2 Command Echo		
4 DATA MANAGEMENT		
4.1 Archive		
4.1.1 Archive Telemetry Data	8, 8.1, 9	5.3
4.1.2 Archive Tracking Data	9	5.2
4.1.3 Archive Command Data	9	
4.1.4 Archive OBC Image Data	9	
4.1.5 Archive Processed Data		
4.1.6 Archive System Configuration Data	9	
4.1.7 Archive Logs	9	
4.1.8 Archive Reports	9	
4.2 Retrieval		
4.2.1 Retrieve Telemetry Data	8, 8.1, 9	8.1, 9.1, 9.2
4.2.2 Retrieve Tracking Data	9	7.1
4.2.3 Retrieve Command Data	9	
4.2.4 Retrieve OBC Image Data	9	9.1, 9.2

SPACECRAFT GROUND SUPPORT FUNCTIONS	Stottlemyer *et al.*	DoD ISG HCI
4.2.5 Retrieve Processed Data		
4.2.6 Retrieve System Configuration Data	9	
4.2.7 Retrieve Logs		
4.2.8 Archive Reports		
4.3 Data Analysis		
4.3.1 Data Trend Analysis	4, 4.1	12.1
4.3.2 Engineering Analysis	4, 4.1	12.1
4.4 Reference Databases		
4.4.1 Telemetry Database		12.3
4.4.2 Command Database		12.3
4.4.3 Spacecraft Ephemeris		10.3
4.4.4 Solar, Lunar, and Planetary Database		10.3, 11.3
4.4.5 Geophysical Database		10.3
4.4.6 Time Reference Database		
4.4.7 Star Catalog		11.3
4.4.8 Spacecraft Properties Database		10.3, 11.3,
4.4.9 Rules Database		
5 SYSTEM OPERATIONS **5.1 Configuration**		
5.1.1 Configure Local Resources	10, 10.1, 11	4.1, 4.2, 4.3
5.1.2 Configure Remote Resources	10, 10.1, 11	4.1, 4.2, 4.3
5.2 De-Configuration		
5.2.1 De-configure Local Resources	10, 10.1, 11	6.1, 6.2
5.2.2 De-configure Remote Resources	10, 10.1, 11	6.1, 6.2
5.3 Remote Resource Transactions		
5.3.1 Send Message to Remote Resource	8, 8.3, 10.1	4.3, 6.1
5.3.2 Receive Message from Remote Resource	8, 8.3, 10.1	4.1, 6.2
5.3.3 Send Data to Remote Resource	8, 8.4	4.2, 6.1
5.3.4 Receive Data from Remote Resource	8, 8.4	4.2, 6.3
6 CALIBRATION		
6.1 Calibrate Spacecraft State		12.3
6.2 Calibrate Telemetry Conversions	5, 5.3	12.3
6.3 Correct Spacecraft Properties and Model	4.2	10.3, 11.3
6.4 Correct for Biases and Misalignment	3.2, 5, 5.3	11.3, 12.3
6.5 Calibrate Propulsion System	4.2	10.3
6.6 Calibrate Tracking Data	3.1	
7 SIMULATION		
7.1 Simulate Telemetry	11	4.2
7.2 Simulate Tracking	11	4.2
7.3 Simulate Commands	11	4.2
7.4 Simulate OBC	11	4.2
7.5 Simulate Vehicle State	11	4.2
7.6 Simulate Dynamics State	11	4.2
7.7 Simulate System Resources	11	4.2

the spacecraft position, velocity, attitude, attitude rates, and additional parameters needed to determine its dynamics. Carrying this concept to its logical conclusion, the process of commanding becomes one of making *transitions* (1.3) between states. Note that the command generation defined in this category refers to generating commands for uplink, distinguished from the command planning that appears in the next category. We made this distinction because of the potential applicability of rules-based systems to generating and integrating safe, optimized command loads.

The concept of *mission and spacecraft operations* (2) appears in all the function sets. We have divided this area into two parts. *Planning and scheduling* (2.1) appear in both of the reference function sets. The *logging and reporting* (2.2) category is less well represented in the references. Here logging refers to making records of actions taken, plans executed, and events that have occurred. Reports are passed among flight team members and to outside parties, and are taken from logs, data, and analysis of data. In all the superset categories the low-level functions are stated as singular, but can be combined to make complex functions for multiple spacecraft. For example, planning an orbit maneuver might require optimizing fuel consumption, the target orbit, and tracking and communication opportunities, requiring iteration and integration of the individual functions.

Spacecraft communications (3) is taken from analysis of Goddard mission operations. *Ground RF support* (3.1) covers the functions needed to establish radio-frequency links between the spacecraft and ground controllers, including antenna modeling and signal management. Two types of data may be received: *tracking* (3.2), bearing position and velocity information, and *telemetry* (3.3), reflecting the vehicle state. Data flows to the spacecraft as *commands* (3.4), effecting state transitions.

Large volumes of data, particularly received from the spacecraft and resulting from processing, are characteristic of the ground support domain, making *data management* (4) essential. As in most application domains, this category includes *archive* (4.1), *retrieval* (4.2), and *analysis* (4.3) of data. We have additionally added *reference databases* (4.4) such as star catalogs, telemetry conversions, or rules for applied intelligence processing.

As found in both reference function sets, *system operations* (5) require functions of their own. NASA and DoD functions differ sharply in this area. For DoD spacecraft, a ground support system deals with multiple spacecraft, while for a NASA satellite there is generally a dedicated ground system. Using one system for several spacecraft makes *configuration* (5.1) and *de-configuration* (5.2) significant problems. A NASA flight operations team generally relies on ground resources physically remote from its control center, unlike DoD facilities that place all the resources in one place. Dealing with distant antennas or networks requires additional communication and data channels for *transactions with remote resources* (5.3).

We have added the category *calibration* (6) to reflect the need to tune the performance of the spacecraft and ground support system based on data from past performance. Calibration results appear in the reference databases of category 4.4.

There is some question whether *simulation* (7) is a part of flight operations, or a test-and-integration function only. We include it on the grounds that changes onboard the spacecraft, evolution of the mission objectives, and pursuit of operational efficiencies will make modification of the system and its configuration necessary, requiring testing throughout the mission. Also, some mission teams utilize simulated data for training, maneuver prediction, and operational activity modeling.

Integrated Ground Support System Prototype

In 1992, CSC began work on a prototype ground system proposed by R. D. Werking (Werking and Kulp, 1993), called the CSC Integrated Ground Support System (CIGSS). The goal was to demonstrate that the functionality needed for

ground support could be placed on a RISC-based workstation under UNIX by taking maximum advantage of the large amount of existing ground support software. Components were to be re-hosted as necessary from other platforms and operating systems, and loosely integrated by creating file interfaces between pairs of programs. Components were to be drawn from the NASA Goddard software legacy, obtained from vendors, or developed if necessary.

A working prototype has been developed and demonstrated, showing the feasibility of this approach and giving some insights into the software and system engineering needed to exploit the large amount of existing ground software on workstations. For example, B. S. Groveman and his co-workers have rehosted FORTRAN programs from IBM mainframe computers, finding the transition of computational modules straight-forward, but the creation of user interfaces more challenging. (Groveman *et al.*, 1994).

The functions originally proposed for this system were command and control, health and safety monitoring, flight dynamics, mission planning and scheduling, and payload data management functions. However, in looking at how to combine candidate components, we soon found it necessary to have a function set that enabled us to describe what a particular set of components could do in combination. This experience led us to create the superset of ground support functions.

Conclusions

We expect that future ground systems will be integrated from existing components, certainly with some modification and tailoring, but rarely developed through the traditional lifecycle. Long-time spacefaring agencies such as NASA and DoD possess enormous legacies of expensively acquired, flight-tested software, and an ever-growing number of commercial vendors are offering products for spacecraft ground support. The result is a range of choices for nearly all the functions needed for a ground support system, albeit in complicated combinations needing some form of evaluation and validation.

We have, therefore, developed a generic set of ground support functions to guide evaluation of the functionality of components and to assist in choosing an appropriate set to integrate. With these goals in mind, we intend to extend this exercise in four ways. First, the ground support domain is large and complex, and its boundaries are not sharp, so we expect to adjust our functions as we continue its analysis. Second, we intend to cover other mission phases. Third, we intend to evaluate different operations concepts and user interfaces as a way to minimize operations costs. Finally, the function set would make a far better evaluation tool if it has quantitative performance indices, which we plan to determine through our continued evaluation of legacy software and COTS products.

References

Groveman, B. S., Liang, E. Y., Starbuck, R. A., Tamkin, G. S., & Boland, D. E. (June 1994). A Recommended Approach to Rehosting IBM-Mainframe FORTRAN Software Systems to UNIX Workstations, *Fourth Annual CSC Technology Conference.* Atlanta, GA.

Integrated Satellite Control Human Computer Interface Working Group, Department of Defense (August 1993). *Human Computer Interface Standard, Draft 1.0, Appendix A1.*

Stottlemyer, A. R., Jaworski, A., & Costa, S. R. (October 1993). New Approaches to NASA Ground Data Systems, *Proceedings of the Forty-fourth International Astronautical Congress, Q.4.404.* Graz, Austria.

Werking, R. D., & Kulp, D. R. (September 1993). Developing the CSC Integrated Ground Support System, *Poster presented at the Seventh Annual AIAA/Utah State University on Small Satellites.* Logan, UT.

Wheal, C. A. (1993). Application of State Space Modeling Techniques to Satellite Operations, *Altair Aerospace Corporation.* Bowie, MD.

111154

N95-17569

354243

P. 8

SOFTWARE PROCESS ASSESSMENT (SPA)

Linda H. Rosenberg, Ph.D.
Unisys Government Systems
10265 Aerospace Drive
Lanham, MD 20706

Sylvia B. Sheppard
NASA/GSFC
Code 522
Greenbelt, MD 20771

Scott A. Butler
University of Maryland
Department of Psychology
College Park, MD 20742

Abstract

NASA's environment mirrors the changes taking place in the nation at large, i.e. workers are being asked to do more work with fewer resources. For software developers at NASA's Goddard Space Flight Center (GSFC), the effects of this change are that we must continue to produce quality code that is maintainable and reusable, but we must learn to produce it more efficiently and less expensively. To accomplish this goal, the Data Systems Technology Division (DSTD) at GSFC is trying a variety of both proven and state-of-the-art techniques for software development (e.g., object-oriented design, prototyping, designing for reuse, etc.).

In order to evaluate the effectiveness of these techniques, the Software Process Assessment (SPA) program was initiated. SPA was begun under the assumption that the effects of different software development processes, techniques, and tools, on the resulting product must be evaluated in an objective manner in order to assess any benefits that may have accrued. SPA involves the collection and analysis of software product and process data. These data include metrics such as effort, code changes, size, complexity, and code readability. This paper describes the SPA data collection and analysis methodology and presents examples of benefits realized thus far by DSTD's software developers and managers.

1 Introduction

Effective management of software development projects requires continual assessment of the development process and the resulting software product. The Software Process Assessment (SPA) program of the Software and Automation Systems Branch (Code 522) of the Goddard Space Flight Center (GSFC) was established four years ago in order to promote understanding of our software development process and to assure the quality of our software products. For the purposes of this paper, terms are defined as follows: "software process" is the set of activities and methods employed in the production of software; "measurements" are raw data relating to the development effort or the software; and "metrics" are combinations of measurements used to quantify a software attribute (IEEE-Std-610.12-1990).

SPA's primary objective is to understand the effects of different life cycles, project domains, development languages, design methodologies, and management techniques on resulting software products. We are interested in developing a process model that incorporates these issues and that supports quality assurance and quality control. At present, our guide for process improvement involves tracking and analyzing daily activities in the context of our experiences and lessons learned. These analyses will benefit on-going projects by reducing development times, decreasing development costs, decreasing maintenance costs, and increasing software reliability (Baumert & McWhitney, 1992). Future development efforts will benefit by having a more accurate basis for predictions about development costs and schedules.

The fundamental premise of SPA is that metrics will not be used to evaluate programmers or project managers. To foster confidence among the programmers, each programmer and project is identified by an identification number to maintain anonymity. Through working closely with SPA personnel, project managers use the metrics to improve or evaluate current development techniques. Guidelines for using the metrics are being developed to assist managers in interpreting project results.

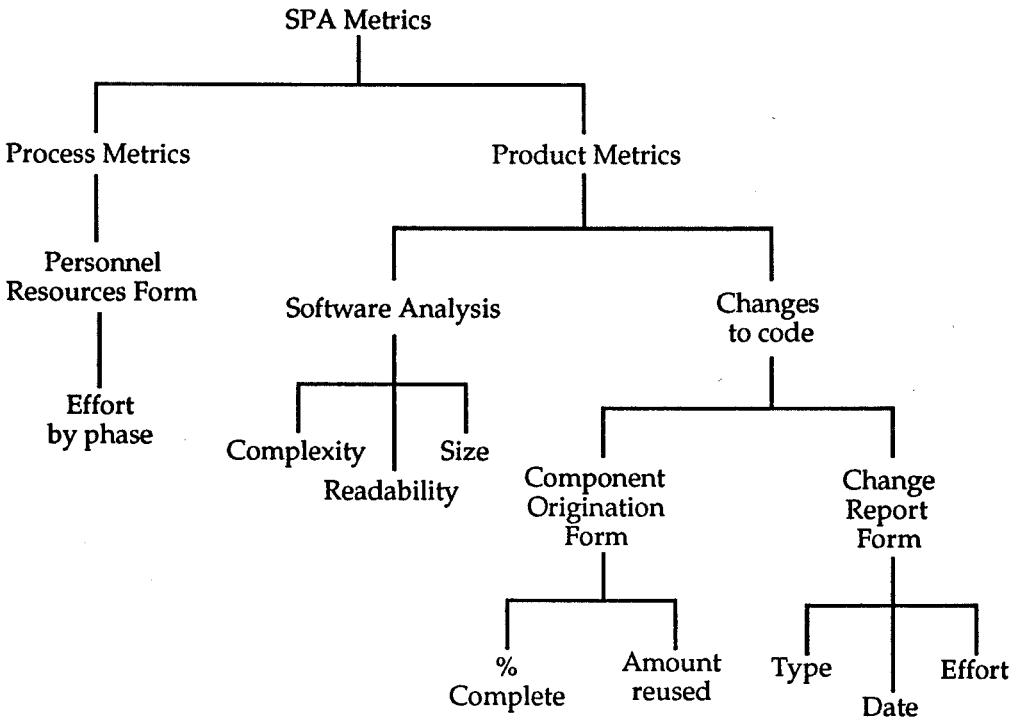


Figure 1: Measurable Components of the Software Process Assessment

2 SPA Metrics: Process, Product, and Changes

2.1 Process Metrics

SPA involves the evaluation of process and product metrics as indicated in Figure 1. To evaluate process, we focus on the application of resources, primarily personnel effort. By understanding how personnel resources are allocated in different phases, we can begin to determine how a project applied a particular life cycle model and the effects that life cycle had on the allocation of effort. This information can also assist in determining stability of requirements by tracking the amount of effort that was devoted to requirements specification. Requirement specification should occur in the initial phases of a project's life cycle; work on requirements later in the life cycle may indicate instability in the project definition (Baumert & McWhitney, 1992; Mills & Dyson, 1990).

2.2 Product Metrics

To evaluate a product, we analyze the software throughout development and after releases. Multiple analyses allow comparisons among releases and allow us to correlate effort metrics to change data. The frequency of analysis is determined by development phase and project manager requests.

Product assessments include metrics such as size, complexity, and readability (Rombach, 1990). We obtain these metrics using UX-Metric from SET Laboratories (Set Laboratories, 1990). UX-Metric produces McCabe's complexity metrics, counts GOTOs and comments, and calculates size metrics (IEEE-Std 1045-1992).

Size metrics refer to line counts, such as total lines of code (including comments and blank lines) and executable statements (measured by delimiters). Because we wish to compare metrics across different languages, we use executable statements as opposed to non-comment non-blank lines (NCNB). Executable statements are least affected by programmer style (Putnam & Myers, 1992).

Complexity metrics describe the logical structure of the individual code modules. We are initially evaluating the structure using McCabe's cyclomatic

complexity (McCabe, 1976), and the extent of the use of the GOTO statement (especially in object oriented design systems) (Booch, 1991). At a later time, we will include level of nesting, fan in and fan out.

Readability metrics include the use of comments and the average length of variable names. Using comments and meaningful variable names contributes to the reader's understanding of code. Readability metrics, as well as complexity metrics, are cited in the literature as contributing to understandability of the code, an issue for code reading during development and for later maintenance of the code (Putnam & Myers, 1992).

2.3 Changes to Code

Additionally, we track the types of changes made to the code, when they were made and why they were made (Baumert & McWhitney, 1992; SEL-87-008). Errors, usability issues, and modifications to requirements are all classified as changes. In short, a change is anything that causes a modification to the code once it has been submitted to the project library. Change data are collected from the time components are entered into the project library until the completion of the development effort and, sometimes, throughout the project maintenance phase. We are also investigating correlations between the number of changes per module/file and the code metrics.

3 Data Collection

The data collection process was designed to ensure that the metrics we collected would be reliable and relevant, i.e. the data can be used to draw valid conclusions and to answer specific questions (Baumert & McWhitney, 1992). The data collection forms are non-threatening, easy-to-use, and non-intrusive. All forms are on-line and are distributed and processed electronically.

SPA uses modified versions of three forms developed at NASA Goddard's Software Engineering Laboratory (SEL) (SEL-87-008). The forms were modified to encompass the range of activities and interests specific to the DSTD. The Personnel Resources Form (PRF) provides information about effort spent in various development activities. It is completed each week by all personnel performing either technical or management activities on a project. These activities have become an integral part of our software development process as opposed to mere adjuncts done at the discretion of the developers.

The Component Origination Form (COF) provides details about an individual software module. A COF is completed each time a component is added to the system library. One area of interest is the number of components generated "from scratch" as compared to the number that are reused (or modified and reused) from the DSTD Reuse Software Library.

The Change Report Form (CRF) describes a software change and provides a reason for the change. A CRF is completed by any person who implements a change to the system that involves modifications to components in the project-controlled source library.

4 Results

SPA data have been collected on over of thirty-five projects to date. The projects are diverse in application domain, use the waterfall or evolutionary prototyping life cycles, and are written in Ada, C, C++ or FORTRAN. Data for some projects were collected using the method described above. In other projects, completed code was obtained, but no process data were available.

4.1 Resource Analysis

The process data collection has yielded interesting results. One result is the use of metrics to drive the development of a process model. Figure 2 shows the total weekly hours by activity across the development of a C++ project currently in the third build. This chart can give management an indication of staffing requirements and can indicate the effects of events such as holidays, winter storms, and design reviews. When data from several projects of this size and type have been obtained, we hope to be able to build a model that will help estimate the staffing requirements for our specific development environment.

Besides aiding in the development of a planning profile for staffing, effort data can be used as feedback for current development efforts. One measure of the stability of a development process is the stability of activities within a phase; earlier phases should be largely completed before subsequent phases begin. For example, once a

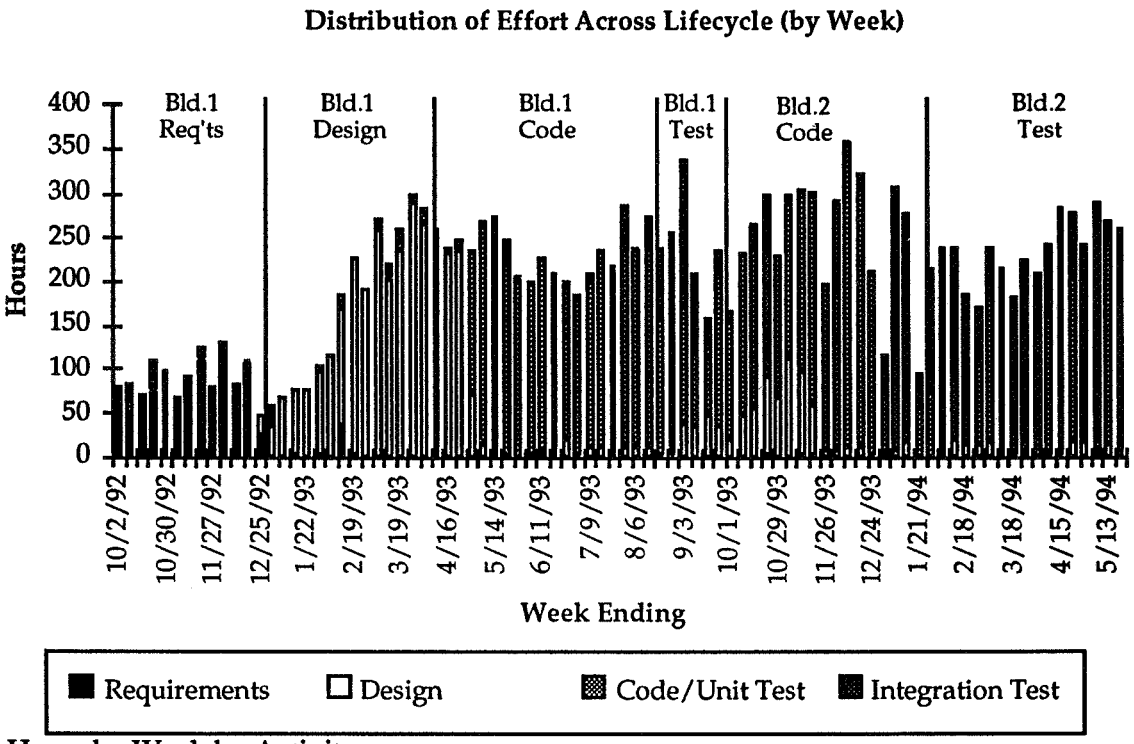


Figure 2: Hours by Week by Activity

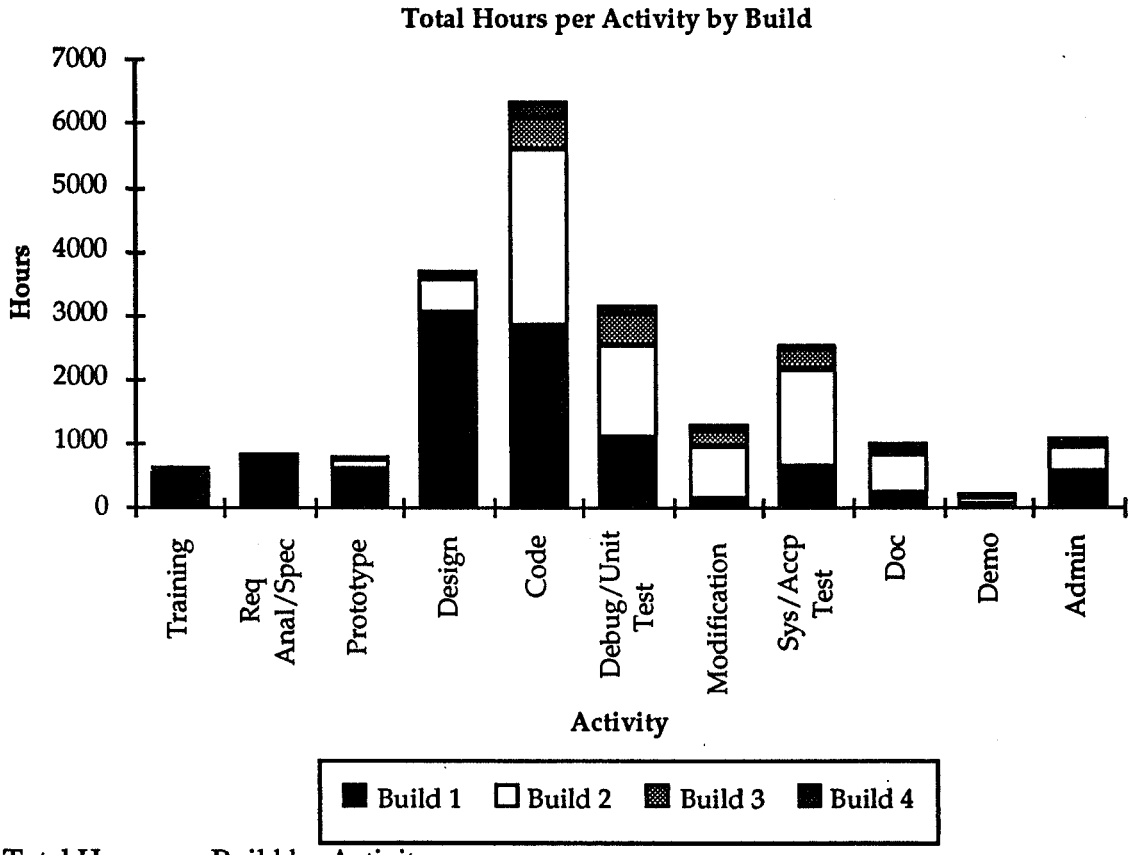


Figure 3: Total Hours per Build by Activity

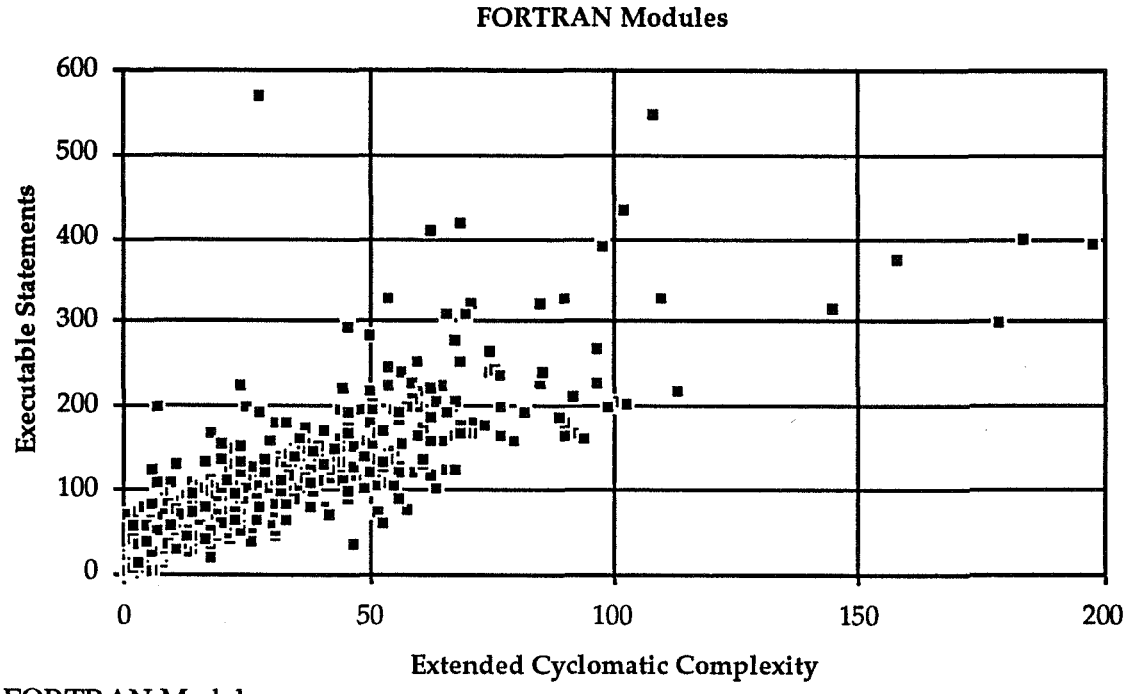


Figure 4: FORTRAN Modules

project has entered the coding phase, requirements-related activities should have been, for the most part, completed. In Figure 2, the design activity that begins on or about 9/3/93, was, in fact, in preparation for Build 2. Had this redesign been associated with Build 1, it would have been an indication of design instability and could have been costly to implement.

Figure 3 shows data from the same project, but with a more detailed breakout of life cycle activities. This graph shows that all requirement activity was completed in the first build. This is a good indicator of requirement stability. Additionally, the large design effort for Build 1 appears to have reduced the need for design in Builds 2 and 3. According to the project manager, the more difficult capabilities were added in Build 2, hence a larger amount of system testing was needed in that build.

4.2 Code Analysis

Code metrics can be used for identifying code that may be difficult to maintain and for identifying modules that may need additional testing. Modules with high complexity and/or large numbers of executable statements are prime candidates for the most extensive testing (Set Laboratories, 1990). These modules also need to be well-commented for readability (Putnam & Myers, 1992).

Figure 4 shows data for a FORTRAN project. Each square represents a module of code. This project contained 906 modules with a total of 75,537 executable statements. Most of the code was FORTRAN 77, but some was older FORTRAN IV code. This older code was difficult to maintain, but funding to rewrite it was not forthcoming. Figure 4 shows five modules (on the right-hand side of the graph) to be exceptionally high in complexity as well as being rather large, as measured by the number of executable statements. Further investigation identified those modules as part of the FORTRAN IV code. Using this chart, it was argued that code this large and complex was expensive to maintain, and an overall rewriting of the code was approved.

For projects currently under development, an effort is being made to prevent outliers such as the five that were identified in Figure 4. Analysis of modules as they are entered into the project library allows project managers to identify modules that need more testing, more extensive documentation and/or division into more manageable components.

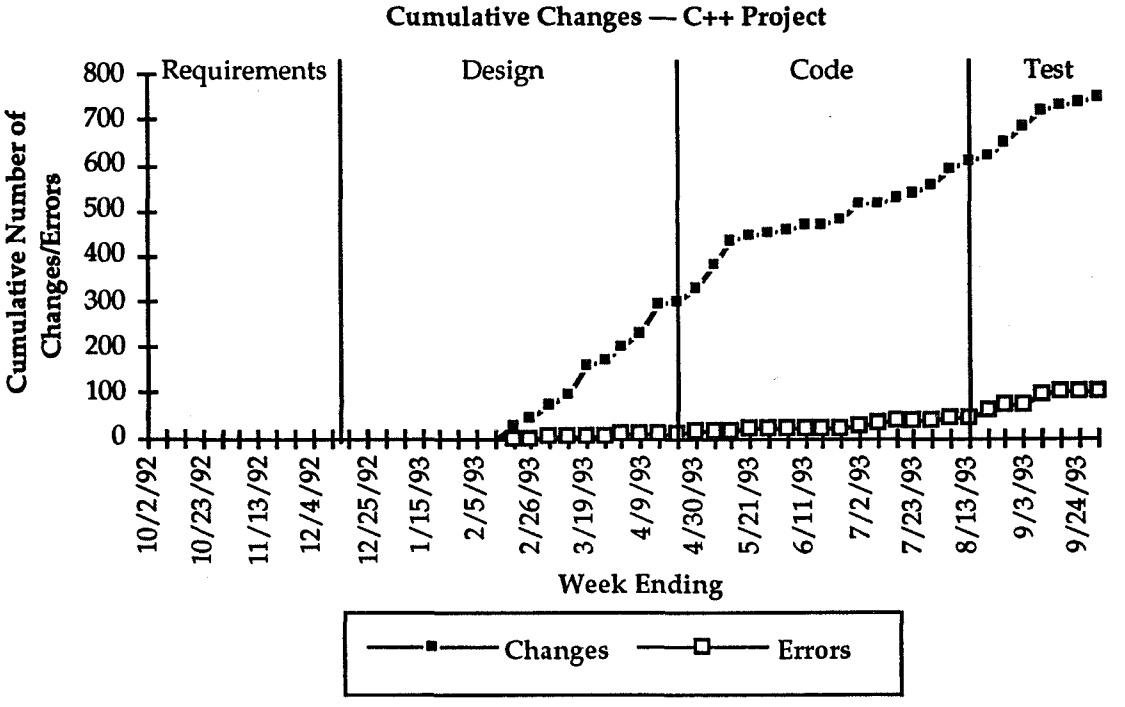


Figure 5: Number of Changes over Time by Development Phase

4.3 Software Change Analysis

Analyzing software changes can provide information about the development process as well as the product. In Figure 5, the black squares represent the cumulative changes for a C++ project currently in development. These changes may be due to planned enhancements, clarifications, requirements changes, or errors. It is expected that when this "total" curve levels off, most (if not all) errors will have been located, and the code will be ready for release. The white squares represent changes due to coding errors. In the initial phases, changes are not due to errors, but by the time of integration testing, most changes are the result of errors. Identifying trends such as this one helps us to allocate resources, both for testing and for error correction.

5 Discussion

The initial results of the SPA measurement-based process model are encouraging. We are meeting our objectives to learn about techniques in applying the life cycle in different application domains and with different languages. On the basis of management interest in the data and its application, SPA seems to be succeeding in supplying useful feedback during the development process. The metrics are also useful in identifying more efficient software development techniques.

The paragraphs that follow contain examples of how SPA feedback has helped developers address issues in the areas of design, training, budget, and quality.

Example 1: We compared two projects done by essentially the same personnel. On the first project, personnel used diagramming for both high level design and low level design. On the second project, they used diagramming on only the high level design and instead wrote class specifications in C++, the development language. Additionally, during low-level design for the second project, they standardized on very structured development techniques involving object-oriented programming and specific call-back mechanisms. Comparing SPA data from the two projects helped to convince management that the changes in methodology had, in fact, increased productivity. The new design methodology will be continued in the future.

Example 2: SPA metrics have been used to draw inferences about training and staffing. Information on personnel activities is being used to justify the number of hours allocated to various activities, e.g. more time spent on training or more time spent writing requirements/specifications. The analysis of an Ada project indicated that more time should have been spent training programmers to use Ada. The supposition is that if more time had been allocated early in the development cycle to learning to program in Ada, the efficiency of the project and the resulting code would have been improved.

Example 3: Another project we studied had finished under budget and ahead of schedule. One supposition for this outcome was that a larger percentage of civil service personnel had been added than had been projected or would normally have been used on a project of this size. (Only contractors' salaries are included in the cost of a project, so in a sense civil servants are "free" labor.) By using PRF data, we were able to differentiate the number of hours and types of activities performed by contractors and civil servants. As a result, management was reassured that the early completion of the project was, in fact, due to more efficient development techniques rather than an excess of civil servants. Because of this analysis, future projects will adopt these development techniques.

Example 4: SPA metrics have also been used to settle questions about code quality. An abbreviated development schedule caused management to question the robustness and maintainability of an application. Using the code metrics, we demonstrated that the majority of the code met the standards used at Johnson Space Center, which is known for its emphasis on software quality. Further, the metrics were used to argue successfully that portions of the code were reliable and maintainable and should not be rewritten.

6 Summary and Future Research

Metrics are often viewed by managers and programmers as threatening, but for the past four years they have been successfully collected and used to evaluate the development process model and software products in the DSTD at Goddard Space Flight Center. We attribute this success to the strict adherence to anonymity of personnel and projects and to non-intrusive data collection methods.

Although it is too early to quantify the financial benefits from these analyses, we have seen process improvements. For example, the need for training in object oriented design methods and in programming languages is determined at the start of new projects. Design and development techniques have been structured and formalized. Different testing methods are being identified and investigated.

The design and use of a measurement-driven process model has been educational. Everyone has become more aware of the structure of the development cycle and the characteristics that are related to quality program code. Through SPA, we continually evaluate our processes, making changes and improvements as necessary. Through the application of metrics, we expect the software development process to be more efficient, more predictable, and we expect higher quality products that are easier to maintain and reuse.

Our initial research used only a core metric set that focused primarily on code. There is much more to be done. We are working on correlating code metrics with discrepancy and change data in order to develop a baseline and tolerances that indicate the quality and reliability. Other code metrics, such as physical source statements, logical source statements and nesting levels are being investigated (Rombach, 1990). In the future we expect to use code metrics for certifying code before it is placed in the reuse library. We are also researching applicable metrics for other phases of the life cycle. The goal is to develop acceptability ranges for software metrics, at all phases of the life cycle, similar to those currently existing for hardware.

7 Acknowledgments

NASA's Software Engineering Laboratory (SEL) provided early and important mechanisms for understanding and managing the data collection process. Papers from the SEL, that appeared as early as 1977, have contributed greatly to the field of software engineering. We gratefully acknowledge the help of SEL personnel in getting the work cited in this paper started and in providing the data collection forms used in the first phase of these analyses.

8 References

Baumert, John H., & McWhitney, Mark S. (1992). Software measures and capability maturity model. Software Engineering Institute, Carnegie Mellon University, CMU/SEI-92-TR-25.

Booch, Grady, (1991). Object oriented design. Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA..

Software Engineering Laboratory, NASA Goddard Space Flight Center (1987). Data collection procedures for the rehosted SEL database. Software Engineering Laboratory Series, SEL-87-008.

IEEE Standard of Software Productivity Metrics, IEEE-Std 1045-1992, January, 1993.

IEEE Standard glossary of Software Engineering Terms, IEEE-Std-610.12-1990.

Mills, Harlan D., & Dyson, Peter B., (1990). Using metrics to quantify development, IEEE Software, March.

McCabe, Thomas J., (1976). A complexity measure. IEEE Transactions on Software Engineering, Vol. SE-2(4), Dec.

Putnam, Lawrence H., & Myers, Warren (1992). Measures for excellence: reliable software on time, within budget. Prentice-Hall, Inc., Englewood Cliffs, NJ.

Rombach, H. Dieter (1990). Design measurement: some lessons learned. IEEE Software, 7(2) March.

UX-Metrics, Set Laboratories Incorporated, Oregon, 1990.

111155 N95-17570

354244

TAKING ADVANTAGE OF GROUND DATA SYSTEMS ATTRIBUTES TO ACHIEVE QUALITY RESULTS IN TESTING SOFTWARE

Clayton B. Sigman
National Aeronautics and Space Administration
Goddard Space Flight Center

John T. Koslosky
National Aeronautics and Space Administration
Goddard Space Flight Center

Barbara H. Hageman
Integral Systems, Inc.

ABSTRACT

During the software development life cycle process, basic testing starts with the development team. At the end of the development process, an acceptance test is performed for the user to ensure that the deliverable is acceptable. Ideally, the delivery is an operational product with zero defects. However, the goal of zero defects is normally not achieved but is successful to various degrees. With the emphasis on building low cost ground support systems while maintaining a quality product, a key element in the test process is simulator capability. This paper reviews the Transportable Payload Operations Control Center (TPOCC) Advanced Spacecraft Simulator (TASS) test tool that is used in the acceptance test process for unmanned satellite operations control centers.

The TASS is designed to support the development, test, and operational environments of the Goddard Space Flight Center (GSFC) operations control centers. The TASS uses the same basic architecture as the operations control center. This architecture is characterized by its use of distributed processing, industry standards, commercial off-the-shelf (COTS) hardware and software components, and reusable software.

The TASS uses much of the same TPOCC architecture and reusable software that the operations control center developer uses. The TASS also makes use of reusable simulator software in the mission specific versions of the TASS. Very little new software needs to be developed, mainly mission specific telemetry commutation and command processing software.

By taking advantage of the ground data system attributes, successful software reuse for operational systems provides the opportunity to extend the reuse concept into the test area. Consistency in test approach is a major step in achieving quality results.

INTRODUCTION

The TASS is a crucial test tool used in the acceptance test process for unmanned satellite operations control centers (Payload Operations Control Centers and Mission Operations Centers) at GSFC. The TASS is used for development, integration, acceptance and regression testing phases of the system development cycle.

For a software delivery to be completely successful, it must meet or exceed all requirements, be delivered on time, within budget, and with minimum defects. Typically, varying degrees of success are achieved, and ideally the software should be delivered to the customer with zero defects.

To help support testing during the system life cycle, the TASS was designed to produce quality results in the testing process at the lowest possible cost. By utilizing proven testing fundamentals, commercial off-the-shelf

products, open industry standards, reusing software and taking advantage of the available infrastructure, the TASS provides a very cost effective way to complete effective software testing for numerous project software deliveries.

TESTING FUNDAMENTALS

The goals of a quality software delivery is to meet all the requirements, with zero defects, on time and within budget. To ensure quality software deliveries during the entire system life cycle, effective testing is necessary for all phases (unit testing, integration testing, acceptance testing, and regression testing). Figure 1 depicts a typical system life cycle.

In order to successfully test all phases of software development, a carefully developed test strategy must be used. First the test process should accurately identify defects in a cost effective manner and perform this process in the shortest possible time. Likewise, availability of the necessary test tools has to be maximized, and the test tool must be easy to use. Finally, the use of automation should be part of the process in order to shorten the testing time and eliminate human error.

OPERATIONS CONTROL CENTER

Next, an understanding of the operations control center infrastructure is necessary. At GSFC, the operations control center is the focal point for the health, safety, command and control of the unmanned satellite. The Flight Operations Team (FOT) commands and controls the spacecraft and monitors its health and safety via the ground data system.

The design of the ground data system is based on the TPOCC architecture and its reusable building blocks. In the operations control center, the ground data system includes a primary and a backup system. See figure 2. The architecture of each system is a distributed processing system consisting of a general purpose workstation, X-terminals, and a real-time front-end processor (FEP) connected by Ethernet. The FEP communicates with NASA Communications (Nascom), and ultimately the spacecraft, through a matrix switch using proprietary Nascom lines. These systems all utilize generic core TPOCC software, a software reuse library, which is the basis for which the mission software is built upon.

TYPICAL TEST SUPPORT SOLUTION

In the operations control center development environment, a tool to simulate the spacecraft and the status messages of the ground station is necessary to test the operations control center ground data systems. The TASS design concept is to make use of the software reuse library and be able to host its software on existing ground data systems. The TASS was developed with the capability to simulate the Nascom link protocols required to support various satellites, generate simulated spacecraft telemetry streams using the operations control center operational data base, and respond to spacecraft commands.

Unique implementations of spacecraft memory load and dump capabilities are provided. Network Control Center (NCC) communications protocol are simulated for Tracking and Data Relay Satellite System (TDRSS) support. In addition, the TASS validates spacecraft commands and alters the real-time telemetry stream in response to those commands. The user has the capability to alter the telemetry stream either by data base mnemonic or by specifying the individual bits in the telemetry frame or packet. Complexity can also be added by incorporating various dynamic models for the telemetry generating functions.

The TASS records all received Nascom blocks and all received spacecraft commands in history files that can be viewed for detailed analysis through the use of an off-line utility program. All system events, errors, operator input, procedure input recorded in the event log; and spacecraft memory images that are saved can be viewed by using the off-line utility programs. After completing the test, the user generates test reports using the report generation subsystem. These reports can later be used to evaluate the test results during the analysis process.

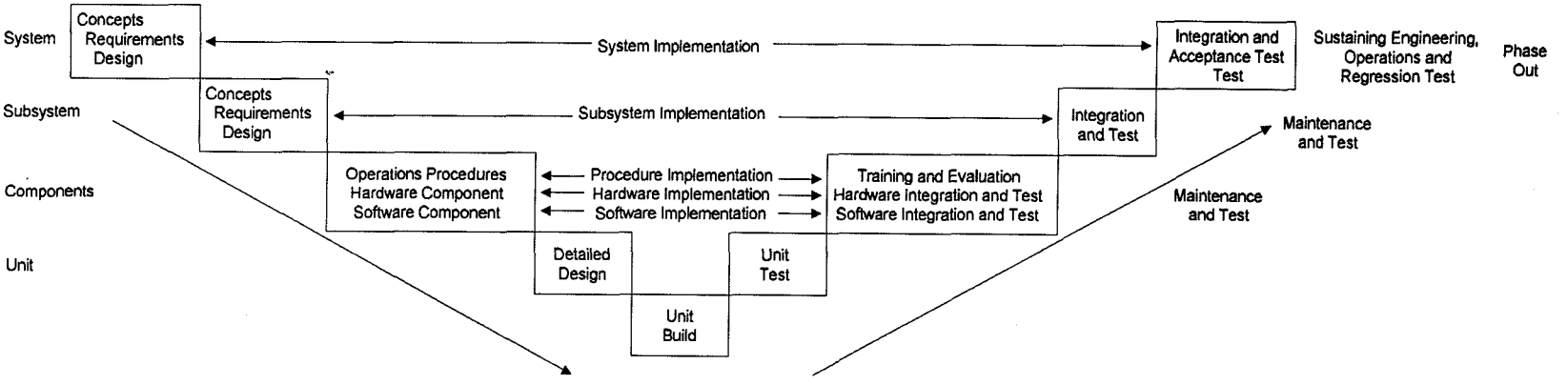


Figure 1. Typical Operations Control Center System Life Cycle

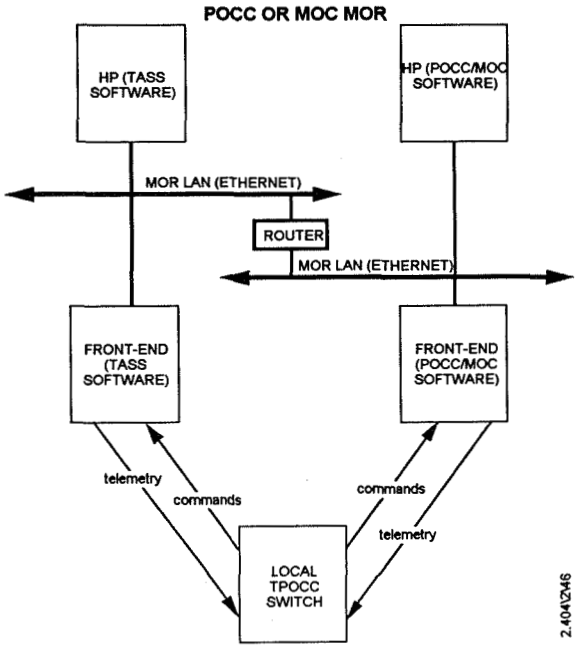


Figure 2. Typical Operations Control Center Ground Data Systems.

Since the test tool is used in all phases of the development cycle, it must be readily available, easy to use, and cost effective. In a typical operations control center, the design provides for a primary and a backup system. The TASS was designed so that it can be hosted on the primary or backup system; thus taking advantage of the control center architecture. Utilizing the backup system eliminates the hardware cost of an additional system, the need for additional floor space, power, cooling, and maintenance. It also eliminates the need to schedule Nascom communication lines and an externally located simulation system during the software development cycle. Likewise, in the development facilities with similar architectural capabilities, the TASS can be hosted on any system string and is essentially available at all times.

The hardware configuration that is used to host the TASS consists of two distributed computers connected by Ethernet and their associated peripherals as shown in Figure 3. One of the computer systems is a real-time VMEbus based front-end processor. It is used to receive and process spacecraft commands and to build and transmit the telemetry streams utilizing the Nascom link protocols. The other computer is a general purpose workstation used to configure, control, and monitor the FEP from one or more user terminal.

To minimize simulator development cost, the TASS utilizes a proven software reuse library. A major component of the software reuse library is the generic TPOCC software. Seventy-eight percent of the TASS software consists of these TPOCC building blocks. This reuse library is also the same core software building block for the operations control center. For the TASS, it is used mainly for the user interface (display and TPOCC Systems Test and Operations Language (TSTOL)) and the Nascom interface. Another component of the library used by the TASS is the TASS generic software that is shared across different missions. These components account for seventeen percent of the TASS software. Finally, only five percent of the software is specific to simulating each spacecraft. Figure 4 shows a breakout of the TASS software reuse for a typical mission. To further increase reuse, the TASS utilizes many industry standards, including C, TCP/IP, sockets, XDR, Motif, X11 and RPC.

Another major consideration in the design of the TASS is the user interface. First, to maximize usability, the TASS makes use of a graphical user interface (GUI) which is based on X Windows and fully adheres to the industry-standard OSF/Motif principles. Since a major portion of the software is common between the operations control center and the TASS, they maintain a consistent look and feel between both systems. Finally, an open dialog with the TASS users is maintained in order to provide continued improvement in the test process.

To help automate testing, user inputs from both the command line and the GUI are processed by the TSTOL, the control center script language. By utilizing TSTOL, it is possible to log user inputs into a text procedure file. This text procedure file can be edited and is used to execute an automated test or repeat a previous test under user control. The TASS provides a means for saving and restoring predefined test scenarios and results, telemetry stream contents, and data structures. This allows the user to repeat specific tests, retest

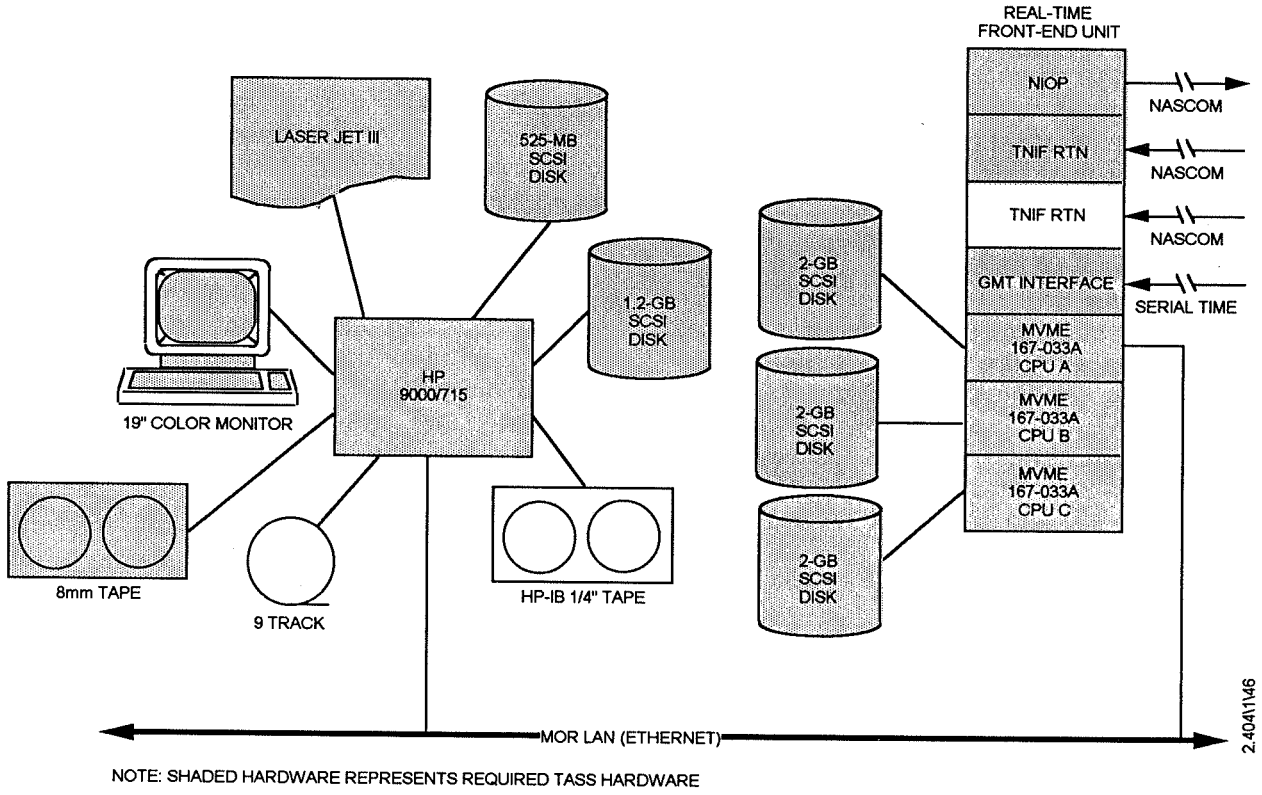


Figure 3. TASS Hardware Configuration

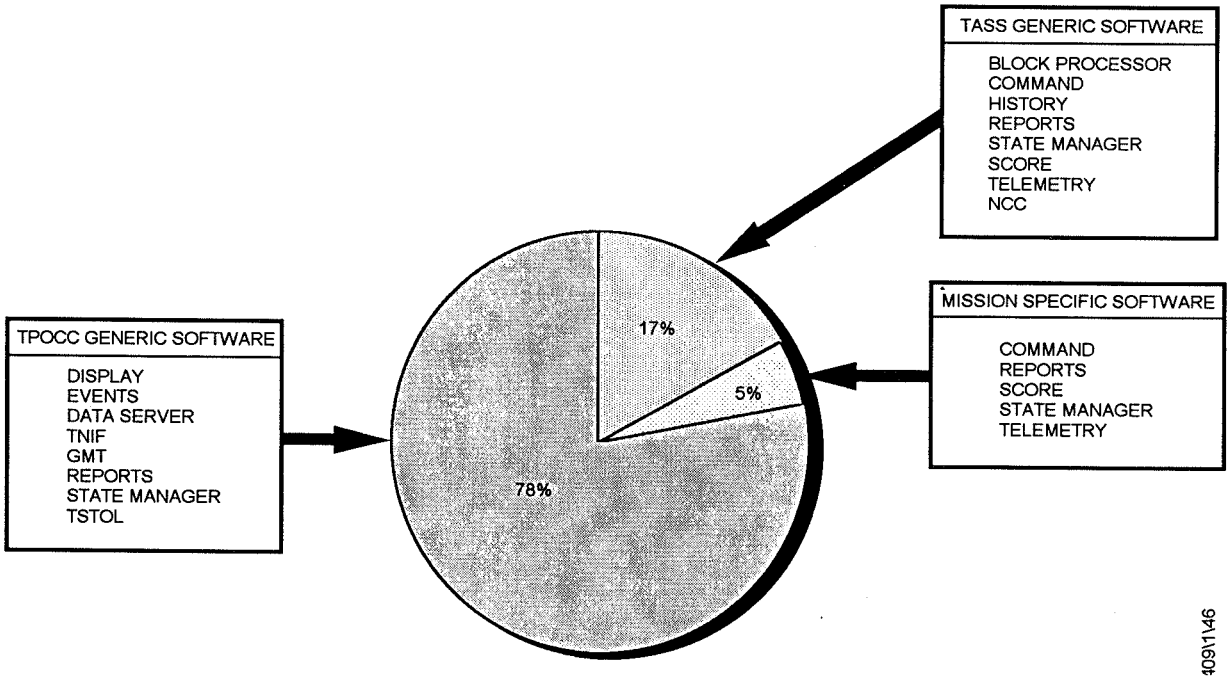


Figure 4. TASS Software Reuse

with known data, or continue testing from a given point in the test scenario.

Another planned feature that is being developed to automate testing is called Test/Score/Report. This function automates testing of the operations control center software in three areas: telemetry decommutation, spacecraft command processing, and spacecraft memory load and dump processing. The TASS system "tests" the operations control center software and provides a "score" based on the test results. Finally, the TASS system provides formatted "reports" that document each step performed during the test and the results of each step. These features help to test new deliveries and perform regression testing in the shortest time possible.

CONCLUSION

By taking advantage of the ground data system attributes, it is possible to achieve cost effective quality results in testing operations control center software. By using proven testing fundamentals, industry standards, reusing available hardware and software, maximizing usability and automation, it is possible to minimize the time and cost to perform quality software testing.

REFERENCES

National Aeronautic and Space Administration. (June 1994). *Transportable Payload Operations Control Center, (TPOCC) Advanced Spacecraft Simulator (TASS) System Requirements Document (Revision* 4). Goddard Space Flight Center, Greenbelt, MD

National Aeronautic and Space Administration. (May 1994). *Transportable Payload Operations Control Center (TPOCC) Advanced Spacecraft Simulator (TASS) System User's Guide for Release 7.* Goddard Space Flight Center, Greenbelt, MD

National Aeronautic and Space Administration. (May 1994). *Transportable Payload Operations Control Center (TPOCC) Advanced Spacecraft Simulator (TASS) Detailed Design Specification for Release 7.* Goddard Space Flight Center, Greenbelt, MD

National Aeronautic and Space Administration. (February 1994). *Transportable Payload Operations Control Center (TPOCC) Detailed Design Specification for Release 10.* Goddard Space Flight Center, Greenbelt, MD

Acknowledgments -- We wish to extend special thanks to the following personnel for their inspirational ideas used to formulate and implement the TASS: Carroll Dudley (National Aeronautic and Space Administration, Goddard Space Flight Center), Darlene Riddle (Integral Systems, Inc.), Luan Luu (Integral Systems, Inc.), and Nancy McCluer (Integral Systems, Inc.).

111156 N95-17571
354247 P.8

SCOS II - An Object Oriented Software Development Approach

*Martin Symonds, Steen Lynenskjold, Christian Müller.**

Computer Resources International A/S
Bregnerødvej 144
DK - 3460 Birkerød, Denmark

ABSTRACT

The Spacecraft Control and Operations System II (SCOS II), is intended to provide the generic mission control system infrastructure for future ESA missions. It represents a bold step forward in order to take advantage of state-of-the-art technology and current practices in the area of software engineering. Key features include:

- Use of Object Oriented Analysis and Design techniques
- Use of UNIX , C++ and a distributed architecture as the enabling implementation technology
- Goal of re-use for development, maintenance and mission specific software implementation
- Introduction of the concept of a spacecraft control model.

This paper touches upon some of the traditional beliefs surrounding Object Oriented development and describes their relevance to SCOS II. It gives rationale for why particular approaches were adopted and others not, and describes the impact of these decisions.

The development approach followed is discussed, highlighting the evolutionary nature of the overall process and the iterative nature of the various tasks carried out.

The emphasis of this paper is on the ***process*** of the development with the following being covered:

- The three phases of the SCOS II project - prototyping & analysis, design & implementation and configuration / delivery of mission specific systems
- The close co-operation and continual interaction with the users during the development
- The management approach - the split between client staff, industry and some of the required project management activities
- The lifecycle adopted being an enhancement of the ESA PSS-05 standard with SCOS II specific activities and approaches defined
- An examination of some of the difficulties encountered and the solutions adopted.

Finally, the lessons learned from the SCOS II experience are highlighted, identifying those issues to be used as feedback into future developments of this nature.

This paper does not intend to describe the finished product and its operation, but focusing on the journey to arrive there, concentrating therefore on the processes and not the products of the SCOS II software development.

INTRODUCTION

SCOS II

SCOS II (Spacecraft Control and Operations System II), ref. [10][11][12][13] is the latest of ESA's (European Space Agency), efforts to increase standardisation and reuse within its control systems. SCOS II has as a predecessor SCOS I which provides standard functionality for the telemetry processing chain and various data management features. These standard features such as telemetry displays, out of limits checking, database maintenance etc. were provided as a collection of middleware routines and tasks around which a mission would build its Telecommanding chain and any other mission specific components. SCOS I uses as front-end, non standard, custom built workstations connected to centralised VAX computers. An enhancement to SCOS I which has recently been made available provides the same underlying functionality but using Sun workstations connected to the VAX's.

SCOS II goes some steps further. In addition to the functions provided by SCOS I, it not only provides standard telecommanding facilities but is also designed to allow much more mission specific customisation of the kernel system. This customisation is readily available as a result of the Object Oriented approach and underlying technology adopted, and is outlined in the sections which follow.

* Martin Symonds (martin@msymonds.demon.co.uk), Steen Lynenskjold (steen@acm.org) and Christian Müller (cmueller@esoc.bitnet) are currently assigned to the European Space Operations Centre in Darmstadt, Germany. They have worked for **CRI** on the management, analysis, design, implementation and testing of the Application part of the SCOS II project under a contract with the European Space Agency.

The approach taken by the SCOS II project was designed to provide the maximum benefit from use of current "State of the art" tools and techniques in the field of Software Engineering. These were not chosen for their own sake, but in order to deliver very real benefits to the development lifecycle and the final SCOS II products. In particular, the use of Object Oriented Analysis and Design techniques, and a move towards an open distributed architecture based on the use of C++ running under Solaris on Sun workstations, complemented each other well. In addition, tools such as those used for user interface design and implementation helped the prototyping and user requirements definition considerably.

Probably the most important design driver was that SCOS II should be **generic**. That is, not only should it make use of the available technology and the re-usability provided by object orientation, but it should also ensure that the re-use is embedded in the design and not just the implementation.

For example, one can imagine the system needing to know about gyros, heaters and thrusters. To use the object oriented approach one could implement these as separate classes and then specialise from them in order to make different kinds of gyro, heater and thruster. The SCOS II approach however has found a way to ensure that gyros, heaters and thrusters can all be specialised from a single parent, called the System Element. The adopted client-server concept plus the distributed architecture brings a flexible system with high performance. It is these extra steps which will deliver some of the real power and benefit of SCOS II.

OOA\OOD

As well as the standard functional goals and requirements of a satellite control system, SCOS II has a number of other goals for ESA/ESOC. In particular these are centred around the concept of reuse of the software and tools used during the requirements definition, development and maintenance phases of SCOS II. SCOS II is also required to allow easy mission specific customisation of the kernel whilst providing for the mission specific components to be optionally later included into SCOS II. This is achieved by implementing a building block approach for both the design and use of SCOS II.

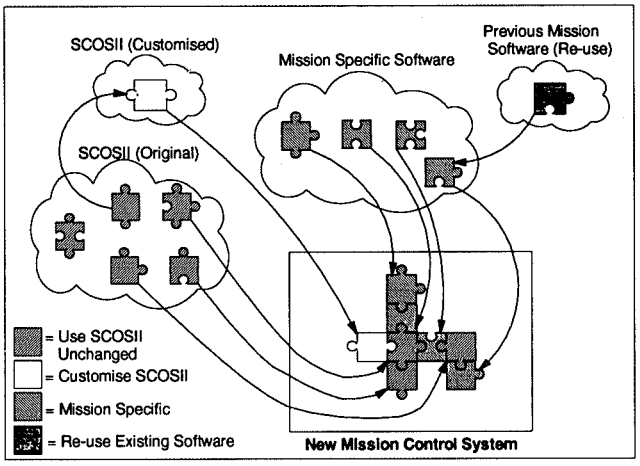


Figure 1 : SCOS II Building Block Approach

The concept behind this will allow a control system to be put together from the SCOS II supplied components, modified SCOS II components and mission specific components. This approach is illustrated in Figure 1 where the final components of the system are shown as being built from each of the various sources. This building block approach is supported by the use of C++ and the class libraries that the SCOS II project provides, allowing a "mix and match" approach to system construction as shown in Figure 1.

In addition to the goals and expected benefits for the developers, there are also changes occurring for the users. These changes include increased involvement in the analysis and design process, the capability to represent and control their spacecraft through the use of a model and changes in the physical appearance of the system.

One of the most significant changes that SCOS II users have had to come to grips with is the change in emphasis between focusing on the mechanisms used for controlling the spacecraft to focusing on the spacecraft itself. For example, the tendency in the past has been to think of commanding and monitoring of the spacecraft in terms of telecommands and telemetry, whereas the SCOS II approach encourages focus on the actual spacecraft and its components, i.e. those objects being commanded or monitored (gyros, heaters, thrusters etc.). This manifests itself primarily as a consequence of the Object Oriented Analysis and Design approach which allows the spacecraft model to be developed as part of the tasks carried out by the users when configuring the system. These modelling features allow the easy expression of physical, thermal and electrical relationships as well as abstract relationships as and when required by the users.

OBJECT ORIENTATION

The concepts of object orientation have been in the software industry for some years now but it is only recently that the tools, methods and experience have become readily available to allow the widespread take-up of this approach and the techniques it supports. The benefits of object orientation permeate the entire software development lifecycle, from the analysis of user requirements through to maintenance and operations. The major advantages of object orientation within the software development lifecycle can be summarised as follows:

- **Analysis of Problem Domain** - Allowing a better understanding of the problem domain; encouraging user/analyst interaction; providing a basis for evolution towards the design and implementation
- **Design of solution** - Encouraging identification and utilisation of underlying commonality within the problem domain; providing a means of concealing changes to the design specification; extending results of analysis phase
- **Maintenance and operations** - Promoting reuse of developed components; concealing low level code changes.

These advantages can be considered a result of the tools, methodology and languages used. In particular the object oriented concepts of encapsulation, inheritance and polymorphism allow a number of the advantages listed above to be realised.

The object oriented approach also supports an iterative lifecycle where iteration is considered part of the analysis and design process as further detail is added to the analysis/design model. Figure 2 below shows the iterative nature of the lifecycle approach taken, which can be compared with the traditional waterfall lifecycle in Figure 3.

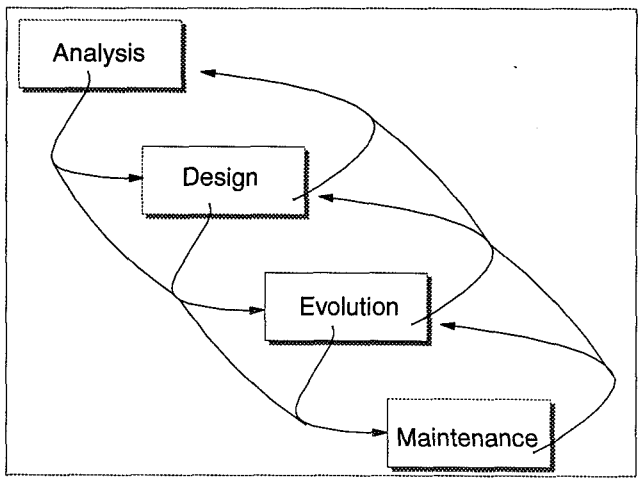


Figure 2 : Object Oriented Lifecycle

The major difference is that iteration and feedback is a fundamental part of the object oriented lifecycle, whereas for the traditional waterfall lifecycle, this feedback is generally only permitted to rectify errors. The object oriented approach allows the analysis results to be gradually expanded and refined with successive layers of detail until the design is complete. It is hence of outmost importance to

define each iteration and its products as part of the planning cycle.

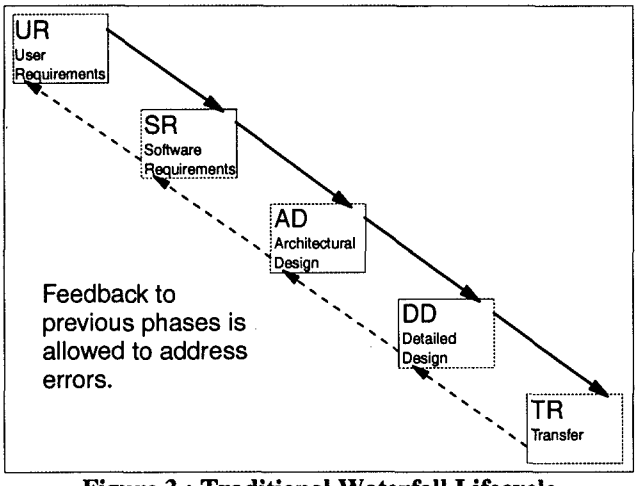


Figure 3 : Traditional Waterfall Lifecycle

SCOS II

In order to satisfy the demands placed upon SCOS II, the project was approached in two phases. In the first phase, the technology to be used was proven in terms of functionality and performance, and the initial analysis work was carried out in conjunction with a significant amount of user interface prototyping.

Once the technology had been proven and the initial analysis performed, the project moved into its main development phase which saw the underlying technical services being provided and the analysis/prototyping activities moving forward into design and implementation of those generic parts of the system identified in the analysis.

Whilst the initial phase was not a pilot project as such, it did allow the project team to get to grips with the technology, tools and problem domain, providing them with the means to determine the route to the system goals as part of the second phase.

The project team structure saw a peak of some 20 software engineers. Of these, 5 were client staff responsible for the overall management, technical management and system testing support. Industry was represented by two consortia, each of 3 companies with clearly defined responsibilities. The Application Team was responsible for providing the analysis of the problem domain and for ensuring that the users functional requirements were satisfied. This team also carried out extensive functional prototyping and is responsible for delivering SCOS II applications. The second industry team was responsible for providing the low level technical infrastructure such as software to handle the transmission and caching of data across the network.

DEVELOPMENT APPROACH

In describing the development approach, it is necessary to understand the standard ESOC activities, how these activities were mapped on to the phases adopted by SCOS II, the modified lifecycle used by SCOS II and what were the key features of the development under these constraints.

Activities

ESA software development projects are developed according to the ESA Software Engineering Standards, ref. [9]. These standards recognise five phases of the development lifecycle known as:

- **User Requirements Definition** - definition of the problem domain to be solved by the system to be procured
- **Software Requirements Definition** - Analysis of user requirements to define a model to allow satisfaction of these requirements
- **Architectural Design** - Design of the hardware and software architecture including data and control flow
- **Detailed Design** - Design, code and test of the system design

- **Transfer** - Installation of software in target environment; performance of acceptance testing

These have traditionally been performed using the traditional waterfall lifecycle shown in Figure 3.

Whilst this is a well proven method, it has a number of difficulties and inconsistencies. These are emphasised when attempting to use this approach in an object oriented environment. The major difficulty is that in the waterfall lifecycle, the output from one phase is the major driver for the following phase, and to a large extent stands alone. The object oriented approach however encourages successive refinement of the initial analysis model right through to the code, without being able to easily produce the corresponding breakpoints of a traditional lifecycle. This is demonstrated by Figure 4 which shows how the relationship between the successive phases of a traditional approach is less closely coupled to its previous phase than that of an object oriented lifecycle.

With the waterfall approach, there are clearly defined deliverables at the end of each phase, which stand alone. With the object oriented approach, each iteration sees further refinement and not necessarily a specific stand alone product. Each iteration product should be defined in a manner that it is tangible; hereby giving the management the necessary information to monitor progress.

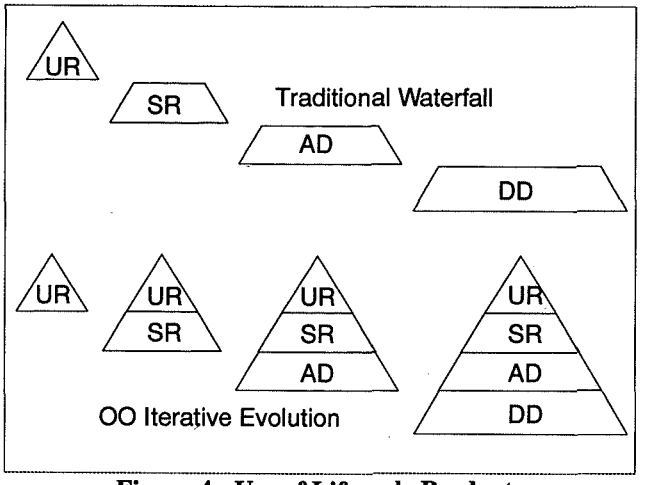


Figure 4 : Use of Lifecycle Products

Phases

The SCOS II approach required that the development be object oriented yet maintain, wherever possible, a correspondence to the ESA PSS-05 phases and deliverables. This was not easy and became more challenging as the project progressed.

The prototyping and analysis phase corresponded closely in some ways to the traditional lifecycle with the SCOS II development team producing an object oriented SRD (Software Requirements Document), ref. [2]. It was found that the nature of the object oriented analysis was such that the SRD activities could in fact be performed in parallel with the URD, with final SRD updates lagging behind the final release of the URD. During this phase, extensive iterative prototyping took place in order to:

- help elicit user requirements
- define user interfaces.

This proved to be a valuable exercise for the users.

The methodology followed for this analysis phase was the Coad/Yourdon method, ref. [5][6][7]. The object/class diagrams were created using the OMTool product which uses the Rumbaugh notation, ref. [3].

The Design and Implementation phase saw SCOS II covering the traditional AD/DD activities. Once more the nature of the object oriented approach is such that it was found that the SRD was more detailed than a traditional SRD and addressed a level of detail not normally found until AD activities. Similarly, the AD documentation progressed to a point where traditional DD issues were being addressed. It was also noted that the coding and detailed design activities were highly iterative, allowing the design and software to evolve together and to take into account feedback from users. Integration however has been more of a continuous process rather than one which progresses in clearly defined stages.

The next phase of the project which will commence in late 1994, will be to continue roll-out of the SCOS II kernel in readiness for customisation and enhancement by its first client missions. These deliveries will consist mainly of collections of C++ class libraries that will be used by the client missions as a basis for their custom and mission specific software development.

Lifecycle Considerations

The mismatch between the traditional lifecycle and that encouraged by the more iterative object oriented lifecycle continues to be a source of frustration. It is not easy to present documents for external review that correspond to some degree with the contents of traditional deliverables of that phase. Whilst less detail could have been documented during the SR and AD phases, the nature of the approach stimulates an analysis philosophy that repeatedly drops down into detail and back up again. It would be inefficient to ignore or document this information in another fashion.

Whilst SCOS II has produced documentation for review, such as the SRD, it has always been clear that the level of detail contained in these documents has generally been higher than the traditional documents. This reflects the lifecycle comparison diagram in Figure 5. This is also discussed in ref. [4].

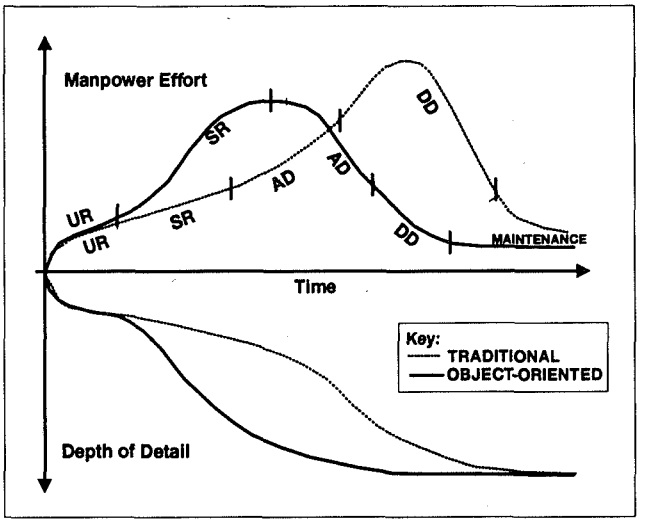


Figure 5 : Comparison of Traditional vs. Object Oriented Lifecycle Phases

Figure 5 shows some interesting comparisons between the traditional lifecycle and the object oriented lifecycle. In particular it demonstrates the OO approach reaching the same level of detail overall, but dropping down much sooner. Similarly, the corresponding amount of effort for an object oriented approach seems to occur rather earlier in the development cycle with the maintenance level is expected to be less.

SCOS II was able to take advantage of the possibility of overlapping phases. Thus whilst the UR/SR/AD/DD phases have overlapped this has not appeared to hinder development at all. This is something of a two edged sword; on the one hand it allows rapid progress towards an initial version/prototype, while on the other it does make the project management more complex.

Key Features

To summarise, the key features of SCOS II which have made it a success include:

- ***Prototyping*** - This helped considerably to elicit requirements, define interfaces and to demonstrate progress to the users.
- ***The iterative approach*** - Allowing frequent tangible results during both the analysis and design phases.
  - ◊ High level (Analysis and Design) - Manifested through successive refinement of the analysis model and refined user requirements.
  - ◊ Low level (Coding and Delivery) - Allowing successive deliveries to provide increased functionality.
- ***Object Orientation***
  - ◊ User interaction / co-operation (Through the Analysis Model) - Providing increased visibility of the design process, for the users and increased visibility of the problem domain for the developers.
  - ◊ Software Modularity - the implementation of the building block concept providing clean mechanisms for mission specific control systems.
- ***Management approach***
  - ◊ 3 groups (client and two teams from industry) - Allowing diverse skills to be brought to bear on a challenging, state of the art project.
  - ◊ split into technical and applications areas - Allowing clearly defined responsibilities
  - ◊ one infrastructure (bottom up) - Starting from the available technology and providing services for the applications.
  - ◊ one requirements (top down) - Starting from the requirements and implementing using the provided infrastructure services.

CONCLUSION

SCOS II is now well on the way to completion. It is a suitable opportunity to take a look back over the past couple of years and with the benefit of hindsight, draw some conclusions from the route that we have travelled.

The project may cost some 50% less than its predecessor infrastructure (SCOS I and MSSS)It is clear that the approach, technology and tools used have led to greater productivity in many ways.

The extent to which the benefits of ease of maintenance and later re-use will be realised, remains to be seen in client project applications. Based on the experience of flexibility to change and extent of re-use throughout the development phase, we have considerable confidence that this will be achieved

In retrospect it would have been immensely useful to have been able to develop a small pilot project. This would have enabled a number of management, analysis, design, implementation and standards issues to be resolved before SCOS II commenced. As it was these had to be addressed as part of the ongoing project work and sometimes distracted and indeed disrupted progress. To tackle a project of this nature and complexity where little appropriate expertise was available, and to add an increased level of complexity by making the SCOS II goal a generic system, is a high risk strategy. That this strategy is starting to pay off is a remarkable tribute to the skills and dedication of the people involved in the project.

REFERENCES

[1] SCOS II User Requirements Document, ESOC DOPS-SYS-URD-001-AMD, Issue 3, February 1994.

[2] SCOS II Software Requirements Document, ESOC SCOS II-SYS-SRD, Issue 0.6, June 1994.

[3] Object Oriented Modelling and Design, Rumbaugh et. Al, Prentice Hall 1991.

[4] Object Oriented Design with Applications, Grady Booch, Benjamin Cummings 1991.

[5] Object Oriented Analysis, Peter Coad/Edward Yourdon, Prentice Hall 1990.

[6] Object Oriented Design, Peter Coad/Edward Yourdon, Prentice Hall 1991.

[7] Object Oriented Programming, Peter Coad/Jill Nicola, Prentice Hall 1993.

[8] Modelling the World in States, Sally Schlaer/Stephen J. Mellor, Prentice Hall 1992.

[9] ESA Software Engineering Standards - Issue 2, ESA PSS-05-0 Issue 2, ESA Publications Division, February 1991

[10]SCOS II: ESA's New Generation of Mission Control Systems - The User's Perspective, P Kaufeler, M Pecchioli, I Shurmer, ESOC - these proceedings.

[11]A New Communication Protocol Family for a Distributed Spacecraft Control System, A Baldi, M Pace, ESOC - these proceedings.

[12]SCOS II: ESA's New Generation of Control Systems, M Jones, N Head, K Keyte, P Howard, S Lynenskjold - these proceedings.

[13]SCOS II OL: A Dedicated Language for Mission Operations, A Baldi, Dennis Elgaard, S Lynenskjold, M Pecchioli - these proceedings.

4. Modeling

* *Presented in Poster Session*

111157 N95-17572

354248 p-6

Evaluating Modeling Tools for the EDOS

Gordon Knoble and Frederick McCaleb
NASA GSFC Code 560
Greenbelt, MD 20771

Tanweer Aslam and Paul Nester
Computer Sciences Corporation
7700 Hubble Space Dr. Lanham, MD 20706

ABSTRACT

The Earth Observing System (EOS) Data and Operations System (EDOS) Project is developing a functional, system performance model to support the system implementation phase of the EDOS which is being designed and built by the Goddard Space Flight Center (GSFC). The EDOS Project will use modeling to meet two key objectives:
(1) Manage system design impacts introduced by unplanned changes in mission requirements and (2) evaluate evolutionary technology insertions throughout the development of the EDOS. To select a suitable modeling tool, the EDOS modeling team developed an approach for evaluating modeling tools and languages by deriving evaluation criteria from both the EDOS modeling requirements and the development plan. Essential and optional features for an appropriate modeling tool were identified and compared with known capabilities of several modeling tools. Vendors were also provided the opportunity to model a representative EDOS processing function to demonstrate the applicability of their modeling tool to the EDOS modeling requirements.

This paper emphasizes the importance of using a well defined approach for evaluating tools to model complex systems like the EDOS. The results of this evaluation study do not in any way signify the superiority of any one modeling tool since the results will vary with the specific modeling requirements of each project.

INTRODUCTION

A set of criteria specific to EDOS modeling requirements was developed for evaluating and selecting the most suitable modeling tool. These criteria identified potential strengths and weaknesses of modeling tools which would affect the EDOS model development time, enabling the team to initially screen each product prior to evaluating its capabilities in detail. This approach ensured timely adjustments to the overall EDOS modeling plan based on manpower estimates for implementing a useful EDOS model with the chosen tool.

The EDOS modeling tool evaluation criteria were divided into two categories, essential and optional. Essential criteria (e.g., modeling of high data rates) identified the modeling tools which could satisfactorily support the development of the EDOS model. Optional criteria (e.g., model software configuration management support) were used to identify modeling tool features which could aid in developing and operating the EDOS model by its users. A ranking and weighting scheme

PRECEDING PAGE BLANK NOT FILMED

enhanced the evaluation process further, ensuring that major differences between modeling tools were well understood by the modeling team. The evaluation approach was even further refined by requesting each prospective vendor to develop a sample model of a representative EDOS function and demonstrate the tool capabilities considered critical for developing the EDOS model. These demonstrations provided additional modeling tool discriminators, improving the team's understanding of the tool capabilities and enabling them to adjust the evaluation scores accordingly. A detailed matrix of evaluation results was developed on an EXCEL$_{TM}$ spreadsheet.

Major categories of the evaluation criteria included: Simulation data collection and generation of results, ease of model development, architecture representations (e.g., hardware, software, and data), user interface, additional development effort (necessary to compensate for modeling tool limitations and meet satisfactory requirements), model execution control, tool reliability, model platform choices and execution speed, documentation and training, vendor support, and portability of developed models. Additional criteria included modeling tool licensing and training costs, annual maintenance fees, and inherent risks (e.g., tool immaturity). The modeling tool with the best combination of evaluation score, least additional manhours estimated, and least implementation risk was selected as the most suitable tool for modeling the EDOS. If the evaluation results in more than one technically compliant candidate, then cost may well become the major deciding factor in the selection process.

The NASA / CSC EDOS modeling team consisted of experienced system engineers, each with at least ten years of experience in developing functional system performance models on various projects. Because of their current knowledge in the modeling field, they were readily able to identify a number of potential candidates for modeling the EDOS. Seven modeling packages and two modeling languages were identified as potential candidates. These were either commercial-off-the-shelf (COTS) items or available through NASA GSFC. Table 1 lists the candidate modeling packages and languages, in alphabetical order.

Table 1: Candidate Modeling Tools for EDOS

Item	Candidate Modeling Tool Name	Type	Developed by
1	Block Oriented Network Simulator (BONeS)	Package	Comdisco, Inc.
2	COMNET III	Package	CACI, Inc.
3	Data System Dynamic Simulator (DSDS)	Package	GSFC/STEL, Inc.
4	Extendible Computer System Simulator (ECSS)	Language	NTIS
5	General Purpose Simulation System (GPSS V)	Language	IBM
6	L•NET and NETWORK II.5	Package	CACI, Inc.
7	OPNET	Package	MIL3, Inc.
8	Quantitative computer Assisted System Evaluator for Reliability and Timing (QASE RT)	Package	AST, Inc.
9	SLAMSYSTEM 2.0	Package	Pritsker Corp.

EVALUATION APPROACH

The EDOS modeling team developed a well defined, structured approach to evaluate modeling tools, consisting of the following activities:

- Defining evaluation criteria
- Identifying available modeling tools
- Screening modeling tools against essential criteria
- Evaluating modeling tools in detail
- Requesting vendors to model a sample processing function
- Selecting the most suitable modeling tool for EDOS

Defining Evaluation Criteria

The EDOS modeling requirements document and the EDOS modeling plan were used in identifying and defining a uniform set of evaluation criteria for modeling tool packages and languages. A total of 12 evaluation categories (EC) consisting of 101 essential features and 24 optional features were identified. The categories are listed in Table 2.

Identifying Available Modeling Tools

Identifying suitable modeling packages and languages as potential candidates for modeling the EDOS was the second step. The experience of the modeling team members, as well as a search of available literature, produced several candidates. This was not intended to be an exhaustive search and many packages were not identified simply due to the lack of available time.

Screening Modeling Tools against Essential Criteria

All candidate modeling tools were evaluated against the essential criteria. After an initial screening, several modeling packages and languages designed for specialized applications (such as packet switching) were clearly not suitable for modeling the EDOS and were rejected from further consideration.

Detailed Evaluation of Modeling Tools

The detailed evaluation assessed the capabilities of each modeling tool qualitatively. The following scoring scheme was used in the detailed evaluations.

Scoring Scheme

A scoring scheme, ranging from "0" to "5", was used to evaluate the modeling tools in detail:

0: The modeling tool has no capability (fail).

1: Only minimal (poor) capability is provided, requiring extensive work to overcome the problem. The additional effort was estimated and included in the detailed evaluation matrix.

2: The capability is less than satisfactory (fair), requiring some work compensate for the deficiency. The additional effort was estimated and included in the detailed evaluation matrix.

3: The tool provides a satisfactory (average) capability.

4: The tool provides more than a satisfactory (good) capability.

5: The tool provides an excellent capability.

Table 2: EDOS Modeling Tools Evaluation Criteria

EVALUATION CRITERIA	Evaluation Category / Weight	Examples of ESSENTIAL FEATURES	Examples of OPTIONAL FEATURES
Data Collection and Output Generation (e.g. flexibility of selecting measurement points and accuracy of results)	12	Self contained production of output results, can produce trace, snapshots, summary and periodic reports	Can transfer results to analytical tools
Ease of Development (e.g. built-in features, formulae and predefined elements)	11	Predefined elements for statistics collection and measurement	Able to automatically verify model completeness
Architecture Representation (e.g. representations of data, time, functions, HW and SW elements and/or resources)	10	Accepts several sources of stimuli	Support modeling of functions independent of system specific H/W and S/W design
User Interface	9	Display/print simulation configurations, input parameters, and results	Tool user's interface (ICONS, GUI, etc. are utilized)
Model Modification (e.g. modification of parameters and configuration of model resources)	8	Ability to propagate parameters thoughout model structure	*(None were identified)*
Development Effort/ Schedule	7	Support implementation in 6 months	*(None were identified)*
Execution Control (e.g. flexibility of simulation control, enabling and disabling of functions and resources)	6	Single functions, grouped functions, and message logging	*(None were identified)*
Tool Maturity (Trusted by users)	5	Tool is mature, field usage is greater than 1 yr	Number of copies sold/licensed to users
Platform and (Run Time e.g. flexibility of choosing and upgrading a platform)	4	*(None were identified)*	Flexibility of selecting modeling platforms (machines supported: PC(DOS), Macintosh, Windows, OS/2, UNIX, etc.)
Documentation and Training	3	User manuals and training courses	*(None were identified)*
Vendor Support (e.g. during sample problem modeling and future commitments)	2	Availability of product support personnel	*(None were identified)*
Developed Model Portability (e.g. due to model growth and platform upgrade)	1	*(None were identified)*	Developed Model Portability to support growth

Assessment of Additional Development Effort

Modeling tool capabilities earning a score of 1 or 2 were considered deficient. The EDOS modeling team carefully reviewed these deficiencies and assessed the feasibility of correcting them with additional development effort. Previous model development experience with similar modeling packages and languages aided in assessing the number of manhours required to compensate for any shortcomings. Consulting with modeling tool vendors also aided in arriving at the most conservative estimates for correcting the deficiencies, if possible.

Modeling of a Sample Processing Function

This step of the evaluation approach was invaluable in the selection process. The EDOS modeling team prepared a sample modeling problem, generic in nature, representing an aggregation of typical processing functions required for EDOS. Each modeling tool vendor was asked to use the sample processing function to prepare a sample model, without cost to the project, to demonstrate the capabilities of their tool in support of the evaluation. Four vendors chose to model the sample processing function free of charge to demonstrate the capabilities of their tools; two did not (three did not pass the initial screening). Models of the sample function were not developed with modeling languages because of the extensive effort required by CSC personnel. There were no disqualifications of modeling package or modeling language vendors if they chose not to develop and demonstrate the sample processing function model. However, the demonstrations of the sample model enabled the EDOS modeling team to accurate assess the capabilities of those vendors' modeling tools.

Selection of the Most Suitable Modeling Tool for EDOS

All modeling tools meeting all essential criteria participated in this final evaluation activity. The following steps were used to identify the most suitable modeling tool for EDOS:

a. The total score for each modeling tool was calculated by adding all scores for each evaluation category (a total of 12).

b. The total effective cost for each modeling tool was calculated by adding modeling tool software cost, training cost, and cost for maintaining the tool for four years.

c. The total additional development effort required to compensate for deficiencies of a modeling tool and to improve its performance to a satisfactory level was calculated.

d. A risk factor (low, medium and high) for each modeling tool was assessed based on the results of detailed evaluation and the amount of additional development effort (manhours) required to improve the tool performance to a satisfactory level.

e. The modeling tool with the best combination of detailed evaluation score, lowest manhours for additional development effort, and least implementation risk was selected as the most suitable tool for modeling the EDOS.

SUMMARY OF EVALUATION RESULTS

Of the nine candidate modeling tools, only six: BONeS, DSDS+, OPNET, QASE RT, ECSS II, and GPSS V were fully evaluated. The development manhour estimates for the two modeling languages, ECSS II and GPSS V were beyond the scope of the modeling schedule. Of the remaining four modeling packages, DSDS+ and QASE RT were chosen as the most cost effective modeling tools which meet or exceed the EDOS modeling evaluation criteria. The data stream feature of DSDS+ enables modeling of scenarios spanning several days and weeks. The separate HW and SW architecture components of QASE RT provide a more realistic, graphical representation of the EDOS.

LESSONS LEARNED

The following key lessons were learned while evaluating the modeling tools for EDOS:

1. The modeling tool criteria should be developed from the modeling requirements and objectives specified for a candidate system. Therefore, system requirements and plans describing the modeling objectives should be complete before defining the modeling tool evaluation criteria.

2. The modeling tool evaluation criteria should carefully distinguish the essential and optional features considered. Non-critical requirements having little impact on the system development must not be allowed to influence the modeling tool selection.

3. Predetermining optional features desired can prevent the evaluation process from being misled by a single interesting aspect of a modeling tool. Several tools had spectacular features which, while very impressive, were not applicable.

4. Vendor development of a sample model of a representative system function to demonstrate the real strengths and weaknesses of a modeling tool can ease the completion of the modeling tool evaluation work in a single demonstration session.

5. The best results are achieved by team evaluation of modeling tool capabilities, which aids in balancing any bias.

6. There is no perfect modeling tool for any system. Use of additional effort, if not major, should not be overlooked for overcoming minor deficiencies of an otherwise robust modeling tool before eliminating it from further consideration.

7. The number of discrete events required for modeling a function has an extremely detrimental effect on the runtime ratio between simulated time and real time, due mainly to the exceptionally high packet rates. While this risk is dependent upon the speed of the platform selected, ways should be investigated early on to minimize it by properly designing the model's structure.

111158 N95-17573

351250 p.8

SOLAR AND HELIOSPHERIC OBSERVATORY (SOHO) EXPERIMENTERS' OPERATIONS FACILITY (EOF)

Eliane Larduinat, AlliedSignal Technical Services Corporation
William Potter, Goddard Space Flight Center, Code 514

ABSTACT

This paper describes the SOHO Instrumenters' Operations Facility (EOF) project. The EOF is the element of the SOHO ground system at the Goddard Space Flight Center that provides the interface between the SOHO scientists and the other ground system elements. This paper first describes the development context of the SOHO EOF. It provides an overview of the SOHO mission within the International Solar-Terrestrial Physics (ISTP) project, and discusses the SOHO scientific objectives. The second part of this paper presents the implementation of the SOHO EOF, its innovative features, its possible applications to other missions, and its potential for use as part of a fully integrated ground control system.

Keywords:
Solar and Heliospheric Observatory (SOHO),
Instrumenters' Operations Facility (EOF),
EOF Core System (ECS).

INTRODUCTION

The SOHO mission is part of the ISTP program. The SOHO EOF is the focal point for instrument operations, experiment planning and science data analysis. The EOF will support the instrumenters in three main functional areas: (1) commanding and monitoring of the instruments' health and safety, (2) receiving and archiving telemetry data, and (3) planning and scheduling of coordinated scientific observations. The particularities of the SOHO mission have dictated and influenced the design of the ECS.

This paper presents the software design for the ECS as well as the physical architecture of the EOF. It also discusses the various choices made, cost savings and risk mitigation realized and the possibilities of reuse of the SOHO EOF for other missions.

SOHO MISSION OVERVIEW

The ISTP program is an international space exploration program involving spacecraft built and managed by the National Aeronautics and Space Administration (NASA), the European Space Agency (ESA) and the Institute of Space and Astronautical Science (ISAS). This space program is coordinated with ground-based and theory investigations. Its intent is to coordinate worldwide studies of Sun-Earth plasma interaction, solar and heliospheric physics and global geospace physics. The ISTP program involves several spacecraft: SOHO, the Plasma Turbulence Laboratory (CLUSTER), the Geomagnetic Tail (GEOTAIL), the WIND spacecraft and the POLAR spacecraft.

SOHO is a joint venture between ESA and NASA: ESA provides the spacecraft that is built and tested in Europe and NASA provides the launch vehicle, launch services and the ground segment to support all pre-launch activities and in-flight operations. SOHO is scheduled to be launched in July 1995 and will be injected in a halo orbit around the L1 Sun-Earth Lagrangian point, about 1.5 million kilometers sunward from the Earth. The SOHO spacecraft will be three-axis stabilized and pointing to the Sun. The total mass will be about 1350 kg and 750 Watts power will be provided by the solar panels. The payload will weigh about 650 kg and consume 350 Watts in orbit.

The SOHO mission duration is 2 years and 5 months and will consist of three main phases:
(1) Launch and early orbit phase which starts at liftoff and includes the coasting period in parking orbit.
(2) Transfer trajectory phase during which the spacecraft will travel from Earth orbit to the halo orbit (Some science observations may begin during this phase).
(3) Halo orbit phase which starts with the commissioning of the service module and the on-board instruments (approximately one month), after which the nominal routine operations will start for a duration of at least 2 years.
SOHO is equipped with sufficient on board

consumables for an extra four years in orbit. SOHO will carry eleven on-board instruments.

SOHO Scientific objectives

The SOHO scientific objectives are to study (1) the structure (density, temperature and velocity fields) and dynamics of the outer solar atmosphere, (2) the solar wind and its relation to the solar atmosphere, and (3) the structure, chemical composition, and dynamics of the solar interior.

SOHO will carry a set of telescopes to study phenomena initiated below the photosphere, and propagating through the photosphere, chromosphere, and transition region into the corona. They will investigate problems such as how the corona is heated and transformed into the solar wind that blows past the Earth.

Spectrometers will study the emission and absorption lines produced by the ions present in the different regions of the solar atmosphere, allowing to determine densities, temperatures and velocities in the changing structures. These measurements will be complemented by the "in situ" study of the composition and energies of the solar wind: particle detectors will sample the solar wind as SOHO passes through it.

While the solar interior is the region that generates the kinetic and magnetic energy driving outer atmospheric processes, almost no direct information can be obtained about any region below the photosphere. The neutrinos generated by the nuclear reactions taking place in the core, are the only direct radiation that reaches us from below the photosphere. Helioseismology is a relatively new technique developed in the last two decades, allows us to study the stratification and the dynamic aspects of the solar interior. It analyses the acoustic and gravity waves that propagate through the interior of the Sun and can be observed as oscillatory motions of the photosphere. The analysis of these oscillations allow us to determine the characteristics of the resonant cavities in which they resonate, much in the same way as the Earth's seismic waves are used to determine the structure of the Earth interior. To study the solar interior, SOHO will carry a complement of instruments whose aim is to study the oscillations at the solar surface by measuring the velocity (via the Doppler effect) and intensity changes produced by pressure and gravity waves. This requires both high resolution imaging and long uninterrupted time series of observations. In addition, because it is of prime importance to understand the structure of the Sun in relation to the oscillation measurements, the total solar irradiance and its variations will be measured.

SOHO Instrumentation

The SOHO instruments can be divided into three main research groups: helioseismology, solar atmospheric remote sensing, and "in situ" solar wind measurements. Table 1 provides a list of the eleven SOHO instruments, indicating the corresponding research group and the primary institution responsible for their development.

The helioseismology instruments, GOLF, MDI and VIRGO, primarily aim at the study of those parts of the solar oscillations spectrum that cannot be obtained from the ground because of noise effects introduced by the Earth's diurnal rotation as well as the transparency and seeing fluctuations of the Earth's atmosphere.

The solar atmospheric remote sensing instruments, CDS, EIT, LASCO, SUMER, SWAN and UVCS, constitute a set of telescopes and spectrometers studying the dynamic phenomena that take place in the solar atmosphere at and above the chromosphere. The plasma will be studied by spectroscopic measurements and high resolution images at different levels of the solar atmosphere.

The "in situ" investigation of the solar wind is carried out by CELIAS and CEPAC, that will determine the elemental and isotopic abundance, the ionic charge states and velocity distributions of ions originating in the solar atmosphere. The energy ranges covered will allow us to study ion fractionation and acceleration from the "slow" solar wind through solar flares.

SOHO will be placed in a halo orbit around the L1 libration point. The halo orbit will have a period of 180 days and has been chosen

Table 1. SOHO Instruments

Instrument Name		Primary Institution
Helioseismology		
GOLF	Global Oscillations at Low Frequencies	Institut d' Astrophysique Spatiale, France
MDI	Michelson Doppler Imager	Stanford University, USA
VIRGO	Variability of Solar Iradiance and Gravity Oscillations	Physikalisch-Meterologisches Observatorium Daos, Switzerland
Solar atmospheric remote sensing		
CDS	Coronal Diagnostic Spectrometer	Rutherford Appleton Laboratory, U.K.
EIT	Extreme-ultraviolet Imaging Telescope	Institut d'Astrophysique Spatiale, France
LASCO	Large Angle Spectrometric Coronagraph	Naval Research Laboratory, USA
SUMER	Solar Ultraviolet Measurements of Emitted Radiation	Max Planck Institute, Germany
SWAN	Solar Wind Anisotropies	Service d'Aeronomie du CNRS, France
UVCS	Ultraviolet Coronagraph Spectrometer	Smithsonian Astrophysical Observatory, USA
"In situ" solar wind measurements		
CELIAS	Charge, Element and Isotope Analysis System	Max Planck Institute, Germany
CEPAC	COSTEP-ERNE Particle Analysis Collaboration	University of Turku, Finland

because it provides a smooth Sun-spacecraft velocity change, which is appropriate for helioseismology, it is permanently outside of the magnetosphere, which is appropriate for the "in situ" sampling of the solar wind and particles, and it allows permanent observation of the Sun, which is appropriate for all the investigations.

During in-orbit operations, the Deep Space Network (DSN) will receive telemetry during three short (1.3 hours) and one long (8 hours) passes per day. Outside of passes, the science data will be recorded on-board and played back during the short passes. The MDI instrument generates a high rate data stream that will be transmitted only during the long station pass. For two consecutive months per year, DSN will support continuous data transmission, including MDI high rate data. Whenever there is data transmission, the basic science data (40 kbits per second) will be available in near real-time at the EOF.

EOF within the SOHO Ground System. The SOHO EOF is part of the NASA Goddard Space Flight Center ground system where it is co-located with the Payload Operations Control Center and the Command Management System. The functions within the EOF are focused on instrument operations. A separate analysis facility, dedicated to the scientific analysis of the SOHO data, will be located in a separate building at the Goddard Space Flight Center.

The EOF is comprised of two main elements:

- The ECS which provides the communications between the instrumenters and other elements of the SOHO ground system. The ECS includes hardware and software to support the primary functions of instrument commanding, telemetry reception, distribution and archiving, and science planning and scheduling. The ECS includes two specialized workstations: the science operations coordinator's workstation and the project scientist workstation.
- The Instrumenters WorkStations (IWS) which include hardware and software provided by the individual instrument teams dedicated to the operation of a given instrument and its science analysis for planning purpose.

The instrumenters may be "resident instrumenters" and be located at the EOF where they have data processing equipment, or

IWSs. The ECS supports near-real-time commanding capabilities and distribution of real-time telemetry for the resident instrumenters. The "remote instrumenters" are located outside of the EOF, that is at their home institution in the US or in Europe. Mainly for security reasons, they may only communicate with the EOF via file transfer. They do not have access to the near-real-time commanding and real-time telemetry distribution capabilities. They can perform preplanned commanding and they can access the telemetry data archived within the ECS. They may also use the telephone or facsimile to communicate with the flight operations team or with an EOF resident team member in order to request changes in their instrument status.

The major ground systems elements that interface with the SOHO EOF are:

- The Information Processing Division (IPD) Packet Processor (Pacor) which captures the telemetry data from DSN via NASCOM and transfers the real-time telemetry to the EOF.
- The IPD Data Distribution Facility (DDF) which provides quicklook telemetry files (mainly tape recorder dumps) to the EOF.
- The ISTP Central Data Handling Facility (CDHF) which provides orbit and attitude data to the EOF and receives other mission support data from the EOF.
- The Command Management System. which serves as the intermediary between the EOF and the Payload Operations Control Center for instrumenter commanding activities.

The ECS will communicate via the NASA Science Internet network using file transfers with international observatories and scientific institutions including, but not limited to, the instrumenters home institutions, ESA, the National Solar Observatory (NSO), the ISTP Science Planning and Operations Facility (SPOF), and the NASA Space Science Data Center (NSSDC).

SOHO EOF DESIGN CONCEPTS

Several considerations and choices have highly influenced the EOF design.

- Conformity with the spacecraft integration and test environment.

In order to minimize development efforts on the part of the instrumenters, the ECS interface with the instrumenters was closely modeled after the interface provided by the spacecraft contractors during the integration and test phase. Some modifications have been necessary to go from a test to an operational environment, but the efforts to maintain that protocol as much as possible have greatly facilitated the integration of the various instrument teamswith the ECS.

- User involvement.

The EOF users were involved as much and as early as possible. Scientists and members of the flight operations team actively participated in the definition of the functional requirements. Additionally, an interface control document was developed very early in the project life cycle. This was of great benefit when dealing with instrument teams that had little contact with each other at the beginning of the project, and whose main concern at that time was not the details of the daily operations.

- Need for adaptability and flexibility.

The functional needs of the various instrument teams are very different. Some SOHO instruments, mainly the coronal imaging instruments, will be operated interactively every day in real-time. Some other instruments (CEPAC, CELIAS, VIRGO, GOLF) will generally operate automatically and will not need real-time operational control except for surveillance of housekeeping data. Consequently, some teams will need to command their instruments and receive the telemetry in real-time, while some other teams will command in the traditional preplanned manner from a remote site and only retrieve telemetry files on a daily or weekly basis.

The instrumenters' requirements on the ECS will also vary during the lifetime of the mission. All the eleven instrument teams will bring in their own equipment for integration into the EOF and most of the teams are planning to be at the EOF during the initial phase of the mission. After the spacecraft is commissioned, only 6 teams are expected to remain located in the EOF while the others teams will return to their home institutions.

The IWSs are supplied by the individual instrument teams and represent a wide range of

hardware and software. The ECS must provide connectivity for each IWS and between IWSs. The ECS must be capable of establishing connections with the SOHO ground e4lements, the analysis facility and with the outside world. The ECS must satisfy stringent performance requirements. It must be able to sustain the real-time telemetry and commanding rates, it must have sufficient storage capacity to archive the science data. Finally, it must be able to support two month per year of continuous science operations.

- Software Reuse.

As much as possible, the ECS design incorporates standard off-the-shelf hardware and software. The main software systems that have been reused in the ECS design are:
(1) the Transportable Payload Operations Control Center (TPOCC), which has been used to support what is referred to as "Global Services" functions such as inter-tasks communications, event generation and event logging. TPOCC also provides an extensive library of routines that have been reused in the ECS implementation.
(2) the Interactive Experimenter Planning System, was used as the basis for the implementation of the ECS science planning and scheduling functions. These functions include batch and interactive scheduling, conflict resolution and automatic scheduling and re-scheduling of activities.

- Rapid prototyping development approach.

Several software prototypes have been developed during the ECS design phase to verify major design choices. In particular the following was evaluated or verified: hardware performance for telemetry distribution, applicability of reused software, and demonstration of operator's interface implementation to the users.

- Adoption of implementation standards.

Representatives of the ECS development team attended all the science operations working group meetings, presented various draft of the interface control document, and were able to help and participate in the choice of a set of standards such as : X-Windows (X 11), Motif, Interactive Data Language (IDL), TCP/IP Ethernet, Flexible Image Transport System formats, Standard Formatted Data Unit, Structured Query Language and Standard U.S. commercial power and receptacles

SOHO EOF IMPLEMENTATION

Facility

The EOF facility is located in Building 14 at the Goddard Space Flight Center. It consists of offices for the project scientist, the science operations coordinator and the various instruments teams. It also includes a large conference room and various equipment such as telephones, fax machine, color printer, scanner, etc. The ECS equipment is located in the science operations coordinator's office. The EOF is also located next to the mission operations center, where the flight operations team will control the day-to-day operations of the spacecraft

Software

The ECS software is comprised of five major subsystems:

(1) The telemetry subsystem receives the real-time telemetry from Pacor and distributes it to the resident instrumenters according to their requests. It also receives files of quicklook telemetry, primarily containing tape recorder dumps from DDF. The telemetry subsystem archives all the SOHO telemetry data for a period of seven days. The archived telemetry is made available in the form of files to the SOHO scientific community.

(2) The commanding subsystem supports the real-time as well as the preplanned commanding functions. It receives the commanding data from all the instrumenters and it provides a single interface to the Command Management System and the Payload Operations Control Center.

(3) The planning and scheduling subsystem provides an automated tool to produce an integrated and conflict-free observation plan. It can merge the individual instrument plans, accept input from the science operations coordinator, incorporate predefined constraints such as DSN schedule and reserved times for spacecraft activities. This subsystem was based on reused software, but it was re-implemented using an object-oriented

methodology as described in more details below.

(4) The user interface subsystem provides a set of windows that will enable the science operations coordinator to monitor and control activities within the EOF.

(5) The "global services" subsystem supports functions such as inter-task communication and event logging. It was implemented in large part by reusing the existing TPOCC software.

Other ECS subsystems support E-mail functions, time services, system administration functions and data base functions. These subsystems were implemented making extensive use of off-the shelf products.

Physical Architecture

The physical architecture of the EOF had to accommodate the diversity in IWS hardware and operational requirements. It also needed to provide efficient communications between secure and public networks while satisfying security requirements. The SOHO EOF physical architecture is illustrated in Figure 1. Its main characteristics are:

(1) Use of high power workstations able to handle the real-time data rates, while allowing the project scientist and the science operations coordinator to monitor the EOF operations through X-windows and use science analysis software such as IDL.

(2) Use of a high performance router which allows to isolate the ECS and its interface to secure networks from the outside world. It also separates the ECS "operational" data traffic from the IWSs and the data traffic associated with science analysis. Based on predicted data volumes for each instrument team, the IWSs were grouped and connected to the ECS router via seven Ethernet "segments" terminated by hubs and converters. This provides a rather low cost standard connection with all the IWSs. The filtering capabilities of the router are also used to implement network-level security.

(3) Full redundancy: All elements are redundant and data storage is done on a Redundant Array of Inexpensive Disks (RAID).

APPLICABILITY OF THE EOF TO OTHER MISSIONS

In many aspects, the SOHO EOF had to be customized to the specific requirements of the mission such as restrictive interfaces with other ground system elements and adherence to the pre-existing protocol used in the spacecraft test and integration environment. However, the EOF contains several "building blocks" that are applicable to other missions. In particular, the planning and scheduling subsystem was designed and implemented with reuse in mind. A more detailed description of this subsystem follows.

Planning and Scheduling Subsystem

The SOHO EOF required a scheduling system to find and resolve conflicts between the individual schedules from each of the satellite's eleven instruments. It had been proposed to reuse an existing scheduling system to support these functions. The EOF new system needed to be flexible, fast and have the capability to merge pre-existing individual schedules. Also, it was to be supplied to several users: the flight operations team within the Command Management System and the instrument teams that wish to use it for planning their own observation sequences. This implied new rules, broad kinds of strategies and activities that the existing system could not support without extensive modifications. This presented the opportunity to re-design and re-implement the scheduling system using object-oriented methods, making it easier to customize and port to different environments: the Planning And Resource Reasoning (PARR) system was developed using an object-oriented design and was implemented in C++.

PARR works as an intelligent tactical planning tool to put specific activities on a timeline by following the strategies and checking constraints found in its knowledge base. PARR's knowledge base consists of a list of strategies used to schedule activities with specific times and durations. One particularity of PARR is that it uses a combination of

conflict avoidance and conflict resolution rules; this limits the number of searches required to build a timeline and accelerates the process of building a conflict-free schedule.

The C++ implementation and the use of classes allows to represent abstractions of scheduling objects, such as activities, strategies, resources and constraints Resources include both data that PARR cannot change and data that changes as a result of the schedule it is creating. Activity classes represent types of activity that can be scheduled. Constraints represent PARR's conflict avoidance rules: a constraint can state how an activity must be scheduled in relationship to other activities or resources. Strategies represent PARR's conflict resolution rules: they are used to place activities on the schedule, and to move activities when the constraint checking process discovers conflicts. PARR also uses several paradigms, enabling it to control which activity classes are to be scheduled, the order in which they are scheduled and the merging of schedules created outside of PARR.

Another feature of PARR is that its user interface code has been separated from the algorithmic code, making it easier to adapt to other applications where the user interface is usually the functionality that needs most to be customized to respond a special requirements.

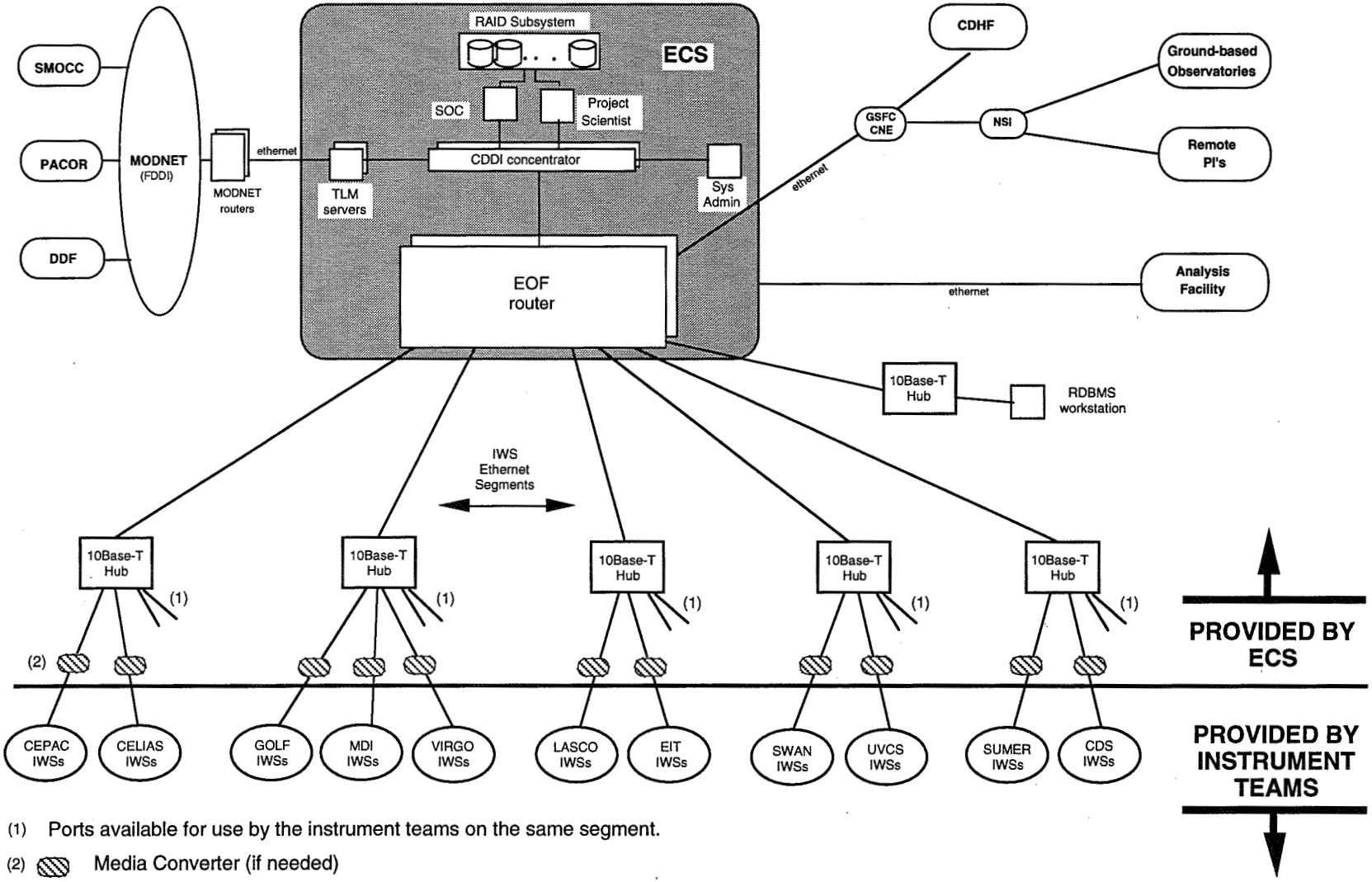
In conclusion, PARR has been designed to facilitate its portability. The object-oriented nature of PARR and its paradigm constructs make it easy to customize for new planning and scheduling applications: for each new PARR application, the classes of generic objects for resource classes, constraints, and strategies can be supplemented with application-specific types.

CONCLUSION

Overall the development of the EOF has been a success. Costs have been kept under proposed budget. All the initial requirements defined by the scientists have been satisfied, and a few additional capabilities have been implemented without increased funding. The timeliness of the EOF development was highly beneficial to the other SOHO ground system elements: it has provided them with early and precise information concerning the interface with the instrumenters. This aided in reducing risks to the SOHO Project. The basic EOF design is applicable to those missions that requires near-real-time commanding, real-time telemetry distribution, and close communications with the flight operations team. Having the facilities co-located allows cost savings in development, facility operations and maintenance.
The physical architecture of the EOF allows for great flexibility, allowing instrument teams to modify or upgrade their software with minimal impact. It has allowed to implement a sufficient security level while allowing easy communications with the outside world: this is a basic requirement for the scientific success of the mission. Finally its modular software architecture makes the ECS a good candidate for applicability to other missions

ACKNOWLEDGMENTS

The authors would like to thank Dorothy Perkins, Code 510, Paul Ondrus, Code 510.1, Patricia Lightfoot, Code 514, William Worrall, Code 407, AlliedSignal Technical Services Corporation, and the SOHO Project for their support of this task. This work was supported by NASA Contract NAS5-27772.



(1) Ports available for use by the instrument teams on the same segment.

(2) Media Converter (if needed)

Figure 1. EOF architecture

111159 N95-17574

354251
P.6

Galileo Spacecraft Modeling for Orbital Operations

Bruce A. McLaughlin and Erik N. Nilsen
Jet Propulsion Laboratory
California Institute of Technology

ABSTRACT

The Galileo Jupiter orbital mission using the Low Gain Antenna (LGA) requires a higher degree of spacecraft state knowledge than was originally anticipated. Key elements of the revised design include onboard buffering of science and engineering data and extensive processing of data prior to downlink. In order to prevent loss of data resulting from overflow of the buffers and to allow efficient use of the spacecraft resources, ground based models of the spacecraft processes will be implemented. These models will be integral tools in the development of satellite encounter sequences and the cruise/playback sequences where recorded data is retrieved.

Key Words: Aerospace, mission operations, sequence planning, spacecraft modeling

1.0 THE GALILEO PHASE II DESIGN

The Galileo Phase II redesign for Jovian orbital operations using the Low Gain Antenna (LGA) is driven by the need to match the high data acquisition rates with the low spacecraft data transmission capability. Many changes have been made to both the spacecraft and the ground data systems to optimize the effective data transmission rate. Spacecraft changes include extensive redesign of the Command and Data Subsystem (CDS) flight software, modifications to the Attitude and Articulation Control System (AACS) software and selected instrument flight software changes. Ground modifications include adding noise reduction equipment at selected DSN sites, intrasite and intersite antenna arraying capability, new receivers and signal acquisition equipment and extensive ground software changes to support new data transmission modes.

The changes to the flight system are numerous and constitute a significant redesign of the flight software. The primary modification to accommodate the low data rates was the switch from Time Division Multiplexed (TDM) telemetry modes to a packetized telemetry system based upon a highly optimized CCSDS packet definition. This allows a flexible, prioritized data transmission system, eliminating the inherent data redundancy of the TDM design.

Onboard data buffering is implemented to allow high rate data acquisition. Central to this design is the Data Memory System (DMS - tape recorder) which will hold 900 Mbits of data. This will be used to store high rate data (remote sensing and fields and particles science data) acquired during satellite encounters and relayed to the ground during the orbital cruise phase between encounters. For onboard data manipulation and real time data acquisition and storage, several buffers are implemented in solid state memory. The most important are the priority buffer, which holds priority engineering and Optical Navigation (OPNAV) data, and the multi-use buffer, which is used for the storage and manipulation of Real Time Science (RTS) and the playback of data from the DMS.

Extensive data editing and compression is implemented to reduce the number of bits transmitted to the ground. The CDS can select or deselect data sources based upon mission phase and can edit many of the data sources. Both lossy and lossless compression schemes have been implemented onboard. Lossy compression based upon the Integer Cosine Transform (ICT) algorithm has been implemented in the AACS software and is used to compress images and Plasma instrument (PWS) data sets. Compression ratios of 2:1 to 80:1 can be selected. Lossless compression using the Rice algorithms (Reference 1) has been implemented for selected science data sets, resulting in data compression ratios of 1.2:1 to 5:1.

2.0 SPACECRAFT DATA FLOW

Figure 1 illustrates the typical data flow within the flight system. As illustrated, the onboard data buffers form key elements of the design. Controlling the data input to the buffers and the data output to the downlink are key tasks for the flight sequences. If the aggregate data input rate exceeds the data transmission rate, the buffers will fill. Overfilling the buffers will result in discarding new data. However, if data acquisition is controlled such that the buffers empty, fill data is inserted on the downlink, lowering downlink efficiency. Maintaining the delicate balance of the buffer fill state will be a significant challenge for Phase II operations.

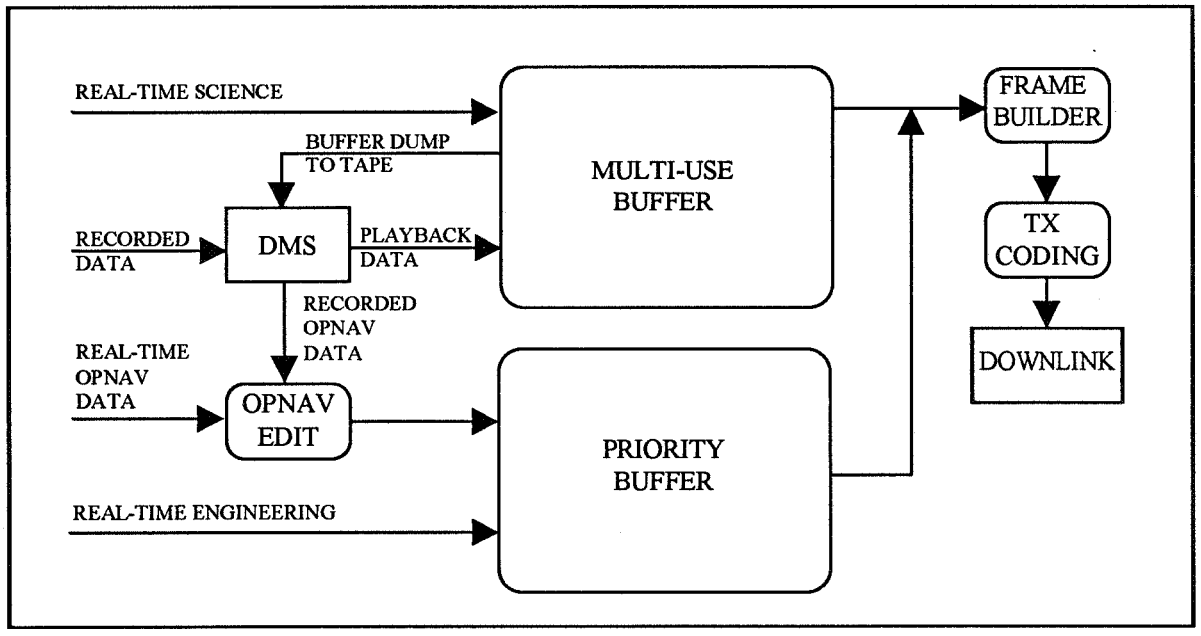


Figure 1 - Spacecraft Data Flow

The data input process has three constituent parts: the real time engineering (RTE) and OPNAV data, which is placed in the priority buffer, Real Time Science data (RTS) and the Playback data which is processed through the multi-use buffer. Real time data (RTE and RTS) is taken continuously and is controlled by the CDS. Data sources can be selected and deselected in accordance with planned observations and the data collection mode is controlled by the CDS telemetry command. Real time data acquisition and the downlink telemetry rate are controlled using the same command. This links the Real time data collection with the downlink telemetry rate via one of 90 selectable modes.

The OPNAV and Playback processes are independent of the real time data acquisition process, and are intermittent activities. OPNAV activities occur prior to encounters, and place certain restrictions on both the multi-use buffer (where the data is processed) and the priority buffer (where the data is stored for transmission). The Playback process for retrieving the recorded data is completely new for orbital operations. In playback, the CDS performs autonomous retrieval and processing of the data from the DMS, controlled by a special parameter set called the playback tables. These tables contain information on the format of the recorded data, lists the data to be retrieved and the editing and compression to be performed on the selected data. Playback data is placed in the multi-use buffer for processing. To control the filling of the multi-use buffer a set of buffer pointers have been implemented. When playback is active and the buffer fill state falls below the low watermark (i.e. the downlink rate exceeds the data acquisition rate, allowing the buffer to empty) the CDS will autonomously control the DMS to replay data into the multi-use buffer. When the buffer fill state exceeds the high watermark, the CDS commands the DMS to cease operation and processes the raw tape data into completed data packets. This process occurs simultaneously with real time data acquisition and is exclusive of all other record activities.

2.1 BUFFER MODELING

The buffer modeling task is necessary for the system to work. The highly interactive nature of the system and the statistical nature of the data compression algorithms necessitates an iterative approach to the design of spacecraft command sequences for orbital operations. With the number of independent variables that must be factored, and the accuracy with which they can be predicted, precise control of the buffer states will be difficult. Without ground based system models, the flight system could not be operated efficiently.

To control the buffer fill rate, many variables need to be controlled. On the output side, the commanded downlink data rate is varied in discrete steps over the course of a DSN track to closely match the data rate capability (Figure 2). These data rate changes must be predicted well in advance and scheduled in the sequence. Any change to equipment capabilities or link performance will affect the data rate capability and the output from the buffers.

On the buffer input, the various data sources must be controlled and the rates at which each source generates data must be predicted. This includes modeling which instruments are selected and deselected, the data editing algorithms and the target compression ratios. Each of these factors vary as a function of time. In addition, the compressibility of some of the sources is very data dependent, thus

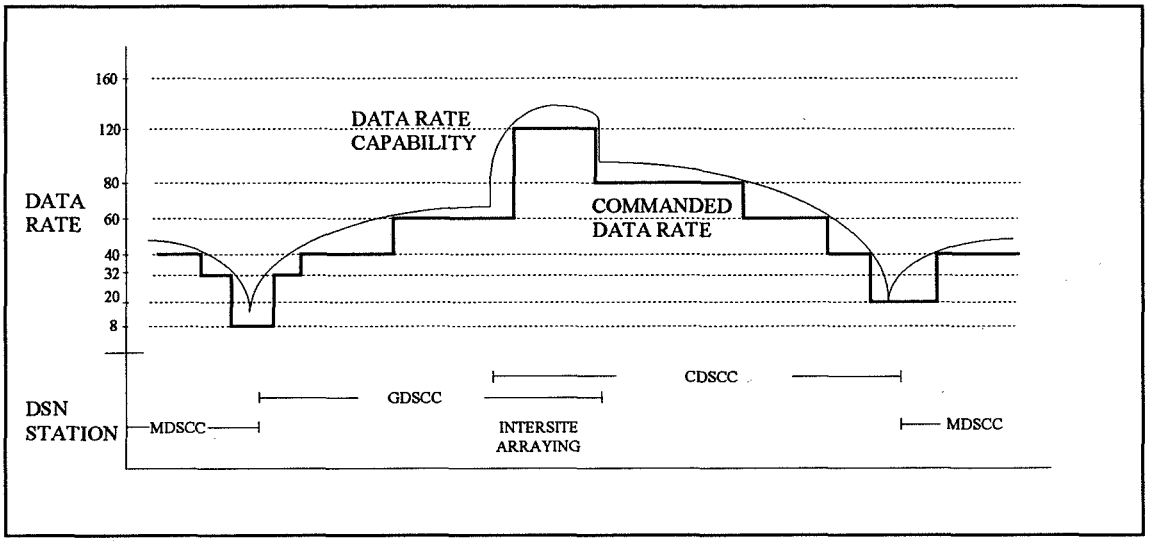


Figure 2 - Downlink Telemetry Rate Change Modeling

considerable variability in data volume is expected.

The priority buffer has only two input data sources; the real time engineering data and OPNAV data. The only significant restrictions on buffer state for the priority buffer is the requirement that the buffer be empty before initiating an OPNAV process. The current priority scheme essentially guarantees that if the downlink rate exceeds the engineering acquisition rate, this state is achieved.

The multi-use buffer requires significant modeling to predict its state. The modeling can essentially be broken down into two separate modeling regimes: the encounter phase, characterized by low downlink data rates and high rate RTS data acquisition with interspersed buffer dumps to tape, and the playback phase with higher downlink rates and with the Playback process active.

The bulk of the scientific data is gathered during satellite encounters. This includes the remote sensing and very high rate fields and particles instrument data which is recorded directly to tape at up to 806.4 Kbps, and the high rate RTS data, which is processed into the multi-use buffer. Typically, the desired RTS acquisition rate well exceeds the downlink rate, filling the buffer. To allow extended high rate data acquisition without overflowing the multi-use buffer the Buffer Dump to Tape function has been implemented. This is a sequence controlled activity wherein the CDS will transfer completed Virtual Channel Data Units (VCDUs - Memory Management Units) from the multi-use buffer to the DMS, freeing the buffer for continued data acquisition. Since buffer Dump to Tape is a sequence controlled activity, it can be scheduled to occur between other DMS activities. Buffer management during encounters consists of predicting RTS data acquisition rates and scheduling buffer dumps to tape when necessary to prevent buffer overflow and loss of data.

During the orbital cruise phase, data is retrieved from the DMS via the Playback process. Typically, the downlink data rate is higher than during encounters and the continuous RTS data acquisition is set to a

lower rate. This allows the playback process to transfer data from the DMS into the multi-use buffer, process the data and prepare it for downlink. Since the replay of data from the DMS is controlled via the buffer high and low watermarks, the process is self-regulating. The modeling task for cruise consists of multiple parts: insuring that the high and low watermarks are properly set, insuring the RTS data acquisition is low enough to prevent data loss due to buffer overflow and modeling playback data editing and compression to recover all of the significant encounter data.

2.2 MODELING TOOLS

The Phase II ground system has two main tools for predicting and controlling the data flow on the spacecraft. They are: SEQGEN, the primary sequence generation tool of the Mission Sequence System (MSS) and MIRAGE, a newly developed tool for processing data rate predicts and producing buffer models. Supporting the generation of sequences and the modeling effort are a suite of tools to automate the process. New tools for Phase II are TLMGEN, which provides automated generation of spacecraft telemetry rate change commands based upon predicted capability and the Playback Table Editor which generates playback table entries based upon the DMS tape map and models playback data production based upon processing parameters selected. These tools, along with a host of existing science and mission design tools, provide data input into the modeling process and are used for optimizing data flow.

2.2.1 MIRAGE

The MIRAGE (Mission Integration, Real time Analysis and Graphical Editor) modeling tool is based upon an earlier multi-mission sequence planning tool developed by the Sequence Automation research group at JPL. Plan-It-II was developed on an UNIX platform using LISP, and specifically developed to be extensible for multiple missions. Plan-It-II provided the capability to simulate functionally the operations of a spacecraft, allow sequences to be staged through the model, and rapidly and interactively present the impacts of the sequence and any proposed changes on the spacecraft resources. The Galileo project adapted the core of Plan-It-II to model the Phase II design. Modifications include incorporating Galileo specific time standards and the Phase II functional design into the model, defining new input data types, providing new constraint checking algorithms and modifications to the user interface to more closely resemble familiar planning tools currently in use.

Mirage provides an interactive environment, displaying on-screen timelines of sequence activities and accompanying graphs showing the states of the spacecraft resources. It also provides an interface to the details of the science planning requests and allows the user to add, delete and modify these activities. This interaction allows the user to explore different approaches to a situation, varying parameters and displaying the results. This results in the rapid development of a viable sequence of data collection activities which the spacecraft can accommodate.

MIRAGE will be used early in the science sequence design process to analyze the effect of the science observations on spacecraft resources. Used primarily in the Orbit Activity Plan (OAP) level, it will determine if a planned set of Real-Time and recorded observations generate buffer overflow conditions, monitor the usage of the DMS and track the allocation of resources to the various science observations.

MIRAGE will also play an important role during sequence execution. Because of the uncertainties involved in the telecommunications link modeling and onboard data compression, the actual data flow may not proceed as predicted. Sequence tweaking, involving modification of one or more data acquisition or transmission parameters, will need to be modeled to determine the overall effect on the data flow. Integral to this process will be the MIRAGE analysis of the spacecraft resources.

2.2.2 SEQGEN

SEQGEN is an existing sequence development tool which takes the OAP level inputs and converts the activities into command sequences and playback parameter tables. SEQGEN is responsible for enforcing many of the sequencing rules and constraints checking for certain onboard data resources and downlink data transmission. For the Phase II mission, SEQGEN was modified to generate the playback table entries. These parameters, which are independent from the spacecraft sequence, instruct the CDS on how the recorded data will be processed. Integral to the generation of the Playback Table entries, the Playback Table Editor allows modification of the playback parameters, adjusting data selection, editing and compression for the recorded instrument data.

The output from SEQGEN can be routed to MIRAGE for modeling. This allows an iterative approach to the sequence generation process. In the early sequence design stages, an activity plan is produced and checked by MIRAGE for proper data flow. This product is refined into a working spacecraft sequence, again using MIRAGE modeling and the Playback Table Editor to adjust playback data parameters. Once the sequence is executing, sequence tweaks to optimize the data flow will be verified using MIRAGE before being sent to the spacecraft.

3.0 SUMMARY

This paper has presented the ground based modeling of the spacecraft processes for the orbital operations mission using the Galileo Low Gain Antenna. The redesign of the flight software requires a higher degree of spacecraft state knowledge than was originally anticipated. In order to optimize the data flow onboard the spacecraft and to the ground, interactive modeling of the data acquisition, buffering and transmission is required during the sequence design process and during sequence execution. These models have been developed concurrently with the flight software design, taking advantage of existing ground software where applicable and developing or adapting software for specific modeling and sequence generation functions.

4.0 ACKNOWLEDGMENTS

The work described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

The authors wish to acknowledge the members of the Galileo Systems Development Office and the MIRAGE and SEQGEN development teams for their input and review.

5.0 REFERENCES

1. Rice, R. F., (November 1991) *"Some Practical Universal Noiseless Coding Techniques, Part III, Module PSI 14, K+"*, JPL Publication 91-3

111160 N95-17575
354252 p 10

THE ADVANCED ORBITING SYSTEMS TESTBED PROGRAM: RESULTS TO DATE

John F. Otranto and Penny A. Newsome

CTA INCORPORATED
Rockville, Maryland USA

ABSTRACT

The Consultative Committee for Space Data Systems (CCSDS) Recommendations for Packet Telemetry (PT) and Advanced Orbiting Systems (AOS) propose standard solutions to data handling problems common to many types of space missions. The Recommendations address only space/ground and space/space data handling systems. Goddard Space Flight Center's (GSFC's) AOS Testbed (AOST) Program was initiated to better understand the Recommendations and their impact on real-world systems, and to examine the extended domain of ground/ground data handling systems. The results and products of the Program will reduce the uncertainties associated with the development of operational space and ground systems that implement the Recommendations.

1. INTRODUCTION

GSFC's AOST Program continues to provide a bridge between the development and widespread use of the CCSDS Recommendations. AOST Program activities include developing and using a Testbed, developing flight-qualifiable components, conducting a test program, performing studies, and actively disseminating the knowledge gained. This paper presents an overview of the Capability Two (C-2) AOST and the results and lessons learned through AOST Program activities to date (July 1994), including architectural issues, the proposed standardized test suite, and flight-qualifiable components. This paper also summarizes the correlation between the AOST and the Code 500 Renaissance effort, and AOST future activities, including implementation of the Space Communications Protocol Standards (SCPS) and the Mission Operations Control Architecture (MOCA).

2. AOST PROGRAM OVERVIEW

An overview of the C-2 AOST is presented in Figure 2.1-1.

2.1 FLIGHT SYSTEM ELEMENTS

The C-2 AOST flight system elements include an Instrument Simulator (IS), a Video Digitizer/ Packetizer/Multiplexer (VDPM) and a Wideband Transfer Frame Formatter (WTFF). These elements have been developed by the GSFC Instrument Electronic Systems Branch (Code 738).

The Instrument Simulator creates simulated spacecraft instrument data. The IS is capable of simulating data for one to six instruments and uses the CCSDS Version-1 Packet format (Reference 1). The data generated by the IS are input to the WTFF via Fiber Optic Transmitter/ Receiver Interfaces (FOXI).

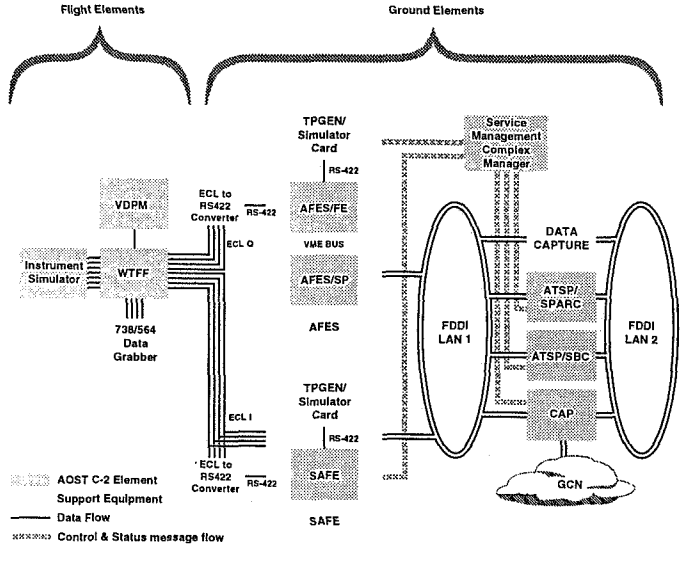


Figure 2.1-1. C-2 AOST Overview

The VDPM generates CCSDS Version-1 Packets and optionally multiplexes them into Multiplexing Protocol Data Units (M_PDUs) (Reference 2). The Packet data field contains either video data that have been converted to digital form by the VDPM or octet-aligned digital data input to the VDPM via a FOXI interface. The VDPM represents a standard interface for CCSDS Path Packet Service. The M_PDUs or Version-1 Packets are input as a single data stream to the WTFF using one of seven available telemetry user data input interfaces.

The C-2 WTFF is designed to serve as a gateway providing transfer frame generation using PT and AOS services for up to seven user virtual channels (VCs). Data arriving from any of the seven user data input interfaces are buffered and inserted into Version 1 Transfer Frames (V1TFs) or Virtual Channel Data Units (VCDUs). WTFF processing of the data consists of: Reed-Solomon (R-S) header, R-S frame, and bit transition density encoding; multiplexing of the frames into a single physical data stream; and appending of a frame synchronization marker to each frame. The WTFF also provides the interface to the ground system elements. The WTFF can selectably output data on one or two physical output channels.

2.2 GROUND SYSTEM ELEMENTS

AOST ground system elements include two Front Ends (FEs), three types of Service Processors (SPs), a Communications Address Processor (CAP) and service management elements. The CAP provides a connection to a ground communications network (GCN). A network and service management system controls, configures, and monitors the AOST ground system elements.

The Microelectronics Systems Branch (Code 521) is developing the Advanced Front End System (AFES), which provides a multiprocessing environment based on a VMEbus open architecture. The AFES uses cards based on custom Very Large Scale Integration (VLSI) controllers to achieve a low cost, high speed, and highly reliable implementation. Each custom card has a 32-bit microprocessor. AFES components are being developed for generic applications and are being used in other systems such as the Small Explorer (SMEX). Commercial off-the-shelf (COTS) cards such as Ethernet and Fiber Distributed Data Interface (FDDI) cards and disk modules are used wherever possible.

The AFES provides return link processing services, configuration management services, and testing and verification services. The AFES comprises a front end (AFES/FE) and one of the SPs, the AFES/SP, housed in a single VME enclosure. Frame synchronization, bit transition density decoding, R-S header and frame error detection and correction, and VC sorting are provided by the AFES/FE. The AFES/FE outputs data formatted as Space Operations Service Data Units (SOSDUs) (Reference 3), a data unit defined by the AOST Program. The SOSDU provides a mechanism for ground transportation and identification of data types consistent with the CCSDS Recommendations. The AFES/FE transfers SOSDUs to the SPs or to the CAP for routing to their user destination(s). The interface to the AFES/SP is internal to the AFES; the data across this interface are formatted as VC Frame SOSDUs. The interface to the ATSPs and to the CAP is accomplished using a FDDI LAN.

The AFES/SP is an integral part of the AFES. The AFES/SP performs CCSDS PT and AOS processing on the SOSDUs received from the AFES/FE, creating Virtual Channel Access (VCA), Bitstream, or Path Packet SOSDUs. The resulting SOSDUs are transferred to the AFES FDDI network interface function for transmission to a predetermined destination address.

Code 521 has also developed a Stand-Alone Front End (SAFE) that is identical to the AFES/FE; these redundant FE systems enable the AOST to process two simultaneous data streams from the WTFF. The SAFE also uses FDDI interfaces to transfer data to the ATSPs and to the CAP.

The Data Systems Engineering Branch (Code 564) has developed two ATSP implementations using solely COTS hardware and operating systems. One implementation is using a "Single Board

Computer" (ATSP/SBC) and a real-time operating system (VxWorks). The second implementation is using a SPARC workstation (ATSP/SPARC) and a UNIX-based operating system. Both the ATSP/SBC and the ATSP/SPARC use Reduced Instruction Set Computer (RISC) technology. The input, output, and management interfaces are identical for both ATSP implementations. The ATSPs receive SOSDUs from the AFES/FE or the SAFE via the FDDI LAN and process these SOSDUs, providing VCA, Bitstream, or Path Packet service processing consistent with Reference 2. The goals of the ATSP developments are to take advantage of and evaluate the potential of the latest technological advancements that industry has produced.

The CAP provides a gateway function for the AFES, SAFE, and ATSPs, translating the global CCSDS identifiers contained in the SOSDU Header to the appropriate user destination address(es), and providing protocol translation between the AOS Testbed and the GCN. The CAP was developed by the NASA Communications Division's Advanced Development Branch (Code 541.3).

The GCN provides communications interfaces to systems external to the Testbed.

The Service Management (SM) system developed for the AOST allows users of a CCSDS service-providing network to interact with that network in terms of services rather than equipment configurations. The service management system manages equipment configuration information, generates periodic reports about the quality of services, and monitors ground system elements for fault isolation. The AOST SM function provides fault detection, isolation, and recovery capabilities for CCSDS data services. The MITRE Corporation is developing the Network and Service Management elements.

The current management hierarchy for SM comprises a Complex Manager that manages the AFES, the SAFE, the ATSP/SBC, the ATSP/SPARC, and the CAP. Each managed system comprises two conceptual components: the data processor, which performs the actual processing and presents the agent with a local representation of managed parameters; and the agent, which translates the management information from the local representation to a global representation understood by the Complex Manager. The agent presents the Complex Manager with a view of the processor as a collection of abstract functions and system operation parameters. The CAP has an agent that is integral to the CAP development; the AFES, ATSP/SBC, and ATSP/SPARC have SM proxy agents.

Proxy agents are separate modules that reside on the Complex Manager workstation, and are not integral to the development of the associated ground element. These agents translate the Management Information Base (MIB) parameters received from the Complex Manager into configuration setup tables that are then transmitted to the appropriate AOST elements.

The AOST has developed and tested a demonstration version of a standard MIB for CCSDS services and protocols (Reference 4). The MIB allows the Complex Manager and managed systems to exchange management information using a common, standard language. The Simple Network Management Protocol (SNMP) was chosen since it is a well-supported standard protocol for managing network elements and allows the use of public-domain and COTS software for the implementation of the agents and Complex Manager.

3. ACTIVITIES TO DATE

The C-2 AOST development effort is complete, and the testing program for C-2 is nearly complete.

The C-2 AOST provided five CCSDS AOS services: VCA service, VC Frame service, Path Packet service, Bitstream service, and Insert service. The C-2 AOST also support a non-CCSDS service called Space Link Channel (SLC) service. The C-2 AOST supports both conventional CCSDS data units, V1TFs, and AOS CCSDS data units, VCDUs.

A completely new test program was implemented for the C-2 AOST (Reference 5). The test program implements the Master Test Suite (Reference 6) to provide a system-independent series of tests that can be implemented to verify any systems' compliance with the CCSDS Recommendations.

Functional testing associated with the C-2 AOST is completed, and research and performance testing is in progress. The C-2 program has produced a number of results, some related to implementation issues associated with the development of AOST elements, and others related to the CCSDS Recommendations themselves. A selection of these results are presented in Section 4.

Flight-qualifiable components have been produced from the equipment in the AOST and additional flight-qualifiable components are currently being manufactured.

A library of AOST Program and related documents continues to serve as a central repository of knowledge gained and products for the AOST Program. A second AOST Workshop has been scheduled for November 1994 to disseminate AOST results.

4. RESULTS

4.1 ARCHITECTURE

4.1.1 Service & Network Management

AOST SM has greatly facilitated the operation of the AOST ground elements, and has streamlined the testing and analysis process within the Testbed. SM controls, configures, and monitors all Testbed ground elements from a single workstation, providing a focus for AOST ground system activity. The SM graphical user interface allows the operator to highlight AOST elements, select predefined configurations, create new configurations, and download these configurations to appropriate AOST element(s). SM is also able monitor the results of data processing by obtaining status information from each element either on demand or on a periodic basis. These status data can be displayed in real-time either numerically or in graphical format; periodic data can be graphed over time to monitor data processing history.

The AOST SM mitigates sources of error in the comprehensive configuration of the AOST by centralizing and streamlining configuration and control. The service-level specifications manipulated at the Complex Manager workstation concisely define the comprehensive Testbed configuration, and mitigate configuration errors often experienced in the past when local system-level configurations were used. The simultaneous configuration and coordination of several Testbed elements via SM has reduced the number of configuration mismatches, and has allowed for spontaneous development of new scenarios and what-if analyses.

By acquiring and displaying status information from multiple ground elements simultaneously, SM expedites the analysis of AOST data processing activities. The ability to display and graph data from multiple sources in near-real time has been an invaluable tool to developing a comprehensive view of AOST data processing activities. SM also maintains log files that permit more comprehensive post-test analysis.

One of the issues related to the development of SM was the coordination of the proxy agent development with that of the data processors. It was necessary to maintain a constant dialog between the proxy agent developer and the data processor developer during the development process to ensure that the system-level configurations performed by the proxy agent matched the system-level specification within the data processor. On several occasions, small software changes were made to a data processor that required corresponding changes to the configurations being managed by the proxy agent. SM functionality is predicated on a successful communication process to coordinate agent-system interaction.

In the next version of the AOST, the Testbed may develop a Network Management Integration and Coordination workstation that manages a set of Complex Managers. Also, SM may be extended to flight elements, either via a direct link or across a ground-space forward link.

4.1.2 Data Latency

The AOST has been addressing latency issues associated with the AOST elements and the interfaces between the AOST elements. Low data latency is desirable, constant data latency is required, and data loss is unacceptable. The AOST test program is currently attempting to vary the transmission approach across the LANs in the AOST to best meet these three criteria. The AOST is being subjected to a series of performance tests designed to measure and improve data throughput. A FDDI LAN analyzer is being employed to assist in the analysis of the FDDI LAN components and AOST elements.

Data losses have been experienced on the FDDI LANs used to transmit data between AOST ground elements. The AOST test program is currently investigating these data losses, employing strategies to analyze and eliminate these data losses.

The FDDI LAN packet size in use for the AOST is predetermined to be 4136 octets in length, with 4096 octets dedicated to data. A ground rule established for AOST regarding the FDDI LANs and designed to facilitate low data latency was that a single FDDI packet would contain no more than one SOSDU. With the exception of Path Packet SOSDUs (which vary in length in proportion to the packet length) the length of all the SOSDU data types handled by the AOST are significantly shorter than the FDDI LAN packet length. Non-Path Packet SOSDUs use no more than 32% of the FDDI LAN packet capacity.

The AOST will "pack" SOSDUs into FDDI packets in an attempt to increase the effective FDDI LAN utilization. The challenge is to ensure low data latency and data delivery without substantially compromising constant data latency. Once hardware and software modifications are made to effect this change in FDDI LAN utilization, tests will be conducted to measure the resulting data latencies and the data throughput capability.

Future ground systems that transport data consistent with the CCSDS Recommendations should consider data transmission methodologies that better facilitate the rapid transfer of variable-length data units. For example, the use of variable-length FDDI LAN packets that more closely accommodate the varying SOSDU sizes would improve AOST FDDI LAN utilization while maintaining constant and low data latencies. The equipment currently in use in the AOST does not facilitate using variable FDDI LAN packets.

4.1.3 Data Distribution

Equipment designed to support data processing consistent with the CCSDS Recommendations should manage and control each VC data stream separately. As built, AOST ground elements do not always regard each VC as a separate channel, limiting data management and distribution capabilities. Furthermore, the inability to manage each VC separately impacts other ground elements in the Testbed.

The CAP was implemented within the AOST to route data, based on VC, to destinations outside the AOST. The CAP is the only AOST ground element that can route data to multiple output destinations by VC. During the C-2 design phase, a decision was made to route the output of the AOST FEs and SPs to a single destination by Internet Protocol address. The implementation of testing scenarios has been limited by the inability to route data between the AOST ground system elements by VC. For example, scenarios were developed to use separate service processors to process different VCs emanating from a single front end system. It was not possible to selectively route data from a single front end to more than

one service processor. To implement this scenario, the front end system was set to "broadcast" data, that is, send all the data to all the active service processors. While introducing obvious security concerns, data broadcasting forces each service processors to examine each incoming data unit to determine if it should be processed. A service processor not designed to perform this query as the initial data processing step will suffer certain performance degradation as it partially processes data before rejecting it.

A fundamental change in the approach toward data transmission is being instituted in the AOST to enable each AOST front end system to transmit each VC data stream to a separate destination. The extra processing required to route each VC to a specific destination will have some effect on the performance of the front end processors, but should result in better service processing performance. Implementation and testing are necessary to determine the aggregate AOST performance improvement.

4.1.4 Interactive Determination of Grade of Service

In an attempt to achieve a more "data driven" system (Reference 7), the C-2 AOST was prepared to implement the dynamic model given in the "AOS Green Book" (Reference 8), section A.4.1, Option-B, for determining Grade of Service. Figure 4.1-1 presents a flowchart representing the referenced algorithm. The AOST has identified three issues associated with the interactive determination of Grade of Service. The first issue is related to R-S header decoding (Grade 3), the second related to R-S frame decoding (Grade 2) when the data zone is populated with an octet-repetitive data pattern, and the third is related to performing R-S decoding on a VC basis.

4.1.4.1 R-S Header Decoding (Grade 3)

The algorithm illustrated in Figure 4.1-1 initially attempts to perform R-S frame decoding; the presence of the R-S header encoding and CRC fields are not considered in this portion of the algorithm. If the frame fails R-S frame decoding, R-S header decoding is then attempted. There is a 31% chance that a frame that is not R-S header encoded will pass R-S header decoding with two "correctable" errors (see *R-S (10,6) Header Decoding Analysis*, next page). When the frame is falsely identified as being R-S header encoded, the "errors" are corrected changing values in the header. The changed header values can result in misidentification and misrouting of the frame. As the algorithm checks for R-S header encoding only after R-S frame encoding has failed, the actual probability of a frame being altered due to false determination of the presence of R-S header encoding is 0.31(probability of an uncorrectable R-S frame encoded VCDU).

The AOS Green Book, section A.4.1, Option-A specifies a dynamic model for determining Grade of Service in which header decoding is performed first. Using this same analysis, there is a 31% chance that a frame that is not R-S header encoded will pass R-S header decoding with two "correctable" errors when Option-A is implemented.

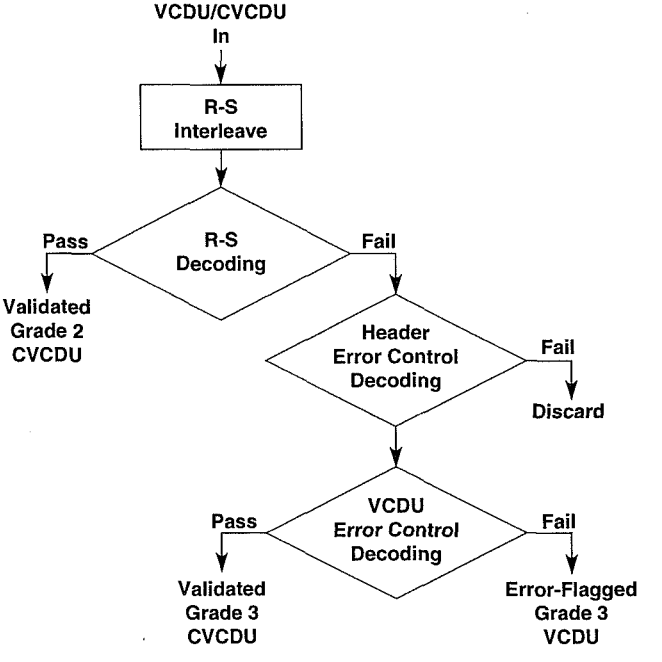


Figure 4.1-1. Option-B Error Control Decoding

4.1.4.2 (255, 223) R-S Frame Decoding of Octet-Repetitive Data Zone Patterns

The data used in the AOST is usually simulated data that contains octet-repetitive data patterns in the data zone portions of the packets and frames, e.g., the data zone of a frame would be populated with "A5A5A5A5A5A5..." (hexadecimal). Tests reveal that a frame containing a high percentage of octet-repetitive data patterns will be decoded and "corrected" when the dynamic model for R-S decoding is applied using the (255,223) R-S code, whether or not the VCDU is R-S frame encoded. An invalid correction can alter header values resulting in misidentification and misrouting of the frame.

R-S (10,6) Header Decoding Analysis

CCSDS R-S encoded headers consist of 3 octets of header data and 2 octets of parity. The code can correct up to 2 errors in the 5 octet pattern. Since parity is derived from the 3 octets of header data, there are $2^{(8)(3)} = 2^{24}$ possible codewords, and $2^{(8)(5)} = 2^{40}$ possible 5 octet patterns. Since parity of the R-S (10,6) code is 2 octets in length, a 5 octet pattern has a $1/2^{(8)(2)} = 2^{-16}$ probability of being a codeword with no error.

The R-S (10,6) code operates on 4 bit "nibbles"; for a 5 octet pattern, there are 10 nibbles. For each nibble, there is 1 correct value and 15 possible incorrect values. The code will correct for any of the 15 possible error patterns in any one of the 10 nibbles. Therefore, there are 15 x 10 = 150 possible single error cases that will be corrected for each codeword.

A 5 octet pattern has a $150 \times 2^{-16} = 0.0023$ probability of being a codeword with one correctable error.

The code will also correct for any of the 15 possible error patterns in any one of the other 9 nibbles for each of the 150 single error cases. Therefore, there are 150 x 9 x 15 = 20,250 double error cases that will be decoded correctly for each codeword. A 5-octet data pattern has a $20{,}250 \times 2^{-16} = 0.309$ probability of being a codeword with two correctable errors.

The source of this invalid correction is the fact that 255 octets containing octet-repetitive data represent a valid R-S codeword. Thus, if any set of 255 octets has 239 or more octets that are repetitive, a R-S frame decoder will "correct" the set to have 255 repetitive octets. A R-S header encoded VCDU with an octet-repetitive data zone will contain only 10 octets not equal to the data zone octets. The following 10 octets will be "corrected" to match the repetitive octet pattern:

- frame primary header - 6 octets
- header parity - 2 octets
- frame CRC - 2 octets

For a VCDU of Interleave 5 containing CCSDS Version 1 Packets, the R-S frame decoder will "correct" the frame first header pointer (2 octets) and the packet primary header (6 octets per packet) for up to 6 packets in the frame data zone, assuming all the packet source data zones share the same octet-repetitive data pattern.

4.1.4.3 R-S Decoding by VC

The AOS Blue Book (Reference 2), paragraph 5.4.9.2.1.5.a states, "The presence or absence of [R-S frame encoding] is an attribute of the Virtual Channel and is pre-specified by management." The system performing R-S decoding and correction must look at the VC to determining whether to perform R-S header or R-S frame decoding. Prior to decoding, the value in the VC field is potentially erroneous and is therefore not a reliable value upon which to base Grade of Service determination.

4.1.4.4 R-S Decoding Conclusions

The AOST has addressed these three issues by prespecifying a Grade of Service for the entire physical channel, and limiting the physical channel to a single Grade of Service. This approach resolves the issues associated with R-S specification per VC, and the potentially erroneous decoding of both R-S headers and R-S frames associated with "on-the-fly" Grade of Service determination. While prespecifying a Grade of Service for a physical channel is a compromise of the Recommendations, it is currently the only reliable alternative presently identified.

4.2 TEST SUITE

The Master Test Suite (MTS) for New AOS Implementations, (Reference 6) has been used to implement the tests for the C-2 AOST Test Plan (Reference 5). The functional test cases used for C-2 testing have a one-to-one correspondence to the tests identified in the MTS, and the structure of each test case is as close to the specifications in the MTS as possible.

There was no single AOST data generator that could implement the entire MTS. A combination of the C-2 AOST flight elements and a data simulation tool developed by Code 521, the Test Pattern Generator (TPGEN), were used to implement portions of the MTS for the C-2 AOST. A portion of the MTS could not be implemented by any test tool available in the AOST; the error patterns identified in some of the MTS test cases could not be created. The next iteration of the AOST may provide a test tool based on TPGEN that provides the full complement of tests in the MTS.

The portion of the MTS that was implemented provided a rigorous and thorough test of the functionality of AOST elements. The AOST data generators did not always provide a sufficient amount of data, however. Some problems with functional production occurred when the systems were tested with large data volumes, for longer periods of time, from a few minutes to 24 hours, and at higher data rates. Systems that successfully passed functional tests with brief test data sets, composed of only a few hundred frames each and processed within 1 to 5 seconds, developed anomalies after processing more continuous data sets and/or data transmitted at a higher data rate. An analysis of the test cases identified in the MTS will be performed to ensure that each test case requires a sufficient amount of data.

The MTS defines tests to be implemented at the system module level. For example, the test case for frame synchronization tests only the part of the system that performs frame synchronization. The MTS approach of testing system modules is analogous to the testing performed by a programmer during integration and development. The C-2 AOST Test Program is analogous to an independent system verification and validation. The C-2 Test Program provided at least one data set to test each function identified in the MTS.

Providing one data set to test each function identified in the MTS created significant redundancy in the complement of tests. For instance, the first test performed, frame synchronization, also succeeded in testing processing for R-S header decoding, CRC decoding, and VC Frame creation. Some streamlining of the MTS is appropriate when testing is performed at the system level. The next iteration of the MTS may provide a streamlined set of test cases for testing performed at the system level.

4.3 FLIGHT-QUALIFIABLE COMPONENTS

Two CCSDS-based and one non-CCSDS flight-qualified components are being developed:

- Reed-Solomon Encoder
- Reed-Solomon Decoder
- Lossless Data Compressor

The flight-qualified R-S Encoder features a selectable interleave depth (1 to 8) and supports a sustained data rate of 200 Mbps. This Encoder is currently available for flight project use, and has been delivered to the Tropical Rainfall Measuring Mission and the X-ray Timing Experiment.

The flight qualifiable R-S Decoder is designed and currently scheduled for production at the NASA Microelectronics Research Center at the University of New Mexico. This chip will perform 1 to 16 symbol error corrections at a sustained data rate of 150 Mbps. The flight qualifiable R-S Decoder will incorporate technology allowing the production of flight-qualifiable components by a commercial foundry.

The flight-qualified lossless data compressor has been developed and manufactured. This compressor chip is available for flight project use, and has been delivered for use on Landsat 7.

4.4 KNOWLEDGE TRANSFER

The knowledge gained through the AOST Program is disseminated to a wide audience that includes flight projects, users, and ground system developers, among others. Workshops provide a forum for the exchange of knowledge between AOST participants and other interested organizations. A library and knowledge database have also been created. An AOST workshop is scheduled for November 1994.

4.5 AOST AND RENAISSANCE

The GSFC Code 500 Renaissance effort is an approach to data systems development designed to improve quality and lower development life cycle cost through the implementation of standards, modularity, and reusable components (building blocks) supporting varying classes of missions and complexity.

Should the Renaissance effort chose to implement a Testbed for the prototyping of building blocks, the AOST architectural approach is a effective model. The concept of well defined functional building blocks on a distributed communications network that supports commercial protocols is central to both the AOST and Renaissance. The redundant front end processors are developed from a set of Code 521 modular components. The front end processors used in the AOST are easily reproducible from both COTS and custom components. Two of the service processors developed in the AOST are also software based; one is developed using the C programming language on a UNIX platform, making it easily transportable to a large number of commercial workstations. The FDDI LAN connecting the AOST ground system elements can incorporate other components developed either within or external to the AOST. The AOST has the potential to easily incorporate and/or test new components.

5. AOST FUTURE PLANS

The next iteration of the AOST, Capability Three (C-3) will incorporate a forward link capability to demonstrate, validate, and verify future implementations of the CCSDS Telecommand (TC) and AOS (forward link) Recommendations. Specifically, the forward link capability will be designed to support the SCPS and MOCA. Implementation of the TC and AOS Recommendations, SCPS, and MOCA will necessarily be incremental, since SCPS depends on the underlying Layer 1 and 2 services provided by the CCSDS Recommendations, and MOCA depends on the upper layer services provided by SCPS. The incorporation of the forward link will require the addition of new ground elements to the AOST, as well as enhancing existing ground and flight elements.

6. SUMMARY

The AOST continues to provide a key source of findings and information related to the implementation of the CCSDS Recommendations. The AOST work will continue through 1995 with a Testbed that supports the AOS and TC forward link command and uplink data generation and processing, SCPS, and MOCA. The AOST remains available to support testing of flight elements and ground system data processors.

7. ACKNOWLEDGMENTS

The authors would like to thank Mr. Michael Bracken, AOST Program Coordinator, and the members of the AOST Program whose dedication to the success of the AOST Program is reflected in this paper. Thanks also to Mr. Charles Fuechsel, NASA Headquarters, for providing the funding for the AOST Program.

8. REFERENCES

1. CCSDS. 1987. *Recommendations for Space Data System Standards. Packet Telemetry.* CCSDS 102.0-B-2, CCSDS Secretariat, NASA, Washington, D.C.

2. CCSDS. 1989. *Recommendations for Space Data Systems Standards. Advanced Orbiting Systems, Networks and Data Links: Architectural Specification.* CCSDS 701.0-B-1, CCSDS Secretariat, NASA, Washington, D.C.

3. March 22, 1993. *Space Operations Service Data Unit (SOSDU) Format Definition Document, Version 3*, CTA INCORPORATED, Rockville, MD.

4. December 1993. *Description of SNMP MIB for CCSDS Space Link Extension Services: AOST Version*, MITRE Corporation, Greenbelt, MD.

5. September 29, 1993. *Advanced Orbiting Systems Testbed (AOST) Test Plan*, Final, GSFC, Greenbelt, MD.

6. February 28, 1994. *Master Test Suite for New AOS Implementations*, CTA INCORPORATED, Rockville, MD.

7. November 16-20, 1992, *Proceedings of the Second International Symposium on Ground Data Systems for Space Mission Operations. "The Advanced Orbiting Systems Testbed Program: Results to Date"*, CTA INCORPORATED, Rockville, MD.

8. CCSDS. 1989. *Report Concerning Space Data System Standards. Advanced Orbiting Systems, Networks and Data Links: Summary of Concept, Rationale and Performance.* CCSDS 700.0-G-2, CCSDS Secretariat, NASA, Washington, D.C.

111161 N95-17576

354255
P 8

NCCDS Performance Model

Eric Richmond
NASA, Goddard Space Flight Center,
Network Control Systems
Branch (Code 532)

Antonio Vallone
Computer Sciences Corporation
10110 Aerospace Road
Lanham-Seabrook, MD 20706

ABSTRACT

The NASA/GSFC Network Control Center (NCC) provides communication services between ground facilities and spacecraft missions in near-earth orbit that use the Space Network. The NCC Data System (NCCDS) provides computational support and is expected to be highly utilized by the service requests needed in the future years. A performance model of the NCCDS has been developed to assess the future workload and possible enhancements. The model computes message volumes from mission request profiles and SN resource levels and generates the loads for NCCDS configurations as a function of operational scenarios and processing activities. The model has been calibrated using the results of benchmarks performed on the operational NCCDS facility and used to assess some future SN service request scenarios.

INTRODUCTION

The NASA/GSFC Network Control Center (NCC) is the operational manager of the Space Network (SN) which provides communication services between ground facilities and spacecraft missions in near-earth orbit. The SN consists of a constellation of Tracking and Data Relay Satellites (TDRSs), TDRSs ground terminals, communication and computing facilities, and operation personnel.

The NCC provides the following functions:

- scheduling user support activities
- disseminating schedules to the users and to the SN support facilities
- controlling the services provided by the other SN elements
- maintaining SN status and configuration information
- disseminating service performance data
- coordinating fault isolation
- generating performance reports.

The NCC functions are supported by the NCC Data System (NCCDS) which is a distributed computer system composed of a Communication and Control Segment (CCS), a Service Planning Segment (SPS), and an Intelligent Terminal Segment (ITS) connected by local area networks. The NCCDS performs the scheduling of the SN resources and processes the messages which the SN users, the NCC, and other SN support facilities use for requesting services, for controlling the SN configuration, and monitoring the SN service performance.

The Network Control Systems Branch (Code 532) is concerned with the effect on the performance characteristics of the NCCDS [1] due to changes in the SN resources (i.e., number of TDRSs and ground terminals) and in the number and complexity of the space missions (e.g. EOS and space station) requesting SN services. The volume of message traffic and the computational effort will

increase. The NCCDS performance can be kept to an optimal level by means of changes to the NCCDS design by increasing the hardware and software capabilities and, possibly, by improving the NCC operational procedures.

A model of the NCCDS has been developed with the objective of providing a tool for assessing the impact on the NCCDS performance of workload changes due to the SN services that will be required by future missions and to the new elements that will be added to the SN in the future. This tool can also be used for evaluating the effect of possible modifications to the NCCDS design and to the NCC operational procedure, and to support the identification of the most cost-effective alternative.

NCCDS CHARACTERISTICS

The NCCDS functions included in the NCCDS performance model are summarized in Figure 1. External messages to and from the NCCDS are exchanged via the Front End LAN (FEL) and the High Speed Message Exchange (HSME) which routes the messages to CCS and SPS functions, tests the communication links, and logs the messages. The inter-segment traffic is supported by the Inter-Segment LAN (ISL).

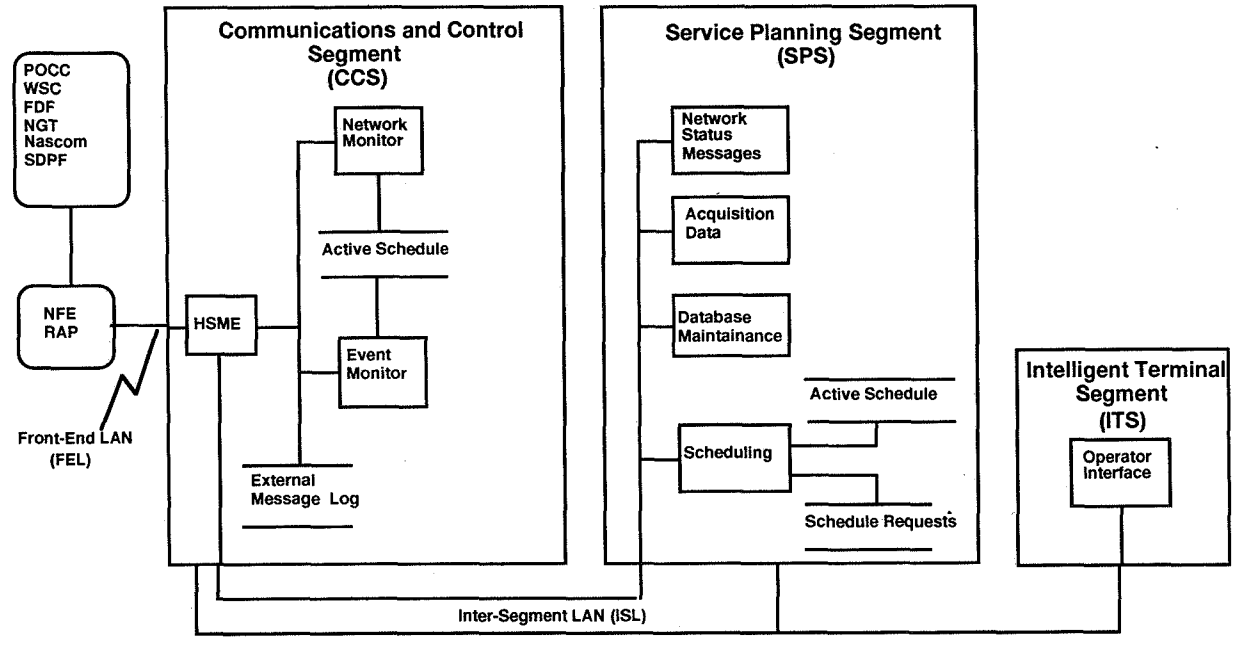


Figure-1 NCCDS Functional Architecture

CCS monitors SN status and performance, sends related data to SPS and operators (ITS) and to the SN users when requested. CCS also monitors SN use and processes users' requests for reconfiguring the space link and the ground communication link. CCS functions are coordinated

with the SN scheduled events stored in a database which is periodically updated from the SPS.

SPS receives from the Flight Dynamic Facility (FDF) acquisition data messages and transfers them to the SN Ground Terminals at White Sands Complex (WSC). SPS also performs the scheduling of users' SN resources requests for future events (forecast schedule) and for changes to the current schedule. It verifies users' requests, generates and maintains SN resource schedules, and disseminates the schedules to WSC, NASA Ground Terminal (NGT), Nascom, Sensor Data Processing Facility (SDPF), and Payload Operation Control Centers (POCCs).

The modeled NCCDS configuration includes the CCS and the SPS computer systems connected by the ISL. Each system is composed by a processing component (CPU), storage peripherals (drives and controllers of disks and tapes) for databases and log files, and the LAN interface components. The model disregards the hardware required for redundancy purposes.

MODEL STRUCTURE

The main requirements [2] for the NCCDS performance model are (1) flexibility for assessing several alternatives of SN users' needs, SN resources, and NCCDS configurations and operational procedures and (2) consistency in comparing results of the assessed alternatives. These requirements are satisfied by a model structure that separately models and integrates the NCCDS performance factors.

Figure 2 illustrates the structure of the model.

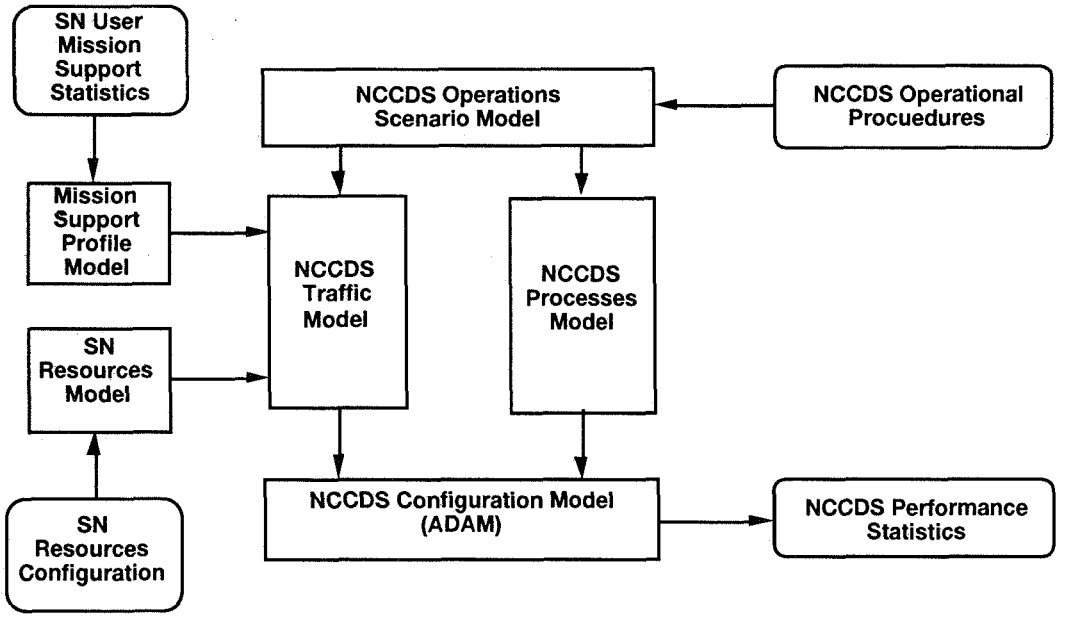


Figure-2 Model Structure

The NCCDS performance model includes the following main models:

- mission support profile model representing users' requests on the SN resources;
- SN resources model representing the number of TDRSs and ground stations;

- NCCDS traffic model representing the volume of messages transferred between the NCCDS and the outside world;
- NCCDS operational scenarios model representing the timing of message distribution and processing;
- NCCDS processes model representing the processes performed on each message and the resulting data transfer between CCS, SPS, and ITS;
- NCCDS configuration model.

The mission support profile model represents the daily average level of support provided to the SN users and is the main driver for NCCDS message traffic volumes and processing loads. Input to this model is the number events (TDRS contacts) and the length of SN resource usage (i.e., K-band Single Access, S-band Single Access, and Multiple Access). The input values may be directly obtained from the Mission Model Database (maintained by GSFC Code 534) or any hypothetical value for "what-if" analysis. Outputs from the model are parameters for the daily load to the NCCDS (i.e., the number of supported events, number of changes to the current schedule, duration of support) and parameters for the forecast scheduling process (i.e., number of requested events per week).

The SN resource model represents the SN configuration (i.e., number of TDRSs and number of antennas per TDRS). It provides values to parameters by which the traffic volume is computed.

The NCCDS traffic model represents the average volume of messages received and transmitted by the NCCDS during a nominal day. The traffic model has been derived from an analysis [3] of message flows covering typical SN request scenarios. The messages are divided in six main groups (Figure 3): schedule related messages, performance related messages, acquisition data messages, configuration related messages, Restricted Access Processor (RAP) monitoring messages and communication test messages, and acknowledgment messages. The grouping is related to the processes performed on the messages.

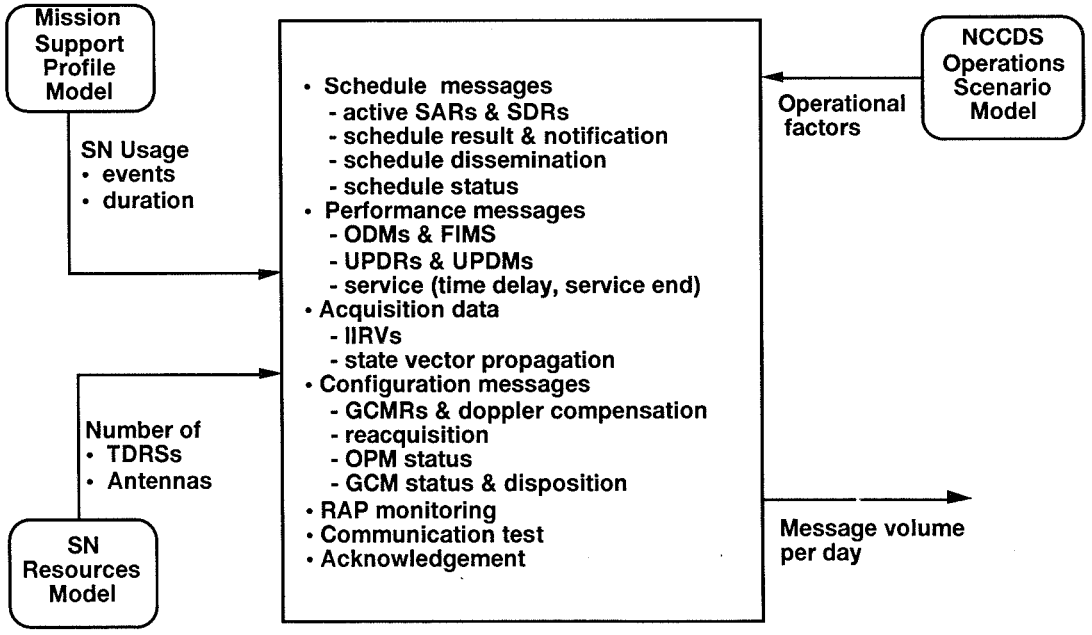


Figure-3 Traffic Model

The model of the NCCDS operations includes two classes of processes (Figure 4): functions initiated by the arrival of a message (i.e., data driven processes), and functions initiated by commands of the operator or NCCDS procedures (i.e., procedure driven processes). The first class of processes is directly driven by the average daily volume of messages computed by the NCCDS traffic model. The operators commands or NCCDS procedures that initiate the second class of processes is represented with a set of operational parameters which indicates the number of initiation per day for each process.

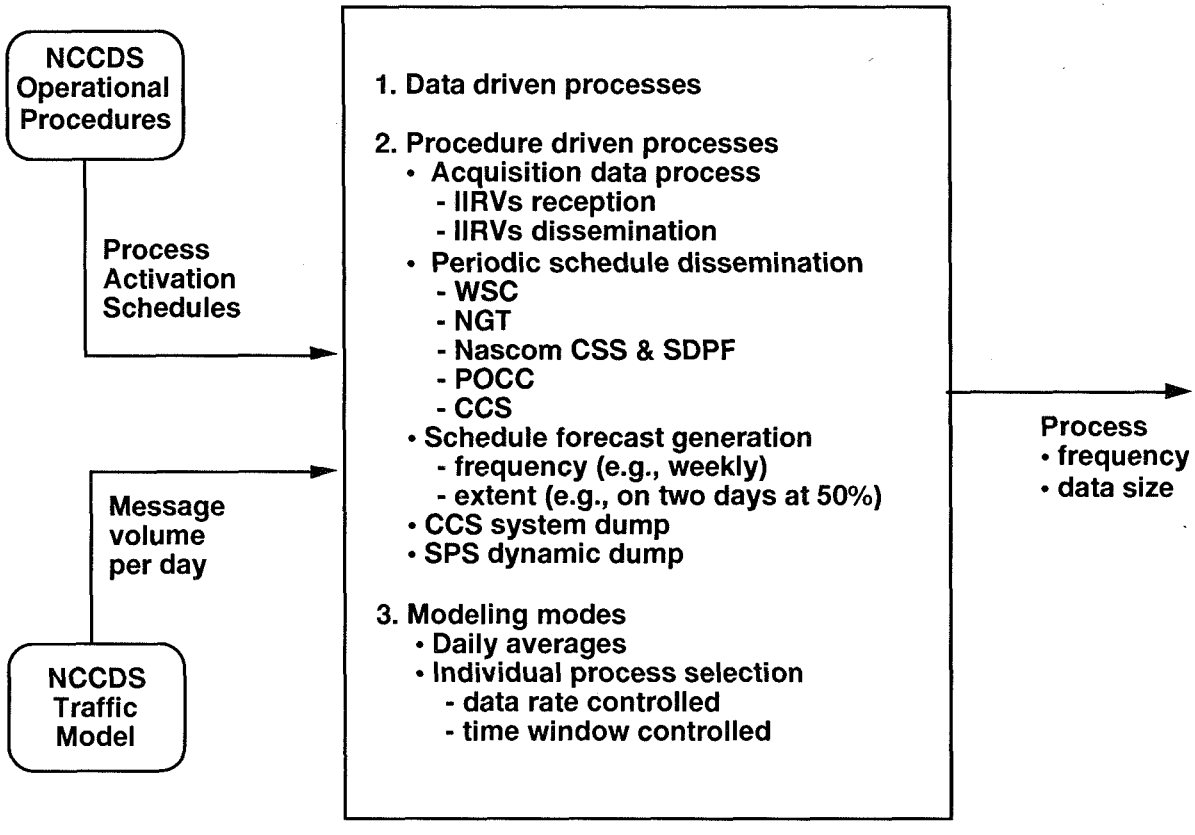


Figure-4 Operational Model

The NCCDS processes model provides the framework for generating the loads of the NCCDS resources from the message volumes computed by the NCCDS traffic model. It represents the actual activities which the NCCDS performs to process each message. Figure 5 is an example of the representation of performance related messages processing.

The NCCDS configuration model represents the NCCDS processing resources (i.e., CPUs and OSs, LANs and protocols, and data storages). It also includes the allocation of the NCCDS process to the resources.

MODEL IMPLEMENTATION

The NCCDS performance model has been implemented by means of two computational packages

running on a PC: a spreadsheet (e.g., LOTUS 1.2.3. or EXCEL) and the Automated Distributed Architecture Modeling tool (ADAM) which is an analytical queuing modeling tool. The reason for splitting the implementation on two packages was to minimize the model development effort and cost.

The spreadsheet component implements the mission support model, the SN resources model, the traffic model, and the operational model. It includes a representation of the process model and generates the parameter values which are input to ADAM.

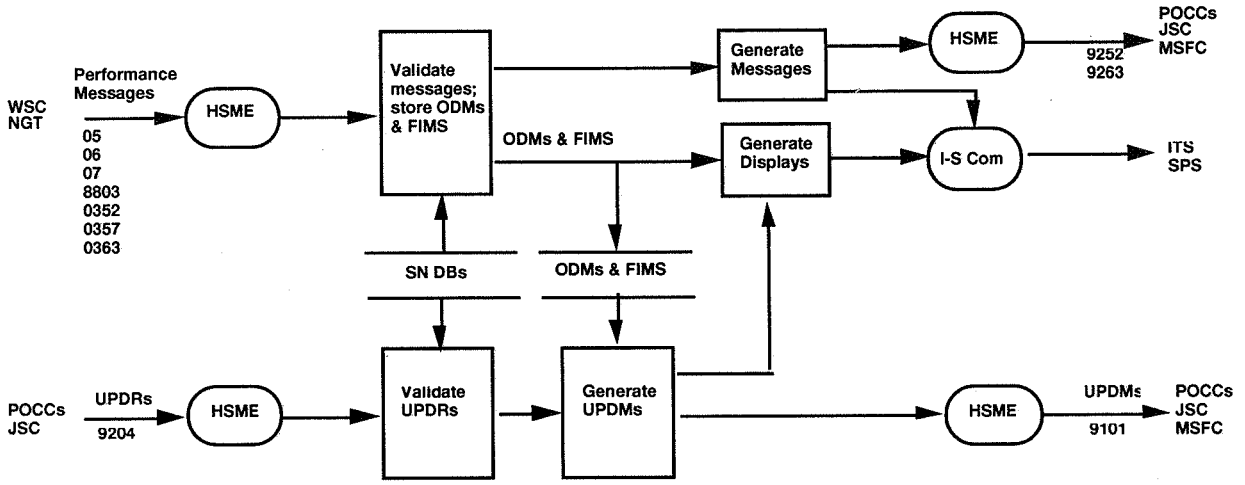


Figure-5 Performance Message Processing

ADAM [4] has been developed by Computer Sciences Corporation for assessing distributed architectures. Figure 6 illustrates ADAM structure.

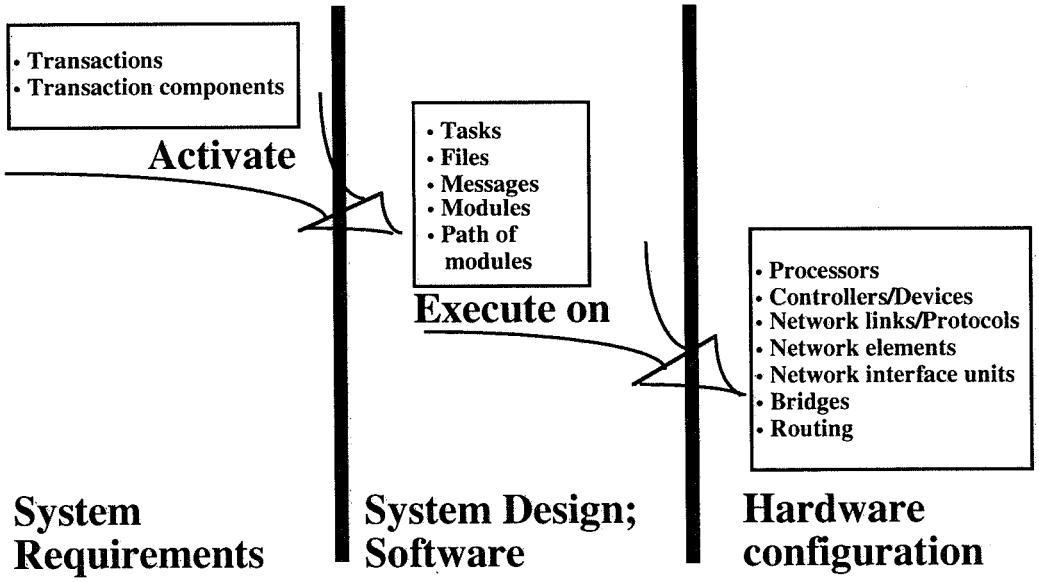


Figure-6 ADAM Structure

ADAM includes a representation of the traffic model (transaction components), of the processes (software components) and of the hardware configuration. The three representations are linked by means of allocation references from which the program computes resources workload and utilization, and service and latency times.

MODEL RESULTS

Input to the model are the characteristics of the missions which use SN services, of the SN resources configuration, and of the NCC operational procedures. The model provides traffic loads, potential bottlenecks, and message service and response times.

To date we have successfully calibrated the baseline NCCDS model representing the current equipment configuration and the data system functions. We have used actual SN resource requests [5] as input to the model and compared the model results with the results of performance monitoring executed on the operational NCCDS facility during the same period of time [6]. The CPU utilization computed by the model was 12% for CCS and 16% for SPS. This compares with monitoring measurements of 14% and 18% respectively.

We have started analysis of future SN resource requests scenarios. Figure 7 shows the CPU utilization of the CCS and the SPS when processing the workload generated by three different hypothetical mission scenarios. The SN resources (two TDRSSs) are used with 1000, 2000, and 3000 TDRSS contacts per week by 10 nominal missions. The scenario assumes a worst case day in which the Space Transportation System (STS) is flying and forecast schedule generation is also performed.

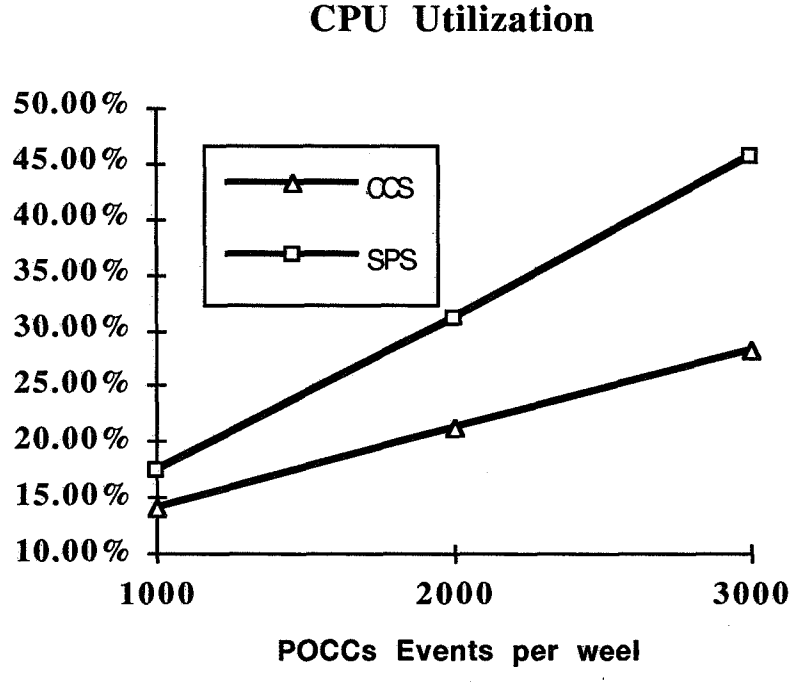


Figure-7 Model Results

The NCCDS performance model will be used for the assessment of the performance characteristics related to various uses of the Space Network services, and alternative configurations of CCS and SPS.

REFERENCES

1. Network Control Center Data System (NCCDS) Detailed Requirements, 530-DRD-NCCDS, November 1993.

2. NCCDS Performance Model - Model Requirements Specification, Contract NAS 5-31500, CSC/TA 33-600-A2, May 1993.

3. NCC Message Traffic Flow Analysis, TR 92014, September 1992.

4. Automated Distributed Architecture Modeling (ADAM). User's Guide, Contract NAS 1-17757, CSC/TM-87/6074, Version 2.0, January 1988.

5. NCC Summary of STDN Operations, December 1993.

6. Operation Data for STS Schedule Generation and Mission Launch: Periods 8/30/93 through 12/3/93.

N95-17577

11162

354256

EVALUATION OF NASA's END-TO-END DATA SYSTEMS USING DSDS+

Christopher Rouff
William Davenport
NASA Goddard Space Flight Center
Code 522.1
Greenbelt, MD 20771
(301) 286-2938
(301) 286-5149

Philip Message
Stanford Telecommunications Inc.
7501 Forbes Boulevard
Seabrook, MD 20706
(301) 464-8900

ABSTRACT

The Data Systems Dynamic Simulator (DSDS+) is a software tool being developed by the authors to evaluate candidate architectures for NASA's end-to-end data systems. Via modeling and simulation, we are able to quickly predict the performance characteristics of each architecture, to evaluate "what-if" scenarios, and to perform sensitivity analyses. As such, we are using modeling and simulation to help NASA select the optimal system configuration, and to quantify the performance characteristics of this system prior to its delivery.

This paper is divided into the following six sections:

I. The role of modeling and simulation in the systems engineering process. In this section, we briefly describe the different types of results obtained by modeling each phase of the systems engineering life cycle, from concept definition through operations and maintenance.

II. Recent applications of DSDS+. In this section, we describe ongoing applications of DSDS+ in support of the Earth Observing System (EOS), and we present some of the simulation results generated of candidate system designs. So far, we have modeled individual EOS subsystems (e.g. the Solid State Recorders used onboard the spacecraft), and we have also developed an integrated model of the EOS end-to-end data processing and data communications systems (from the payloads onboard to the principle investigator facilities on the ground).

III. Overview of DSDS+. In this section, we define what a discrete-event model is, and how it works. The discussion is presented relative to the DSDS+ simulation tool that we have developed, including it's run-time optimization algorithms that enables DSDS+ to execute substantially faster than comparable discrete-event simulation tools.

IV. Summary. In this section, we summarize our findings and "lessons learned" during the development and application of DSDS+ to model NASA's data systems.

V. Further Information.

VI. Acknowledgments.

I. THE ROLE OF MODELING AND SIMULATION IN THE SYSTEMS ENGINEERING PROCESS

As illustrated in Figure 1, modeling and simulation are invaluable tools throughout the systems engineering life cycle, as described in the following paragraphs.

During the concept definition phase, modeling is used to validate the operations concepts, and to derive preliminary estimates of system requirements. For example, an operations scenario for EOS entails recording of payload data generated onboard the

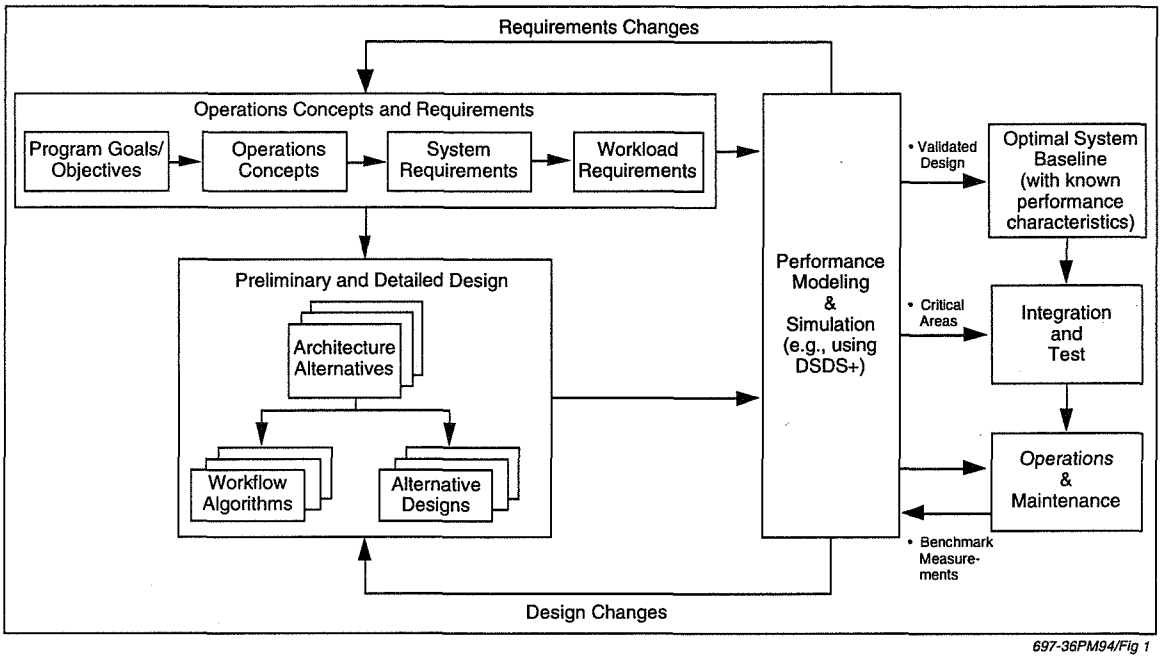


Figure 1. The Role of Modeling and Simulation in the Systems Engineering Life Cycle

spacecraft during each orbit, followed by periodic downlinking of the data during 10 minute contacts scheduled with the Tracking and Data Relay Satellite System (TDRSS). Modeling these scenarios provides estimates of the minimum onboard and ground-based storage requirements, and the minimum communications bandwidths necessary to distribute all of the data received during a downlink contact before data is received for the next contact period.

During the preliminary and detailed design phases, modeling is used to evaluate the performance of physical resources, configured in a certain topology to process the offered workload. The resources modeled include CPUs, busses, disks, networks, etc., and the workload includes software jobs/tasks to be executed, data to be processed/transferred, etc. Performance metrics generated by such a simulation include CPU utilization, queue sizes, network utilization, data latency, etc. Thus, simulation of the physical design adds an additional level of fidelity and insight into the anticipated behavior of the system, and the performance metrics generated reflect the practical constraints of the real system, above and beyond the theoretical minimums generated by modeling the operations scenarios.

During the integration and test phases, modeling is used to identify critical system functions and interfaces, and aspects of the system that have the smallest performance margins. Particular attention should be paid to these areas during testing, and the simulation results can be used to devise stress scenarios for subsequent testing.

During the operations and maintenance phase, modeling is used to evaluate the impact of any proposed changes to the system requirements or system design, such that the changes can be well-understood, and any side-effects identified. Further, performance benchmark measurements can be taken of the real system and compared against the simulated results generated in earlier life-cycle phases. These benchmark measurements can then be used to validate the simulation models (and, if necessary, to make refinements to the models), thereby enhancing the fidelity and level of confidence in subsequent simulation activities.

II. RECENT APPLICATIONS OF DSDS+

DSDS+ is currently being used at Goddard Space Flight Center (GSFC) to model the space and ground segments of the Earth Observing System, at Marshall Space Flight Center (MSFC) to model the Space Station Freedom Data Management System, and at

Johnson Space Center (JSC) to model the Space Station Freedom Control Center.

A major component of NASA's Mission to Planet Earth (MTPE) is the EOS program at GSFC. EOS encompasses many project boundaries, each responsible for different technical disciplines (e.g. spacecraft/instrument command and control, raw telemetry data processing, science data processing, data distribution, etc.); several of these organizations have utilized DSDS+ to conduct performance assessment studies germane to their areas of interest, and in addition, GSFC is sponsoring development of an end-to-end simulation model of EOS.

DSDS+ Model of End-to-End EOS System

The top-level schematic of the return-link, end-to-end data flows modeled for EOS is illustrated in Figure 2. The bullet-items listed to the right of each subsystem in the figure indicate those functions that have been modeled to-date. Other functions will be simulated in the near future, and the model will be updated as the EOS system definition evolves.

In addition to the wide range of functions noted on Figure 2, the following salient features of the model are worth pointing out:

- The simulation consists of a single, integrated model of three distinct segments of the EOS architecture: the EOS AM-1 spacecraft, the Space Network, and the EOS Data and Information System (EOSDIS).

- The end-to-end model is supplemented with more-detailed models of the Solid State Recorder, the Telemetry Processing Systems, and the network connecting the Science Data Processing Systems.

- The end-to-end model is being used to quantify the performance characteristics of the systems and sub-systems within each segment, as well as the performance impact of one segment on another.

- The fidelity of the simulation results is improved by reading external instrument timelines which specify the exact data rates of each instrument at

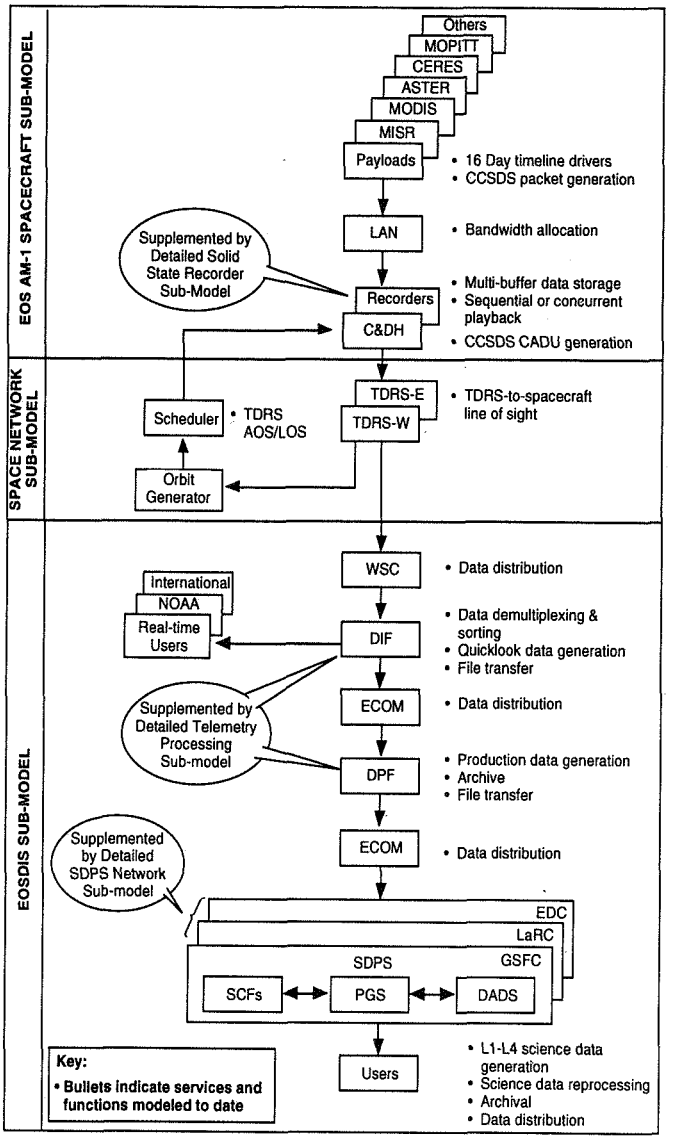


Figure 2. DSDS+ Model of End-to-End EOS AM-1 Architecture

each point in time throughout the 16-day cyclic period of the spacecraft. (The spacecraft makes successive orbits of the Earth, such that the entire surface area is viewed after 16 days, and then the cycle repeats.)

- Each iteration of the model (i.e. each "what-if" evaluation) is executed for a 16-day simulated period, corresponding to the spacecraft cyclic period. Each 16-day iteration takes less than 5 minutes to execute, due to the simulation optimization algorithms described in Section IV of this paper.

- The model generates hundreds of statistics that depict the performance characteristics from three perspectives: end-to-end, point-to-point, and sub-system by sub-system. For example, Figure 3 illustrates the end-to-end latency of NOAA data, assuming that there are no service interruptions in the system. As illustrated, in this scenario there is a 95% probability that NOAA will receive its data in 81 minutes or less, and none of its data will be delivered more than 127 minutes after the time of generation onboard.

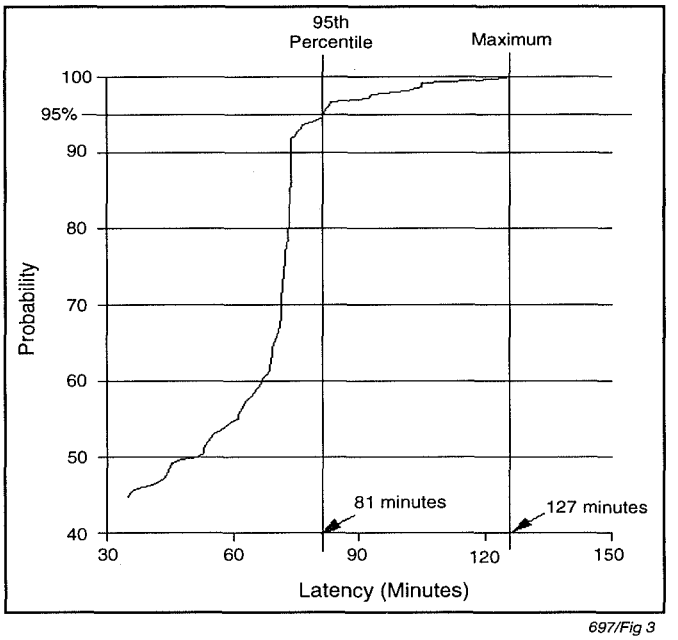


Figure 3: End-to-End Latency for NOAA Data

DSDS+ Model of EOS Solid State Recorder

During the last five years, several different technologies and management schemes have been proposed for implementation of the data recorders onboard the EOS spacecraft. The particular solutions proposed have had widely differing effects on cost, size, weight, shelf-life, maintainability, and performance. During this period, we have applied DSDS+ to evaluate the performance metrics of these different technologies, and we have determined factors such as: the number of recorders required, their capacities, their latencies, their required recording and playback rates, their impact on the ground data processing system, etc.

The most recent advances in technology now support high capacity, space-qualified, solid state recording devices (i.e. memory chips), with significant performance benefits. For example, these devices enable the different payload data streams to be written to different physical partitions, that can then be played back sequentially (thereby enabling high-priority data sources to be transmitted first), or they can be played back concurrently (thereby providing each payload with equal access to the downlink channel).

The DSDS+ results recently obtained by modeling the Solid State Recorders are illustrated in Figure 4. As indicated, the maximum buffer size required to support the EOS-AM1 payloads is approximately 122.5 Gbits, well below the planned capacity of 140 Gbits. However, these results are contingent upon the assumption that there are "near-perfect" operations throughout the end-to-end system. A more realistic assumption is that there are occasional service interruptions: for example, missed contact periods between the spacecraft and TDRSS due to loss of signal. The EOS-AM1 spacecraft makes 233 orbits during each 16-day cycle, and it is scheduled to receive two contacts with TDRSS during each orbit; i.e. it receives a total of 466 contacts per 16 day cycle. Therefore, we re-ran the Solid State Recorder model 466 times, missing a different TDRSS contact each time. As each simulation executed, we obtained the maximum buffer size observed during the 16 day simulated period; we then plotted the results, which are given in Figure 5.

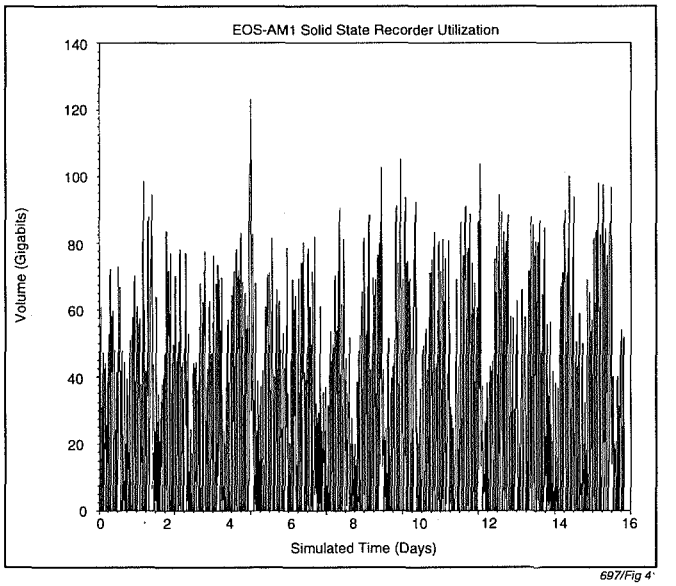


Figure 4. EOS AM-1 Solid State Recorder Utilization

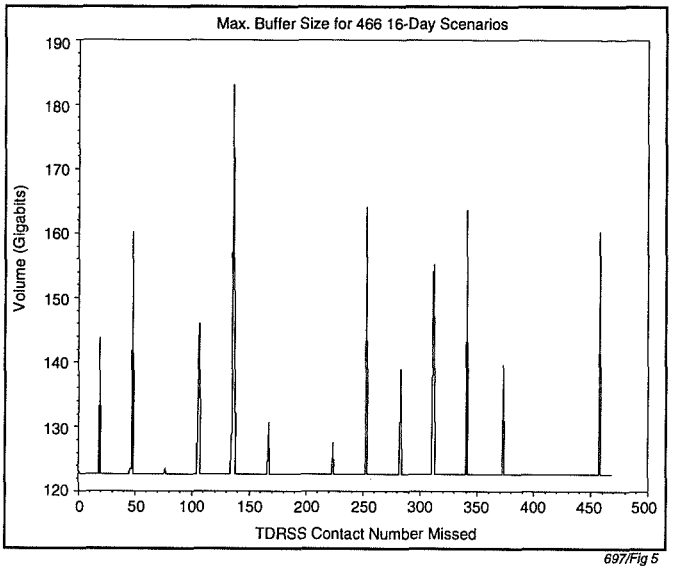


Figure 5. Maximum EOS AM-1 Solid State Recorder Utilization

As indicated in Figure 5, the volume of data buffered exceeded the Solid State Recorder capacity of 140 Gbits on eight occasions (e.g. when TDRSS contact number 19 was missed, when contact number 48 was missed, etc.). Therefore, there is approximately a 2% probability (8/466*100) that data will be lost if a TDRSS contact is missed. Also, it is worth noting that a TDRSS contact can be missed in the majority of cases without impacting the maximum volume of data that has to be recorded (i.e., the volume remains constant at 122.5 Gbits because the worse-case buffering occurs at some other point in the 16-day cycle, and is not related to the TDRSS contact that was missed).

III. DSDS+ OVERVIEW

The Data Systems Dynamic Simulator (DSDS+) is a general-purpose, discrete-event simulation tool. It contains an extensive library of pre-programmed simulation elements that are connected together by the user to represent the real system being modeled. Examples of the pre-programmed elements include: data generators and sinks, data processors (e.g. CPUs with various service disciplines), buffers and queues, and data switches and routers. Each of these elements simulates a particular function or service, which may be tailored by the user to represent the specific application being modeled. For example, the data generator has a list of parameters associated with it that enable the user to define characteristics such as the packet sizes to be generated, their inter-arrival times, their priorities, etc. If desired, multiple instances of an element may be included in the model (e.g. multiple data generators), and each instance will have its own set of parameters defining the specific operations being simulated.

Models are developed pictorially in DSDS+, using a graphical user interface that provides close correlation between the model representation and the real system. Further, the model drawings can be developed hierarchically, to any depth required, so that complex models can be decomposed into a series of detailed sub-level models, as illustrated in Figure 6.

As illustrated in the figure, events (i.e. messages) flow from element to element within discrete-event models. When the event arrives at an element, the underlying code associated with the element is executed, and some action is taken to simulate the operations of the real system. For example, an element that simulates the TDRSS propagation delay might hold the event for a quarter of a second before forwarding it to the next element in the model. A slightly more complex element might calculate the transmission delay by dividing the bandwidth (input as a user-supplied parameter associated with the element) by the size of the incoming event to be transmitted. As the model executes, simulation results can then be collected automatically, as a function of time, simply by observing the flow of events in the system, or by observing the sizes of the internal queues, etc.

It should be noted that DSDS+ events do not carry the real data with them in the model, but rather, they carry attributes that define the characteristics of the real data (such as the packet size). As illustrated in Figure 6, the events are held on a chronologically ordered list (called an event calendar) that is maintained by the scheduling engine. The engine removes the event from the top of the list, it instantaneously advances the simulation clock time to the new scheduled time, and it then forwards the event to the appropriate element for subsequent execution. Thus, there is no relationship between wall-clock time and simulated time, and the next event might be scheduled for processing in a (simulated) nano-second or a (simulated) day.

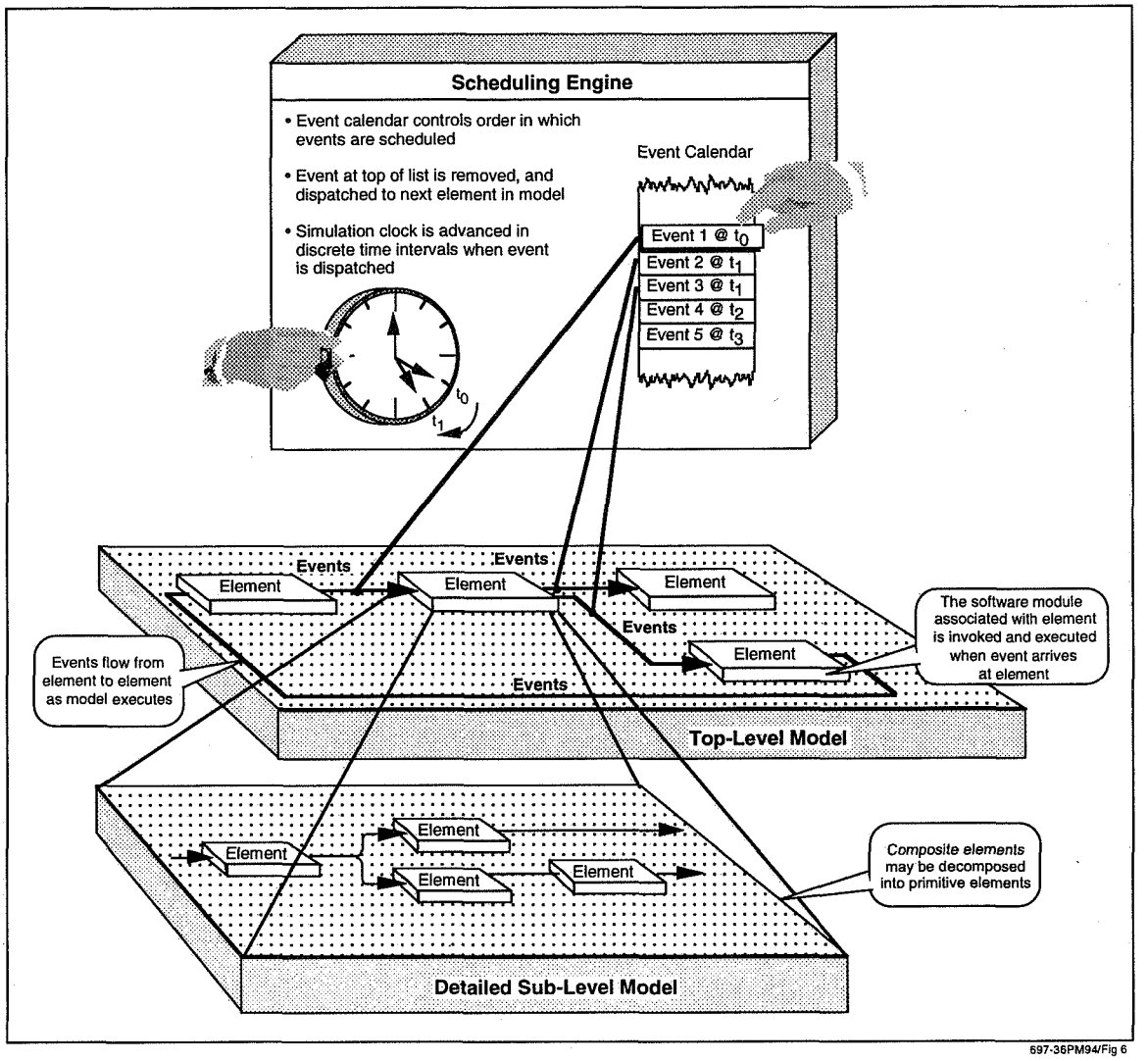


Figure 6. DSDS+ Simulation Concepts

However, the time required for a discrete-event model to terminate will increase with the total number of events to be processed. If each packet is modeled as an event, then end-to-end models of NASA's high data rate systems will require many months to terminate, even when executed on high performance workstation-class computers. The reason is obvious: the real system will be implemented by multiple "super-computers" distributed throughout the space and ground segments, each processing tens of thousands of packets per second. Therefore, how can a simulation model keep pace, since it is hosted on a single computer? We have implemented a solution to this problem within DSDS+, using a hybrid continuous-flow and discrete-event technique that we call "data streams". Briefly, the data stream methodology takes advantage of the fact that successive packets flow through a data system at a constant data rate, with relatively infrequent changes in the rate. Thus, the system can be modeled by considering the impact of what happens when the rate changes, without regard to the individual packets that constitute the data flow. For example, if during some time interval, a data source temporarily generates data at a rate that exceeds the processing capacity, then the queue size (and resultant queuing delay) will increase linearly with time until the source stops generating data, and then the queue size will decrease linearly with time (although the queuing delay will continue to increase linearly with time until the queue is empty).

The data stream approach is ideally suited to model NASA's data systems, since many of the science

instruments generate data at a constant rate during each duty cycle, with relatively infrequent rate changes. Therefore, a data stream model is required to process relatively few events (each of which represent a change in data rate), and it doesn't matter that the data rates themselves are extremely high (typically, up to 150 Mbps). As a result, we are able to utilize DSDS+ to model complex, end-to-end data systems, at a detailed-level, for very long periods of simulated time and yet generate the results within just a few minutes (for example, the 16 day simulations of EOS require less than 5 minutes to terminate).

IV. SUMMARY

The preceding sections have demonstrated that modeling and simulation are invaluable systems engineering tools to help define and select the optimal system configuration. Further, the performance characteristics of this system will be known prior to its delivery. This is not just because simulation results have been generated, but also because modeling is a two-way street, and the questions asked in order to develop a model usually prompt the systems engineer to resolve ambiguities or incomplete specifications that would otherwise have gone un-noticed. Therefore, it is our belief that the steps required to develop a model should be undertaken, even if the model itself is never actually constructed.

Simulation models are also relatively inexpensive to develop - far less than the cost of trying to correct performance problems subsequently found in the as-built system! For example, the DSDS+ simulation models of the EOS Solid State Recorder were developed in just a few staff-weeks, and yet their pay-off has been tremendous: the EOS project has decided to increase the recorder capacity to 200 Gbits to prevent loss of the science data.

Finally, we believe that the unique run-time optimization algorithms in DSDS+ make it the most suitable tool available to model NASA's end-to-end data systems. While there are many excellent commercial tools on the market, none contain any optimization methodologies; therefore, practical constraints limit their use to evaluation of localized systems, simulated for short time durations.

V. FURTHER INFORMATION

This paper is presented in conjunction with an online demonstration of DSDS+, including the simulation models developed recently of NASA's end-to-end data system.

DSDS+ is a NASA-owned tool, and therefore it is available free of charge to any NASA organization or support contractor. For further information, please contact Bill Davenport at (301) 286-5149, or at the address given at the top of this paper.

VI. ACKNOWLEDGMENTS

The Data Systems Technology Division, GSFC Code 522.1, is managing the development of DSDS+, and Stanford Telecommunications, Inc. is the implementation contractor, under the Systems Engineering and Analysis Support (SEAS) contract.

The following NASA organizations and programs have sponsored the development and/or application of DSDS+ to-date[1]:

GSFC Code 500 (1992 - 1993): Chief Engineer's Office
GSFC Code 502 (1986 - 1991): Customer Data and Operations System (CDOS) Project
GSFC Code 502 (1988 - 1990): Earth Observing System Data and Information System (EOSDIS) Project
GSFC Code 504 (1993 - 1994): Systems Engineering Office
GSFC Code 505 (1993 - Present): Earth Science Data and Information Systems (ESDIS) Project
GSFC Code 520 (1986 - Present): Data Systems Technology Division
GSFC Code 560 (1993 - Present): EOS Data and Operations System (EDOS) Project
HQ Code R (1991 - 1993): Office of Aeronautics and Space Technology (RTOP Project)
HQ Code S (1989 - 1991): Space Station Freedom Program

[1]Several of these projects/organizations have since been reorganized and renamed, but the affiliation names listed are the ones in effect when the work was sponsored.

N95-17578

111163

354257

P. 7

Analysis of Space Network Loading

Mark Simons
Goddard Space Flight Center, Code 534.2,
Greenbelt MD 20771
jsimons@gsfcmail.gsfc.nasa.gov
301-286-7323

Gus Larrson
AlliedSignal Technical Services Corporation, 1 Bendix Road,
Columbia MD 21045
gcl@npas16.atsc.allied.com
410-964-7664

Abstract

The NASA Space Network (SN) consists of several geosynchronous communications satellites, in addition to ground support facilities. Space Network management must predict years in advance what network resources are necessary to adequately satisfy all SN users. Similarly, users of the Space Network must know throughout all stages of mission planning and operations to what extent their communication support requirements can be met. NASA, at the Goddard Space Flight Center, performs Space Network and Mission Modeling using The Network Planning and Analysis System (NPAS), to determine the answers to these questions.

Introduction

The Network Planning and Analysis System (NPAS) is a deterministic modeling tool that accepts either generically or specifically stated communication support requirements. Using its own scheduling and orbital determination software, the NPAS produces an operationally valid schedule. Analysis software can then generate a variety of reports such as the percentage of satisfaction for users, or the total utilization of the SN. Analysts can view schedules graphically allowing them to identify conflicts easily.

Detailed in its approach, the NPAS can model such difficult problems as antenna blockage and support requirements based on the physical position of a user satellite. The tool can also model such factors as the effect of solar radiation on the spacecraft and the radio–frequency interference between users. Also, the NPAS is not limited to the NASA Space Network alone. Given the necessary information, it can model other space based communication networks, as well as ground–based networks, both foreign and domestic.

NASA has used the various versions of NPAS successfully over the past 15 years as a tool in analyzing Space Network loading. The NPAS has also changed with the times. It has recently been ported to a Unix–based workstation and has a new X–window graphical user interface. Currently, efforts are in place to develop a neural network application for the NPAS. This application could be used to obtain an instant response to many questions that arise during the planning of communication support for new space missions.

Overview of the Space Network

Modeling with the NPAS at GSFC is applied to NASA's Space Network (SN) . The space-based portion of the SN is referred to as the Tracking and Data Relay Satellite System (TDRSS). The TDRSS consists of five geosynchronous Tracking and Data Relay Satellites (TDRS), two of which are currently operational at 41° W and 174° W. Two others are held in reserve and the fifth, the oldest and least capable satellite in the network, is dedicated to support of the Compton Gamma Ray Observatory.

Only low earth orbiting spacecraft can make use

PAGE 1070 INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

of the SN. These users can communicate with a TDRS at K-band using one of two Single Access (SA) antennas or at S-band using SA or Multiple Access (MA) Ground-station limitations restrict the number of MAR users to to about five per TDRS. Even though each TDRS has only one MAF, the SA resource, due to its heavy use, is by far the most constrained TDRSS resource.

It is the job of the NPAS to determine how this network will perform in the future under changing conditions.

Modeling SN Loading

Modeling different aspects of SN loading with the NPAS proceeds along two lines, *Space Network Modeling* and *Mission Modeling*. In theory, SN Modeling methods hold user requirements constant and vary network resources. This is contrasted by Mission Modeling techniques which hold network resources constant and vary user requirements.

Analysts perform SN Modeling by determining what combination of TDRSS resources and mission priorities result in the best overall performance of the SN on a yearly basis. They make recommendations to NASA management, who then decide what course to adopt. These formally approved NPAS models are called *baseline models* and used as starting points for further studies. Using such approved baseline models, analysts perform Mission Modeling. Usually the requests come from management of new or prospective mission projects, or from management of existing mission projects that are contemplating changes to communications support requirements. During the course of mission modeling analysts produce *variant models* that reflect some change, or a collection of changes, to a baseline model. End products of Mission Modeling are termed *standard models* to differentiate them from true baseline models.

Regardless of the modeling method selected, analysts interact with the NPAS software in the same way, defining network resources, mission requirements, and evaluating resulting schedules.

Obtaining Model Parameters

The model parameters used by the NPAS can be divided into two parts, scheduling and coverage. Scheduling parameters include information about mission support requirements. Examples include minutes of support needed per orbit and the minimum separation between contacts. Coverage parameters deal with the geometry between user spacecrafts and network resources. Locations of the TDRS, the orbital elements of the user spacecraft, and information relating to the blockage of the user antenna are examples of such parameters.

The process of collecting user parameters is conducted in advance of actual modeling. This serves to reduce the needed during modeling to acquire needed information.

Capabilities of the NPAS

Once the modeling requirements have been gathered, the analyst can prepare to input the parameters into the NPAS. The number of steps required in this process is dependent upon the type of study being performed and similarities between the new requirements and existing baseline, variant, or standard models. Often times, minor modifications can be made to an existing model to analyze the new requirements.

Usually, the first step taken by an analyst is to define the support network. The support network can consist of SN or ground-based stations, and is referred to as the "network model."

Then, missions that would be users of the defined network for the year being studied are modeled. The combination of orbital and coverage parameters of the spacecraft with the scheduling requirements of the mission is referred to as the "mission model" for the particular mission.

Once the network and all appropriate mission models are created for the year being studied, coverage data is generated for each mission at each station. From this coverage data, and subject to any constraints defined in the

network and mission models, a schedule for each mission and each station can be generated.

This schedule can then be analyzed using a number of applications included in the NPAS package. Various mission and station report analyses can be requested, and a facility exists that allows an analyst to examine graphically the schedule and certain coverage events.

Modeling the Network

A network model in the NPAS can consist of stations, physical antennas, services, and crews. Hardware limitations, such as TDRS interface channels, may also be modeled.

SN and ground-network stations can have their locations specified in a number of ways. In particular, SN TDRS locations can be specified as either fixed or moving. In the former case, only the longitude and the height of the TDRS need to given. In the latter, orbital elements for the TDRS are specified by the analyst.

When creating the network model, the analyst has the ability to define constraints, such as service availability times and fixed down times. Other special-purpose constraints, such as allowing scheduling to occur on no more than six out of eight SA antennas simultaneously, also may be modeled. Events such as planned maintenance also are easily modeled.

These features allow the analyst to define real-world support network situations. Some examples might include TDRS' that are damaged or are otherwise not fully-functional. In the course of analyzing SN loading using baseline models, the number of single-access (SA) antennas available at each TDRS is modified often, and the NPAS Modeling Tool interface was designed to ease this process.

Defining Coverage Parameters

Embedded within the NPAS is a complete orbit and coverage generation system which uses a modified GTDS to generate station visibility data with accuracy to the nearest second. Additionally, a facility exists to accept externally-generated orbit data.

An NPAS analyst defines the basic orbital parameters of a spacecraft using Keplerian orbital elements. Other options exist that allow more complex orbit models to be created from this point. An example of such a model may be one that uses impulsive orbital maneuvering, thrust, or transfer orbits. Further modifications to the spacecraft orbit can include gravitational, solar and drag forces.

A number of mission-specific coverage events may also be calculated. Some of these mission-specific events include mission apogee and perigee points, ascending and descending nodes, spacecraft or subpoint sun events, land, water, and user boundaries. These events may be calculated in the coverage event generation, and would normally be used for scheduling options in the schedule parameters portion of the mission model.

Many missions in the loading studies performed using NPAS include mission-specific events to direct the mission scheduling. One of these missions is Landsat, which specifies a number of user and earth boundaries over which to schedule. Also, this and other spacecraft may only want to schedule when the spacecraft or the Earth sub-point is in sunlight, and this may be modeled using the mission-specific events.

As another coverage parameter, the analyst also can apply an antenna mask to the spacecraft, and define the spacecraft attitude and antenna orientation. Spacecraft masks are defined for a small number of missions in the most recent set of baseline models. The effects of masking on individual spacecraft has been extensively analyzed in variant and standard models for some missions, including EOS and the Space Station. For the latter mission, a number of masks reflective of the various "build" stages were modeled.

Options exist to determine separation angle events between a spacecraft and the sun or another spacecraft. Here, the angle apex may be located at either the spacecraft or the station, and the station may be either SN or ground-based. Separation angle events have been used in the past in variant models in which mutual interference between spacecraft was analyzed.

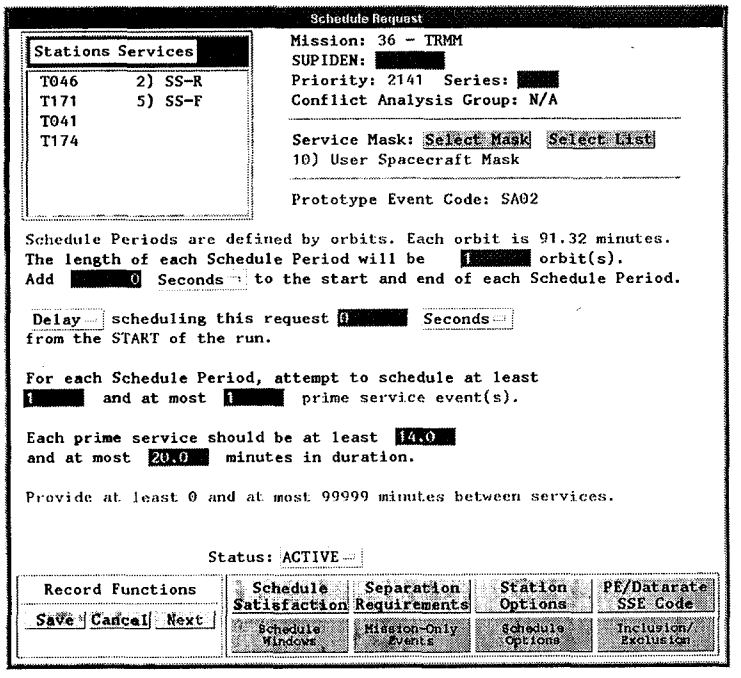


Figure 1- Sample Schedule Request

Defining Schedule Parameters

Tasks to be performed by each mission are modeled as schedule requests in the NPAS. A schedule request is simply a request for a service, or set of related services, of some time duration, to be scheduled over a set of stations. Each request is subject to a number of constraints that can be imposed by the network model, the given request, and other schedule requests. Additionally, some constraints, like mission pre-pass and post-pass times at various relay stations, can be applied to all the requests of a single mission.

Each schedule request is given a priority number by the analyst and is subsequently scheduled in "highest priority first" order. Requests for different missions may be intermixed to allow scheduling of all missions' critical requirements before any other requirements, if this is desired. The current practice, however, is to schedule all requests for any individual mission together, and place the missions into priority order.

Most schedule parameter modeling in the NPAS is accomplished using generic schedule requests. A generic request does not specify an exact time at which the defined service or set of related services is to be scheduled. Additionally, the request is typically repeatable over periods that extend across the schedule span.

Furthermore, using a generic request, an analyst can specify a variable length service contact time, in which shorter contact times, down to a minimum, can be accepted if the maximum length contact cannot be scheduled.

One mission that is modeled in NPAS using generic requests is TRMM. The primary request is one 20-to-14 minute S-band forward service, with concurrent S-band return service, event per orbit. Only one generic request would be required in the NPAS to model this requirement.

In the above example (Figure 1), the TRMM requirement was for two concurrent services, repeatable over the schedule span. Whenever

two or more services need to be related in some manner and scheduled repeatedly over the span, a Prototype Event (PE) structure is defined in the generic request. A PE is best viewed as a "template" for defining a variety of complex relationships between desired services.

There are a number of options and constraints that an analyst can model in each schedule request in order to accurately represent the request in NPAS and to emulate real-world situations. One of these options is the mission antenna masking toggle. This allows the schedule request to use, or not use, the mission antenna mask specified in the coverage parameters for the mission. This toggle is useful in determining the net effects of blockage on the spacecraft visibility.

A multi-mission shared resources option allows a number of selected spacecraft to communicate simultaneously over one physical antenna. This option could be used to model situations in which the Space Shuttle and its payload can communicate simultaneously on the same TDRS link.

Many other options exist as well, including dynamic rescheduling, in which selected higher-priority requests can be removed from the schedule if the invoking request did not attain a given satisfaction. Once the other requests are removed from the schedule, the invoking request is rescheduled, and the requests which had been removed are scheduled following.

Other special scheduling options include station and station antenna schedule preferences, forced-handover and hybrid support, and maximum elevation priority scheduling.

Some constraints might include a minimum or maximum separation between services, or scheduling windows, which define repeatable periods in which to schedule. Scheduling can also be directed around mission-specific events, such as when the spacecraft is in sunlight or when passing over a desired land mass, for example. The request can be directed to schedule when these events occur, or they can be directed to avoid scheduling at these times. Other mission-specific events include any that were defined in the coverage parameters for the mission.

Schedule Algorithm Summary

Three of the more commonly used schedule algorithms in the NPAS are the standard PE, standard non-PE, and geometric optimization algorithms.

Before any request is processed by these algorithms, the set of available time on individual station and mission antennas is generated. This set, referred to in NPAS documentation as "freetimes," consists of the mission visibility at each station with any time scheduled by previously scheduled requests removed. Additionally, any station or station antenna or service downtimes, as well as mission-oriented station service suppressions, would be removed from this set.

The standard PE and non-PE algorithms select events to schedule from this set using the aforementioned priority scheme in which higher priority requests are scheduled before lower priority requests. Further event scheduling is performed on a longest-prime-service-length-first, first-come, first-served basis.

The geometric optimization (GO) algorithm is similar to the standard algorithms, but the major difference is that this algorithm determines a large number of suitable schedules for each request and chooses the one that maximizes the total scheduled time for the request. In GO, an initial greedy schedule is determined by selecting the local-optimal solutions from partitions of the schedule span. If the total scheduled time of this solution is less than the predicted maximum, then a backtracking process is invoked in which the schedule is reevaluated using local-sub-optimal solutions from one or more partitions.

Post-Schedule Analyses

There are a number of applications available in the NPAS that allow an analyst to prepare different types of reports from the schedule results. For example, it is possible to create reports of utilization of service types, antennas, and frequencies. Analysts can also generate

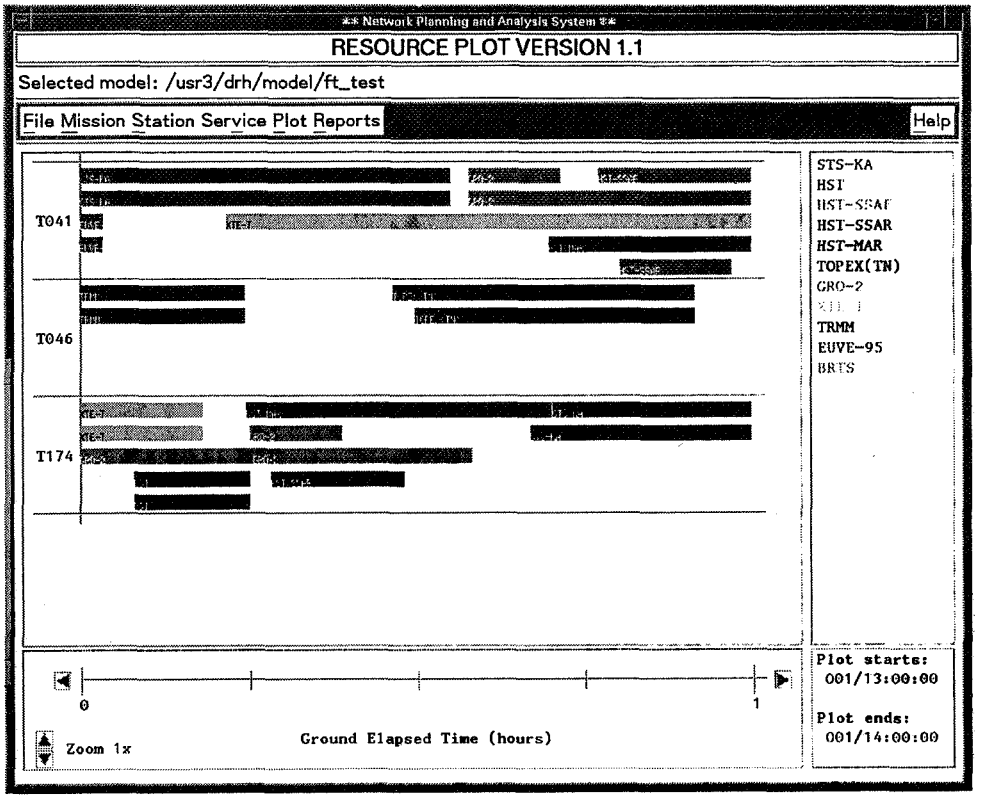


Figure 2 - Sample NPAS Schedule

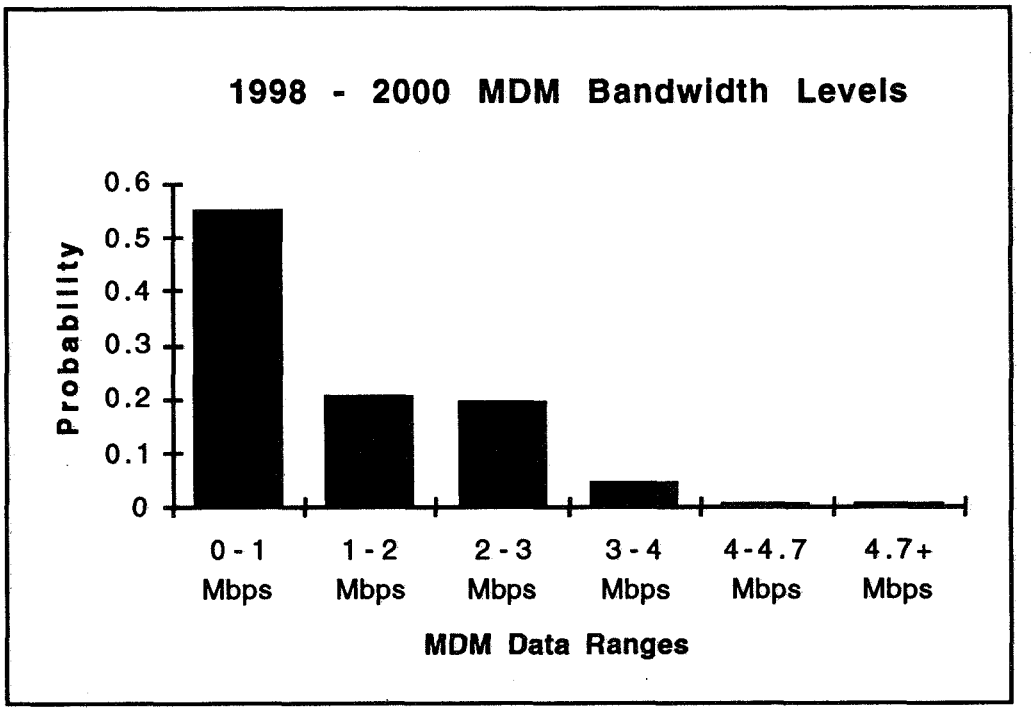


Figure 3 -Results from TRMM Study

reports of mutual interference between spacecraft. Communication channel loading can also be analyzed using an NPAS report application.

One of the more frequently used analysis tools is the NPAS Resource Plotter (RPLOT). RPLOT allows an analyst to view a mission's or station's schedule graphically in an X-Windows display. A wide range of options exists for examining mission- or station-related data, where different types of objects to be plotted appear in different colors. Mission-related data includes raw station antenna visibility and mission-specific events, such as apogee or a spacecraft-in-sun condition. Station-related data includes station antenna and service schedules. An example of an RPLOT display appears below in Figure 2.

Recent Work

As examples of our work, we have two recent mission studies performed for the Tropical Rainfall Measuring Mission (TRMM) and the Earth Observing System (EOS) projects.

The TRMM project will be launched in 1997 and will downlink data at 2 Mbps in real-time during its 20 minute SA events. Since ground terminal equipment (known as the MDM) limit the total downlink bandwidth to 6 Mbps, there was a concern whether or not the TRMM would experience data loss. Analysis using the NPAS took into account the downlink rates of all 1997 SN users, and showed that there is little to no probability that such data loss would occur. It also showed that loads on the terminal equipment follows an exponential distribution. Results are shown in Figure 3. We have also recently provided the EOS project with a projected schedule of the their EOS-AM1 spacecraft as modeled in 1998. This information will assist them as they size the needed on-board solid-state memory.

Future Directions

We are currently exploring aspects of artificial intelligence to help speed parts of SN Modeling. SN modeling can be a time consuming process. We are now developing a neural network application that, once properly trained, could be used to achieve instant responses to specific loading and user satisfaction questions. The application would be trained by automating numerous database modifications, generating schedules, and then collating and preparing the data.

NPAS is a changing modeling system that has adapted itself to the environment that it is designed to model. It has served NASA well over the years and will continue to play an important role in the analysis of Space Network loading.

N95-17579

111164

354258

p. 12

Modeling ESA's TT&C Systems

Enrico Vassallo
Stations and Communications Engineering Department
European Space Operations Centre
European Space Agency, Robert-Bosch-Strasse 5
D-64293 Darmstadt, Germany

ABSTRACT

After a brief introduction on the need for simulation packages for the analysis and design of satellite communications systems, the software tool developed for the European Space Agency (ESA), its main objectives and the design choices made during the development are presented. A very concise description of the available communications and measurement block follows. The ESA standard Telemetry, Tracking and Command (TT&C) system simulator is then introduced along with a description of the ESA standard modulation and coding schemes. As an example, the simulation of the ranging system which is a non-standard communications block, is described in details. Several examples of TT&C simulations outputs are given and compared with measurement results or theoretical approximations, when available. Finally, future developments like the support of advanced modulation schemes and the dynamic satellite link simulation are presented.

I. Introduction

As telecommunications technology progresses, new design tools are required by the system engineer to evaluate the performance of more and more complex systems and subsystems.
In fact, some equipment is so complex that no theoretical calculations can predict what the performance is going to be. Sometimes only simplified formulas or "rules of thumb" exist to qualitatively compare different hardware implementations. In either case some kind of tools are required to quantify system performance under real operating conditions before any expensive hardware bread-boarding or prototyping is attempted. Digital computer simulation for the analysis and design of communications systems is deemed to be a very powerful tool complementing both theoretical calculations and laboratory tests.
Therefore, the European Space Agency (ESA) has been supporting the development of a sophisticated and reliable simulation package for telecommunications systems which would cover all aspects of satellite communications, from radio frequency modulation to baseband encoding and decoding.
As the next step, the modeling of the ESA standard Telemetry, Tracking and Command (TT&C) ground station and satellite equipment based on the developed CAD package has been undertaken.
The requirement was to be able to evaluate telemetry, telecommand and tracking performance with an accuracy comparable to the measurement accuracy of real tests on flight and ground station hardware.
The main objectives of the TT&C simulator are broadly relating to the computer aided design and analysis functions: the optimization of the communications link of any ESA's mission, the setting-up of the various subsystems parameters, the preliminary assessment of radio frequency compatibility between the satellite and the station equipment, the estimation of the end-to-end system performance under the mission impairment conditions and operation modes, and the possibility of quantifying system degradation due to unexpected events in the mission lifetime.

PRECEDING PAGE BLANK NOT FILMED

PAGE 1078 INTENTIONALLY BLANK

II. The Telecommunication System Simulator

For the analysis and design of satellite communications systems, ESA has initiated a research program with the goal of developing a simulation package encompassing up-to-date communications equipment, with a high degree of flexibility in changing parameters and structures yet so user friendly to attract engineers not too keen in learning programming languages.
TOPSIM IV (TOrino Politecnico SIMulator, release IV) [5] is the result of this development. Being written in the FORTRAN-77 language, it can in principle be installed on any digital computer supporting a FORTRAN-77 compiler although the use of its sophisticated graphics interfaces requires that the X Window/Motif software be available.
The simulator is based on the time representation of signals whereby a band limited signal can be uniquely represented by a series of samples taken at the Nyquist rate or higher. This approach has been preferred to the frequency domain representation which is not optimal in dealing with feedback loops and non-linear devices. However, when complicated filters have to be simulated, the time domain approach may result in time-consuming simulation runs; therefore, simulation blocks operating in the frequency domain and integrated in the time domain by FFTs are available.
Linked to the time domain choice is the fact that telecommunications systems normally are characterized by a large ratio between the carrier frequency and the useful signal bandwidth. Therefore, the complex envelope representation of band-pass signals has been adopted to drastically cut down the sampling rate and thus increase the speed of execution. It is well known that a narrowband signal x(t) can be represented as:

$$x(t) = x_p(t) \cos 2\pi f_o t - x_q(t) \sin 2\pi f_o t \qquad (1)$$

where $x_p(t)$ and $x_q(t)$ are the complex envelopes of x(t) and f_o is the carrier center frequency. Each signal is therefore represented by a three-position vector where x_p, x_q and f_o are stored. The sampling rate is determined by the bandwidth of the useful signal (x_p, x_q) and not by the carrier frequency.
Since in the time domain representation of signal, the sampling rate must satisfy the Nyquist theorem for the element with the widest bandwidth, the multirate option is also available to further speed up the run time when different rates are used in different parts of the system (spread-spectrum systems for instance.) Special functions like pre-computation, program segmentation and block processing are also available to increase efficiency and user-friendliness.
A very large library of blocks is available (more than 300 blocks modeling communications devices and about 30 blocks performing various measurement functions.) The communications blocks encompass signal and random generators, analog and digital modems, analog channels and non-linear devices, analog and digital filters, carrier and clock recovery circuits, DSP modules, coders, decoders and trellis-coded modems. The measurement library includes qualitative measurements (eye patterns, scattering diagrams), statistical estimates (jitters), bit error rate routines, power and power spectral density evaluation.
User-defined blocks simulating more complex subsystems can be written from scratch on a default template or by using the supplied TOPSIM routines as elementary building blocks, and then included in a personalized library.
Flexibility, one of the goals of the simulator, is achieved in two different ways: first, none of the parameters of the library blocks is fixed to a default number; secondly, the activation of different parts of the simulation (program segmentation) to compare different configurations is supported by means of simple logic variables.
Application programs can be very easily written by drawing the system block diagram on the screen with the graphic input interface (GII) and letting it convert the drawing into TOPSIM code. Similarly, the simulation outputs can be displayed by the X Window based graphic output interface (GOI) either in on-line or off-line mode.

III. The ESA Standard TT&C Simulator

Of the many different application programs written with TOPSIM, SIMSAT has the task of dimensioning all the parameters involved in the link between a TT&C Earth station of the ESA Tracking (ESTRACK) network and an ESA, or CCSDS compatible, Near-Earth or Deep-Space Spacecraft, or Geostationary Satellite.
The program allows simulation of simultaneous transmission of telecommand, telemetry and ranging signals according to the various ESA standards [1]-[4], and models standard ESA Earth station and spacecraft equipment, although different subsystems characteristics can be easily introduced should the need arises.
The most difficult task SIMSAT supports is the selection of telemetry and telecommand subcarrier and ranging tone frequencies, and their modulation indexes minimizing mutual interference and intermodulation due to transponder non-linearity and modulators' spurious signals.
Available simulation outputs are qualitative measurements (eye patterns, scattering diagrams), statistical estimates (timing and phase jitters, ranging and Doppler mean and r.m.s. values), error rates (telemetry and telecommand bit and symbol error rates, ranging erroneous ambiguity resolution probability), power measurements and power spectrum evaluations.
Being SIMSAT based on the ESA funded satellite communications simulator, most of the subsystems used in both the Earth and the space segment are standard library blocks. For those, trimming the various parameters and introducing non-ideal effects (imbalances, skew, non linearity, AM/PM, phase noise, etc.) according to both equipment specifications and measurement results has been the major task in building the simulator.
On the other hand, the blocks simulating the ranging equipment had to be written from scratch. Ranging/Doppler subsystems are in fact not normally contemplated among basic communications equipment.
The standard ESA tracking system, called the Multi Purpose Tracking System (MPTS) has been fully modeled, including its sequence of operations (carrier and tone acquisition and tracking, code ambiguity resolution, range and Doppler measurement.)
Among the blocks written to simulate the MPTS are the ranging code and tone generators, the replica code and tone generators, the frequency-steered digital tone phase locked loop, the IF and digital correlators, the time interval counters and the processing and control module making sure that the sequential steps of the tracking process are correctly performed.

A. The ESA TT&C Systems

The standard ESA uplinking of commands to the spacecraft (telecommand) is specified to use the following modulation scheme: the telecommand data, which is binary Non-Return-to-Zero-Level (NRZ-L) encoded, phase shift-keys (PSK) a sinusoidal subcarrier (8 or 16 kHz); the composite video signal then phase modulates (PM) the sinusoidal uplink carrier together with the ranging video signal.
The ranging signal [2], [7], [8] is a hybrid signal composed of a special code phase modulating the ranging tone. The resulting video signal phase modulates the uplink carrier sharing its power with the telecommand signal.

The uplink signal is therefore given by:

$$Su(t) = \sqrt{2Pu}\cos[2\pi fo\, t + m_{TC}\, S_{TC}(t) + m_{RG}\, S_{RG}(t)] \quad (2)$$

where

$$S_{TC}(t) = d_{TC}(t)\ \cos(2\pi f_{TC}\, t + \emptyset) \quad (3)$$

is the telecommand video signal,

$$S_{RG}(t) = \cos(2\pi\ f_r\, t + m_r\, r_n(t)\ ) \quad (4)$$

is the ranging video signal, and

Pu	: uplink signal power
fo	: uplink carrier frequency
m_{TC}	: telecommand uplink modulation index
m_{RG}	: ranging uplink modulation index
$d_{TC}(t)$	: telecommand baseband data stream
$r_n(t)$	: ranging code
f_{TC}	: telecommand subcarrier frequency
f_r	: ranging tone frequency
m_r	: ranging code modulation index
Ø	: telecommand subcarrier initial phase

Telecommand and ranging uplink modulation indexes are selected in order to optimize the TT&C link budget.
For the telemetry signals transmitted from the spacecraft to the Earth station, both NRZ-L and Split-Phase-Level (SP-L) binary encoding, Reed-Solomon, Convolutional or Concatenated (Reed-Solomon plus Convolutional) channel encoding, and sine-wave or square-wave subcarriers may be selected depending on the bit rate and the mission requirements. The resulting video signal finally phase modulates the downlink carrier with the ranging signal which has undergone phase demodulation, filtering and automatic gain control (AGC). Due to the limited filtering performed in the transponder, the ranging signal is normally accompanied by the fed-through telecommand signal. Therefore, telemetry, telecommand, ranging and thermal noise share the downlink power.

The downlink signal is given by [8]:

$$Sd(t) = \sqrt{2Pd}\, Re\{\exp\{j\ [2\pi fc\, t + m_C\, S'_{TC}(t) + m_R\, S'_{RG}(t) + m_{TM}\, S_{TM}(t)\ + m_e\, n(t)\ ]\}\} \qquad (5)$$

where

$$S_{TM}(t) = d_{TM}(t)\ \cos(2\pi f_{TM}\, t + \theta) \qquad (6)$$

is the telemetry video signal in case of sinusoidal subcarrier, and

$S'_{TC}(t)$ is the filtered and level controlled telecommand video signal,

$S'_{RG}(t)$ is the filtered and level controlled ranging video signal,

n(t) is the thermal noise in the transponder ranging channel, and

Pd	: downlink signal power
fc	: downlink carrier frequency
f_{TM}	: telemetry subcarrier frequency
m_C	: telecommand echo modulation index
m_R	: ranging effective downlink modulation index
m_e	: noise downlink modulation index
m_{TM}	: telemetry modulation index
$d_{TM}(t)$	: telemetry baseband data stream
θ	: telemetry subcarrier initial phase

Calculations of the downlink modulation indexes are reported in [6] whereas analytical expressions for $S'_{RG}(t)$ are given in [8].

B. Modeling the Tracking System

The ranging and Doppler tracking system (MPTS) has been modeled by writing a series of user defined blocks. Only the most important features will be described here. Detailed descriptions of the equipment can be found in [7], [8].
The ranging modulator is made up of a special code generator, a tone generator and a linear modulator where the code synchronously modulates the tone with three operations dependent modulation indexes. The code is a periodic signal actually composed of subcodes in a proper sequence and was design to allow fast ambiguity resolution. The resulting video signal is fed to the uplink modulator for modulation with or without the telecommand signal.
The Doppler unit performs integrated Doppler measurement on the downlink carrier and estimates the expected ranging tone frequency.
The ranging demodulator performs an I-Q correlation between the received IF ranging signal and the replica ranging tone generated on the information from the Doppler unit. After conversion to baseband by multiplication with the recovered carrier, the phase error is sent to the digital tone PLL. This technique (tone frequency steering) has been devised to use very narrow loop bandwidths yet without having too long acquisition times. When the loop is locked, the quadrature correlator IF output is correlated with the locally generated replica code in phase and quadrature. The downconverted output is filtered and fed to the code ambiguity resolver where the ambiguity resolution logic is implemented.
The processing and control module interfaces with the various units, supervises the various stages involved, i.e. the carrier and tone acquisition, the sequencing of codes for ambiguity resolution, the measurement proper, and generates the required statistical outputs (range and Doppler jitter and bias, probability of erroneous ambiguity resolution, etc.)

C. Simulation Examples

Fig. 1 shows the simulated ESA TT&C space and ground segment. Some of the blocks shown are actually macro blocks made up of several elementary blocks like the Ranging System whose internal structure is depicted in fig. 2.
Fig. 3 is a typical result of the system and detailed design phase of a project, the optimization of the satellite transponder back-off. The figure depicts the estimated telemetry BER for the ISO spacecraft as a function of the selected back-off.
Fig. 4 shows the simulated ranging spectrum of a CCSDS compliant mission whereby the ranging tone is fixed at 100 kHz. The tone itself, the sideband created by the code modulation and the odd harmonics of the tone are visible. This kind of plots can be used to select the telemetry subcarrier frequency.
Fig. 5 shows the worst case in-phase and quadrature ranging correlators output for the ISO mission and is used to determine the minimum code integration time necessary to perform the ambiguity resolution with the mission specific probability of error.
Fig. 6 shows the in-phase ranging correlator output during ambiguity resolution for the CLUSTER spacecraft when no noise is present. The actual curve shows a 10% reduction with respect to the design value, due to the limited bandwidth of the flight transponder. This example demonstrates the tool capability to quantify system performance degradation caused by subsystem non compliant with specifications.
Fig. 7 and 8 respectively show the simulated and measured spectra at the output of the transponder. Due to the limited bandwidth of the modulator, spurious lines at the even harmonics of the bit rate are generated. Note the almost perfect match between simulation and measurement.
Comparisons between simulated and measured telemetry bit error rates (BER), for the cases of telemetry only and simultaneous telemetry and ranging, are shown in fig. 9. The theoretical BER value for the telemetry only case is also included. The maximum difference between simulation and measurement is some 0.2 dB, of the same magnitude of the test equipment measurement accuracy.

IV. Future Developments

ESA is currently considering the use of bandwidth and power efficient (suppressed carrier) and spread-spectrum modulation schemes for TT&C support of future missions.
The next generation modems are being introduced in SIMSAT to replace the present standard PSK/PM modulation. Besides, since suppressed carrier signals are deemed more sensitive to Doppler shifts and rates, a dynamic satellite link simulator will be added to TOPSIM. The link simulator is to compute the link geometry parameters (slant range, Doppler shift and rate, elevation angle, etc.) and derive link budget parameters (carrier-to-noise density ratio, Eb/No, etc.) versus time. The generated output file will then feed the present simulation program so that a dynamic simulation is performed.

V. Conclusions

Based on TOPSIM, a digital computer simulation tool tailored to ESA's requirements in the field of space communications, the simulation template for ESA's standard ground and space TT&C equipment has been developed. The very accurate modeling of the various ground and space TT&C subsystems has resulted in very good matching between simulation results and measurements performed on the equipment itself. Thanks to the achieved accuracy, the simulator is very extensively used:

- during conceptual system design (feasibility or Phase A) to trade off various configurations of the same system or different systems performing the same functions;
- during detailed design phase (phase B) to determine mission specific set-ups (modulation indexes, loop bandwidths, correlation times, carrier and subcarrier frequencies, etc.);
- during the implementation phase (phase C/D) to evaluate system performance under the predicted mission impairments for which analytical solutions do not exist;
- after launch to simulate the effects of subsystem degradation's occurred during the mission and validate corrective actions on a model prior to trying it out on the flying spacecraft.

Therefore, although final radio frequency compatibility tests between the space and the ground segment are performed on the actual hardware as prescribed by the ESA standards [1], the complete design of the TT&C link is done by simulation. Potential problems are likely to be discovered by simulations much earlier than the expensive flight hardware is available for testing thereby potentially minimizing schedule risks and program costs.

The performance of the simulator with respect to the actual equipment has encouraged the development of a new template including future ESA modulation schemes and a dynamic satellite link simulator.

VI. References

[1] European Space Agency (November 1989), *RF & Modulation Standard,* PSS-04-105, Issue 1.

[2] European Space Agency (April 1990), *Ranging Standard, Vol. I, Direct Earth-to-Space Link,* PSS-04-104, Issue 1.

[3] European Space Agency (September 1989), *Telemetry Channel Coding Standard,* PSS-04-103, Issue 1.

[4] European Space Agency (March 1979), *S and S/X Band Coherent Transponder Specifications,* PSS-48, Issue 1.

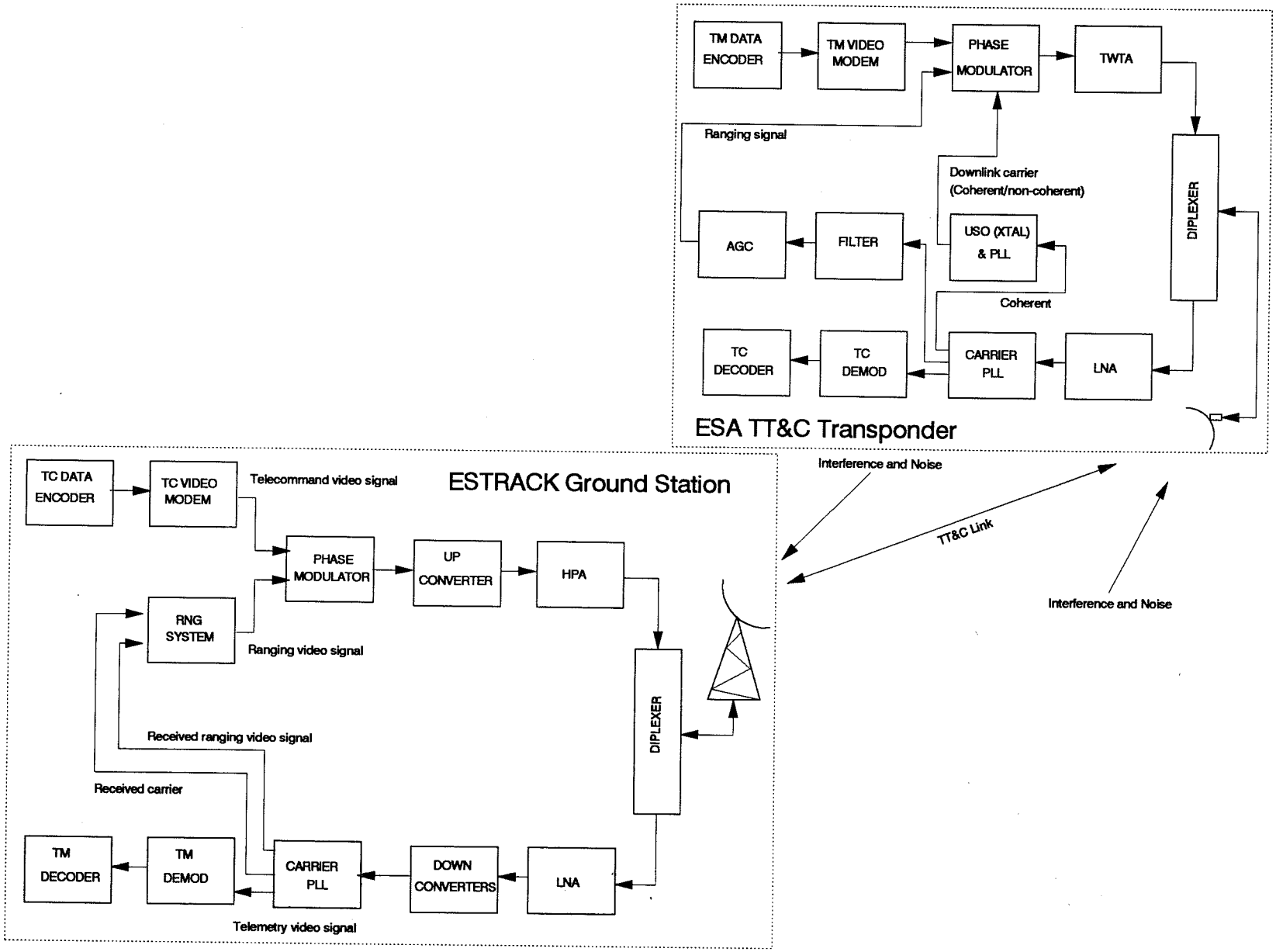
[5] European Space Agency (1986), *TOPSIM IV, Design and Implementation of Software for Simulation and Analysis of Communication Systems,* ESA/ESTEC Contract No. 6981/86/ML/JG.

[6] R. De Gaudenzi and M. Nahvi (1989), *Telemetry degradation due to the ranging signal of the multi purpose tracking system,* CCSDS proceedings, RF and Modulation Subpanel1E, NASA Ames Research Center, CA.

[7] R. De Gaudenzi, E. E. Lijphart and E. Vassallo (1990), *The New ESA's MPTS,* ESA Journal, Vol. 14, No. 1.

[8] R. De Gaudenzi, E. E. Lijphart and E. Vassallo (1992), *A New High Performance Multi-Purpose Satellite Tracking System,* IEEE Transactions on Aerospace and Electronic Systems, Vol. 28, No. 4.

Fig. 1 - ESA TT&C Schematics



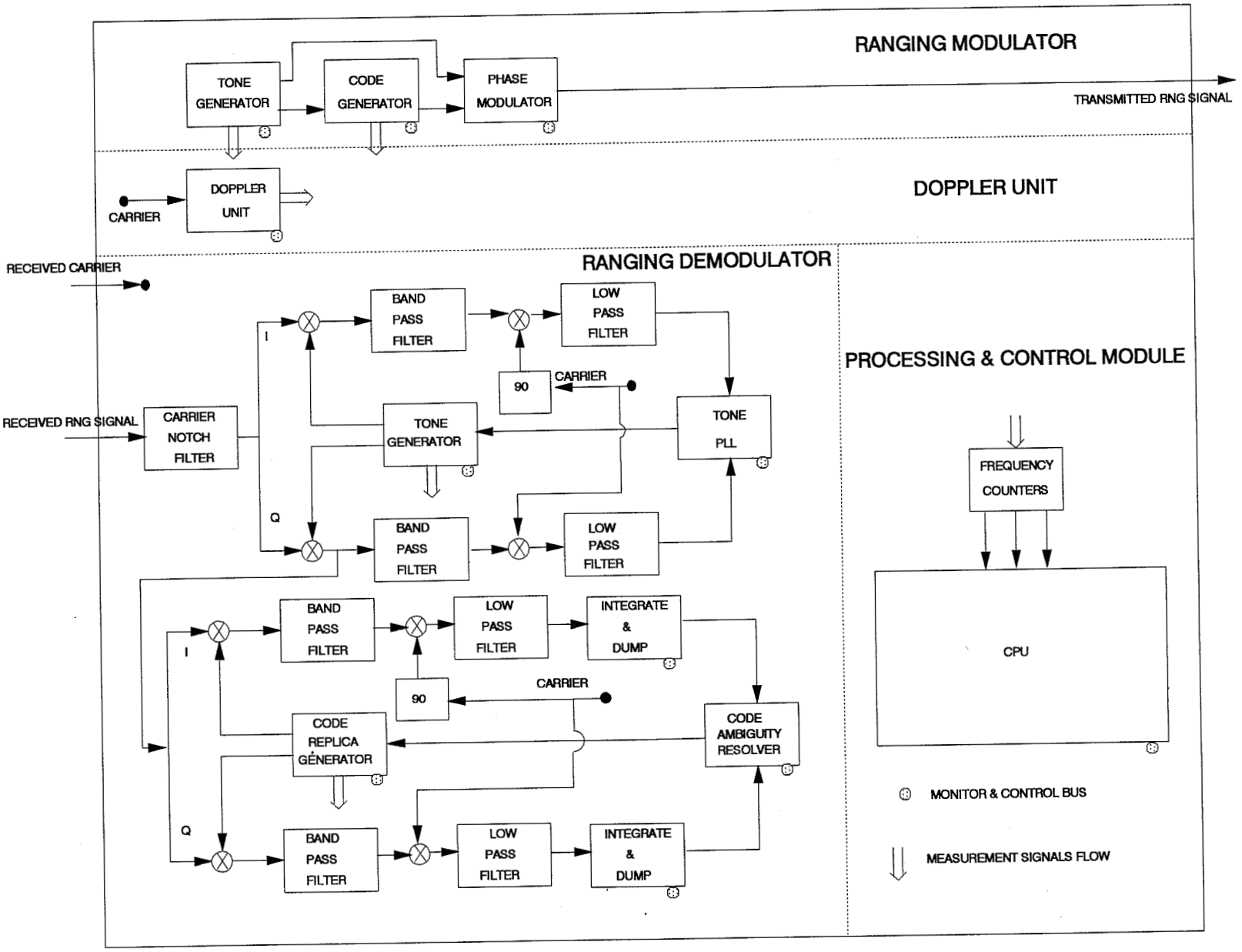


Fig. 2 - Ranging System (MPTS)

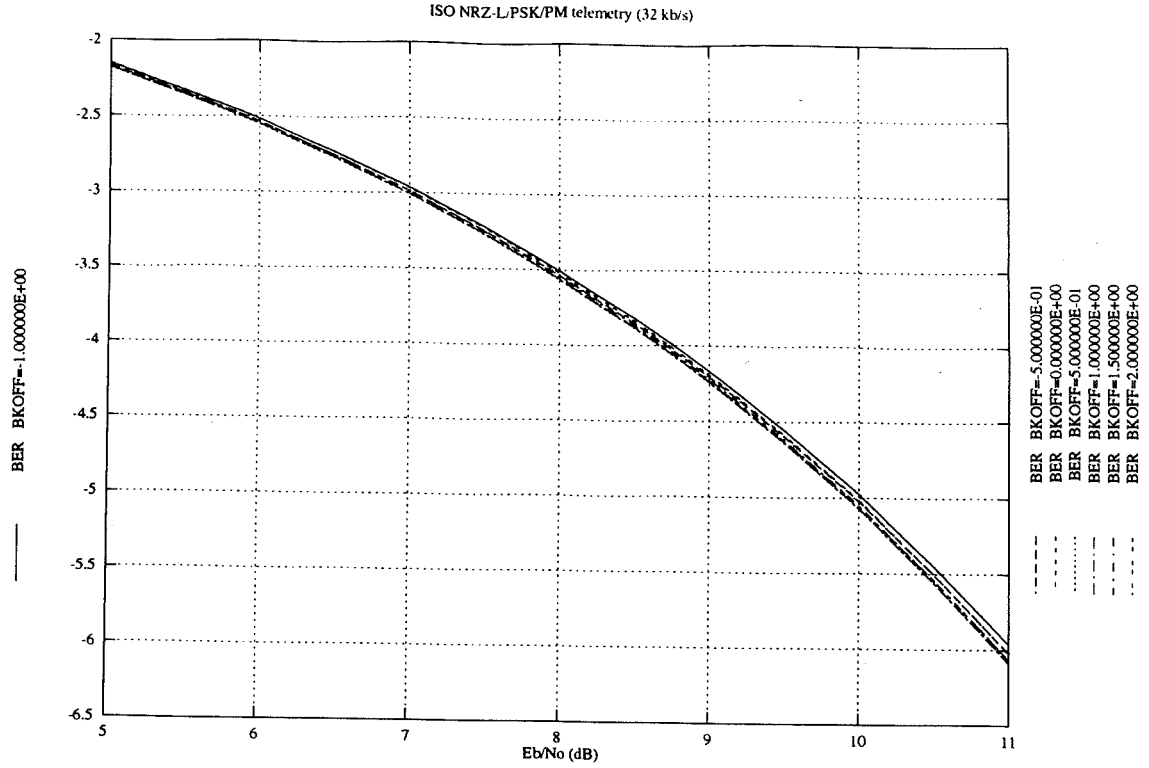


Figure 3. Telemetry BER vs. TWTA back-off

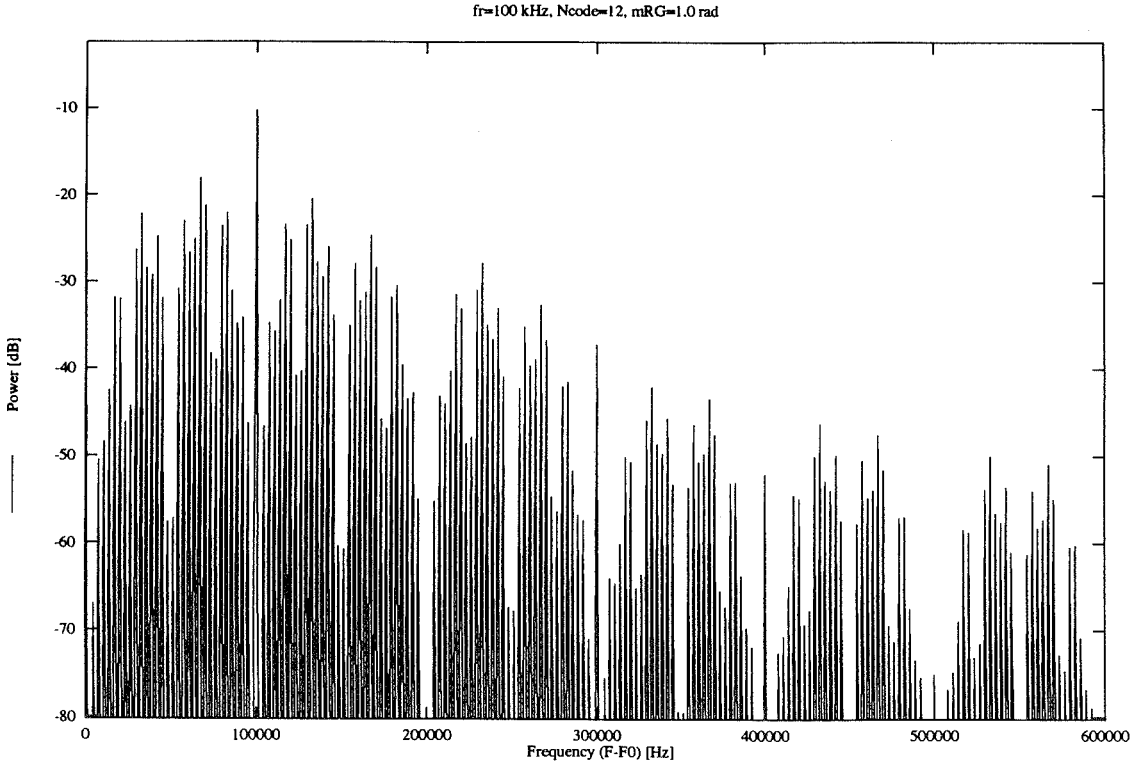


Figure 4. Ranging signal spectrum

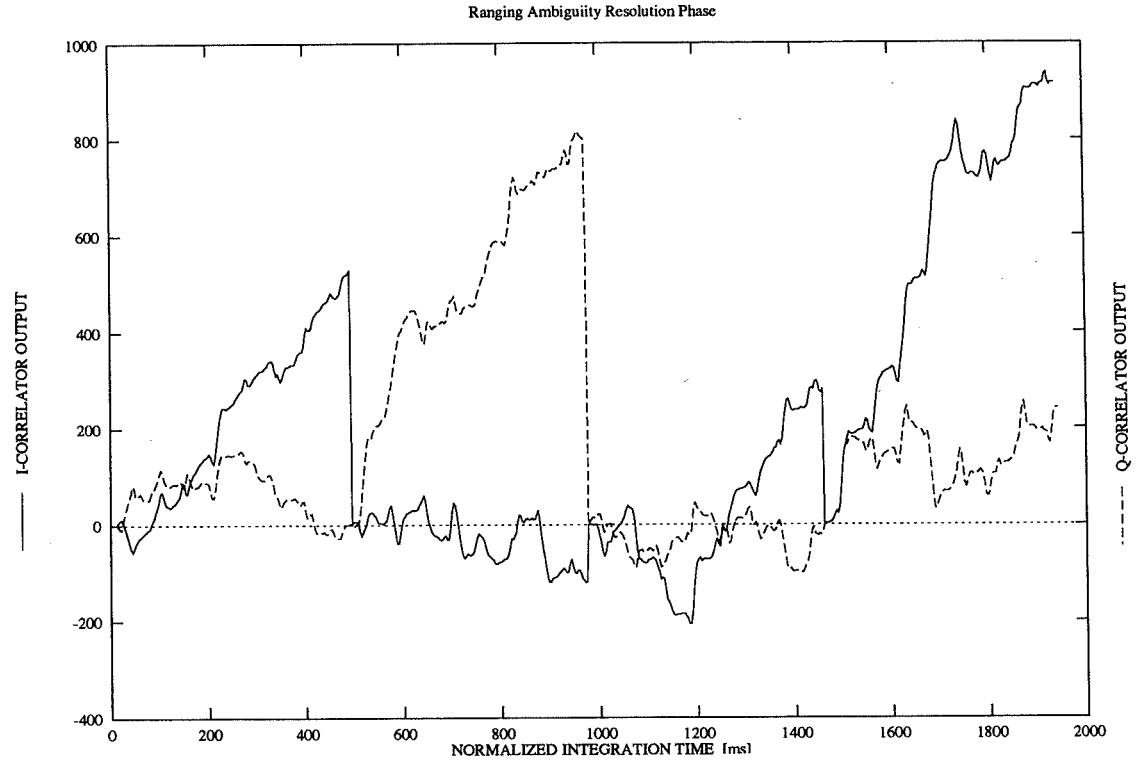


Figure 5. ISO Ranging correlators output

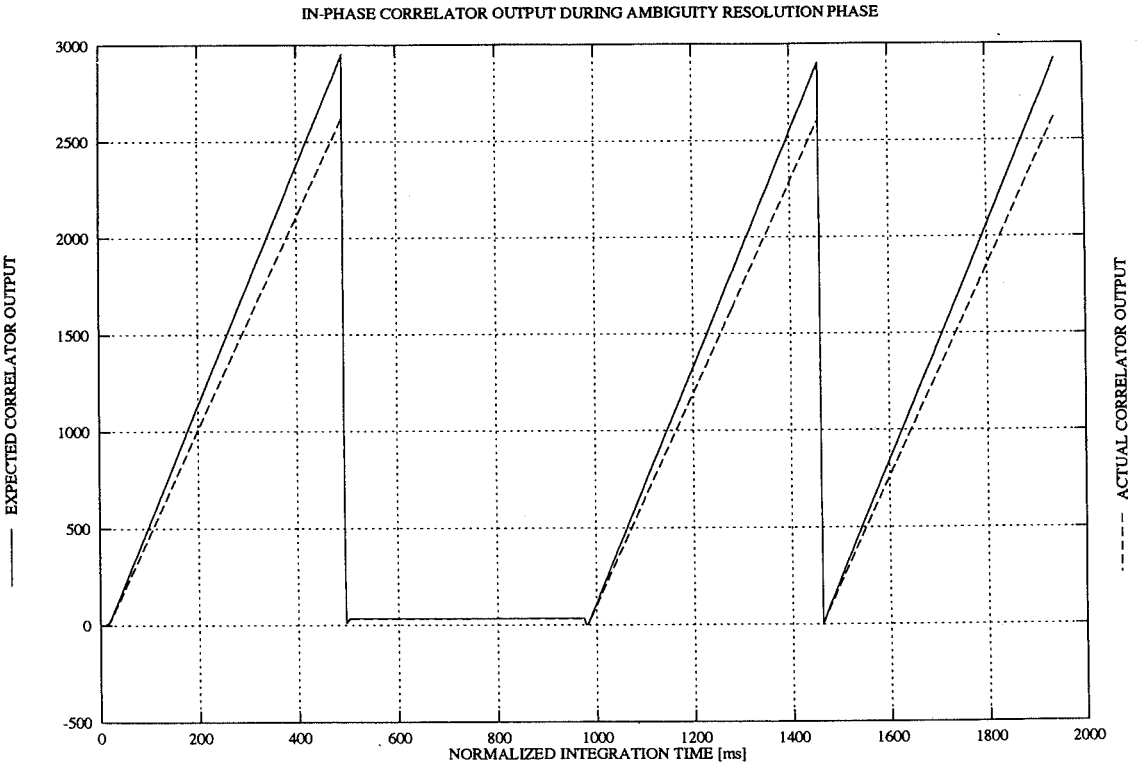


Figure 6. CLUSTER Ranging Correlation

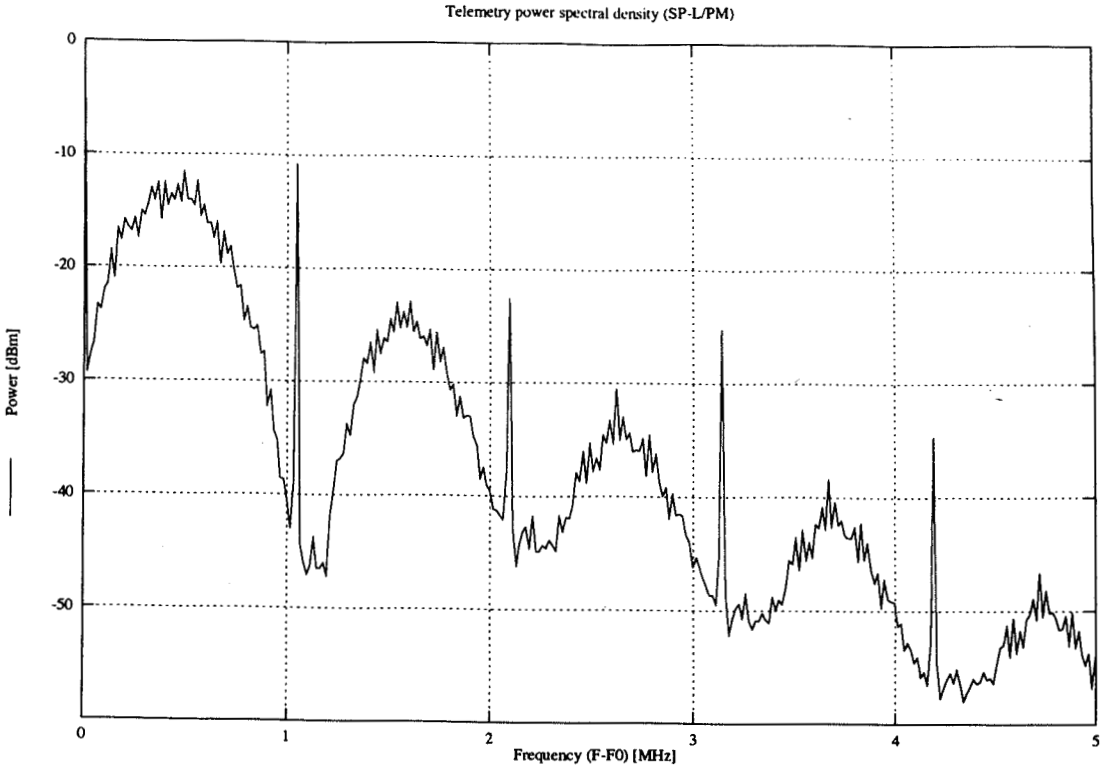


Figure 7. Simulated transponder output

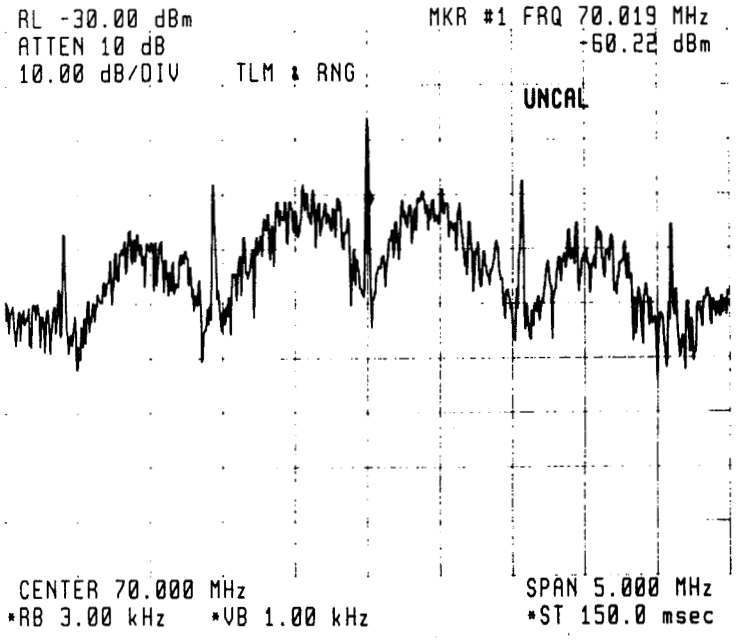


Figure 8. Measured transponder output

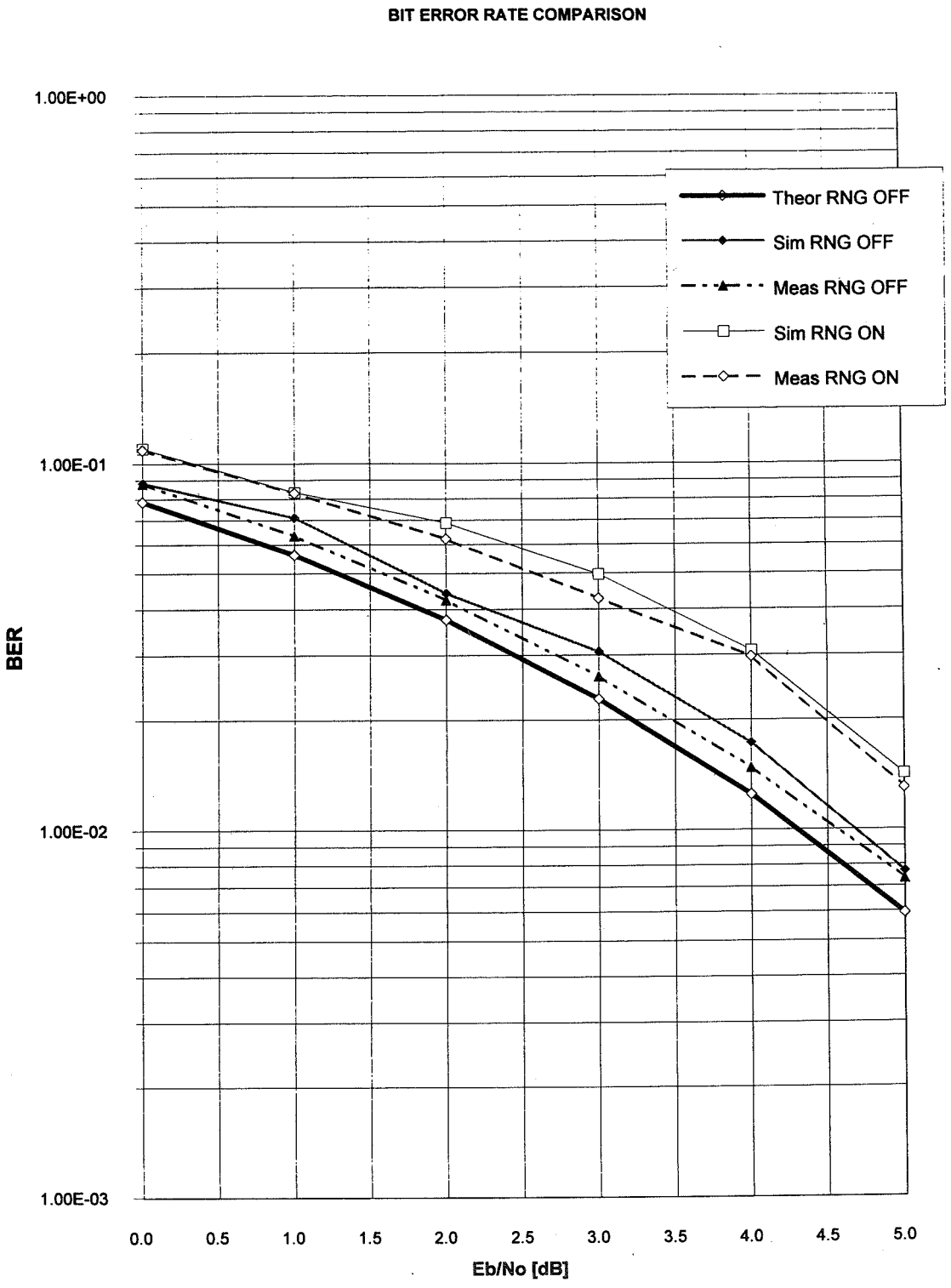


Figure 9. Simulated and measured telemetry BER

Systems Development

* *Presented in Poster Session*

111165 N95-17580

A General Mission Independent Simulator (GMIS) and Simulator Control Program (SCP)

354259
p-8

Paul L. Baker (GST Inc.)
J. Michael Moore (NASA/GSFC)
John Rosenberger (CTA Inc.)

The Purpose of GMIS and SCP

GMIS is a general-purpose simulator for testing ground system software. GMIS can be adapted to any mission to simulate changes in the data state maintained by the mission's computers. GMIS was developed in Code 522 NASA Goddard Space Flight Center. The acronym GMIS stands for *GOTT Mission Independent Simulator*, where GOTT is the *Ground Operations Technology Testbed* . Within GOTT, GMIS is used to provide simulated data to an installation of TPOCC - the Transportable *Payload Operations Control Center* . TPOCC was developed by Code 510 as a reusable control center. GOTT uses GMIS and TPOCC to test new technology and new operator procedures.

Ideally, mission operations staff should have a variety of simulators to serve several purposes:

- Prediction - compute the future state of a system
  - Evaluate the effects of a proposed operational step, i.e., to answer "what if" questions.
  - Verify that the planned steps will cause operations that lie within safety and other operational constraints.
- Test – supply a time-variable system state to exercise subsystems.
- Training – create a realistic environment for training staff.

In practice, missions that use TPOCC have one or more simulators. Consequently, GMIS was not developed to fill a void; rather, it was developed to offer an alternative with certain advantages:

1) Convenience - GMIS is easy to setup and use.
2) Extensible - it is easy to add more simulation functions.
3) Speed - eventually, we expect GMIS to run very quickly.

In the present version, we have not achieved these goals in equal measure. The convenience factor is high, but the speed seems modest. The features that make GMIS extensible are useful, but there is room for improvement. In this report, we will relate some feedback from current GMIS users and indicate how we plan to improve the simulator in these three areas.

The GMIS manages the timing and external data links for optional simulation modules. It accepts any number of compiled or interpreted modules. Compiled modules are written in C or C++. The interpreted modules are written as procedures in TSTOL - *TPOCC System Test and Operations Language* . This language is familiar to flight operations team members, but it is not especially easy to use. In fact, programmers often find it difficult to use because it looks familiar but has a different syntax compared to programming languages. For this reason, the project developed the SCP as a convenience feature.

The SCP is a graphical, syntax-aware editor for TSTOL. Although SCP is really a simulation script editor, its name stands for *Simulator Control Program,* for historical reasons. SCP helps you write a correct TSTOL procedure and then lets you run it with a click of a button. SCP has an embedded copy of the TSTOL interpreter so that it can detect and report syntax errors locally. Finally, SCP reads and displays all the variable names in the data server's database. That feature helps the user find the correct spelling for system variable names.

PRECEDING PAGE BLANK NOT FILMED
PAGE 1092 INTENTIONALLY BLANK

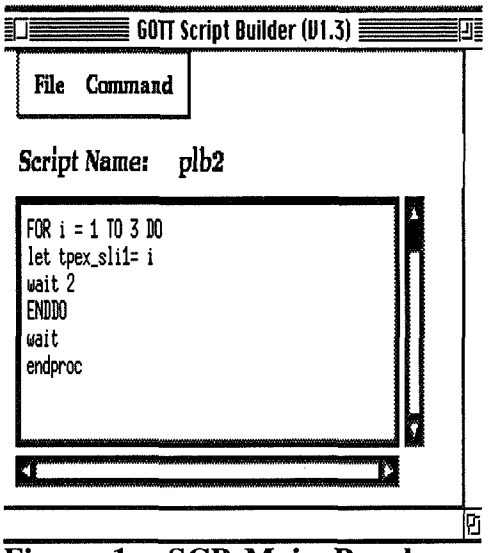


Figure 1: SCP Main Panel

SCP and GMIS Interaction Panels

Simple and easy to use Motif control panels are responsible for much of the convenience of GMIS/SCP. The panels strive for the same look-and-feel as the panels that are used in the TPOCC control centers.

The SCP has a main panel, called the Script Builder, that is used to edit TSTOL scripts. Figure 1 shows a copy of the panel at a point where the user has completed a simple script.

This script will loop three times with two seconds per loop, and it will set the value of the loop index, `i`, in the system variable, `tpex_sli1`.

The script appears within a Motif Text Widget. All of the Widget's editing commands are available. The File Menu has the usual options for saving and retrieving copies of the script. The Command Menu has only one option: Execute. When the user selects that option, the script runs and sends data values to GMIS. From GMIS, the updated values find their way to the data server.

GMIS always shows the updates to values when it receives them from SCP. The GMIS panel is shown in Figure 2 just after the SCP has executed the script in Figure 1. Compiled simulation modules are usually designed for a higher throughput and could swamp the display with output. Consequently, GMIS does not automatically show updates for such modules. However, compiled simulation modules can write progress messages to the display, if they wish.

The SCP has two panels that help a user write TSTOL. Suppose we want to add another statement to the procedure. We only need to click the mouse at the point where we want the new statement to appear. Then, we can go to the Statement Builder panel and pull down the Script menu. That menu shows the basic statements of the TSTOL language as illustrated in Figure 3.

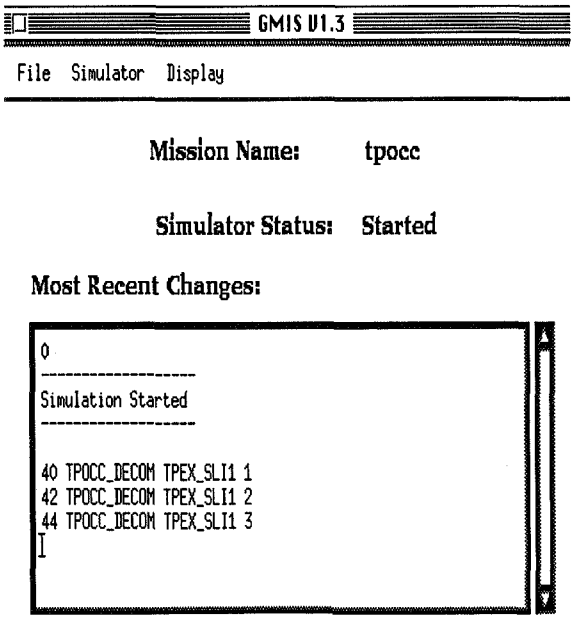


Figure 2: GMIS Main Panel

Many of the TSTOL constructs in the menu require multiple lines. For example, all the block structures have a starting and ending statement. In those cases, the statement builder will insert multiple lines and the user simply clicks within the block to add the statements that belong there. Moreover, some of the TSTOL constructs require parameters. That is indicated in the menu by a series of periods after the name. The Statement Builder helps with two of those: `Let` and `For`. When the user selects one of these constructs, the main area of the Statement Builder changes to display a fill-in form with the required parameters.

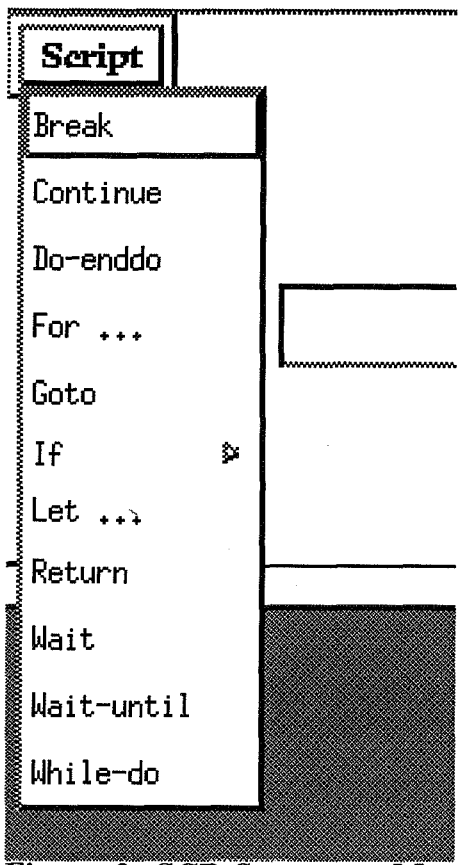


Figure 3: SCP Statement Menu

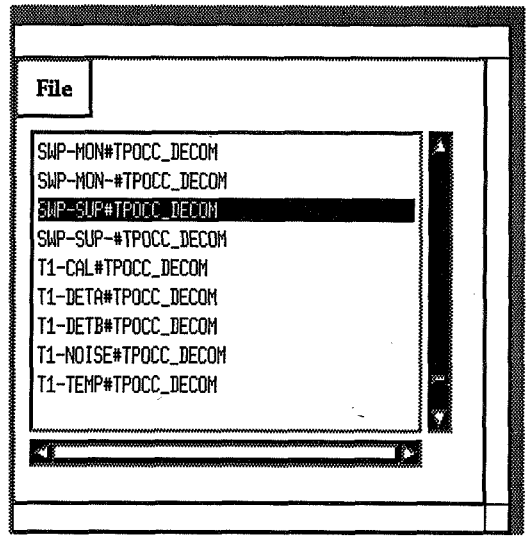


Figure 4: SCP Data Points Panel

When the script needs a TPOCC system variable name as a parameter, the user can type the name or click on the name in the Data Points panel. The Data Points panel lists all the current data server variables. For example, the user has just clicked `SWP-SUP` in Figure 4. Just before the name was selected, we started a `Let` statement in the Statement Builder window. When the name is selected, SCP copies it into the first entry field of the `Let` statement shown in Figure 5.

The statement that you construct in this window will be copied into the script when you click on the Apply option. In this case, we made an error - `foobar` is not a symbol TSTOL will recognize. We inserted this statement anyhow to produce error messages intentionally. When we execute a script, SCP brings up another window to show the dialog with the TSTOL process. Figure 6 shows the dialog for this script and the TSTOL error messages.

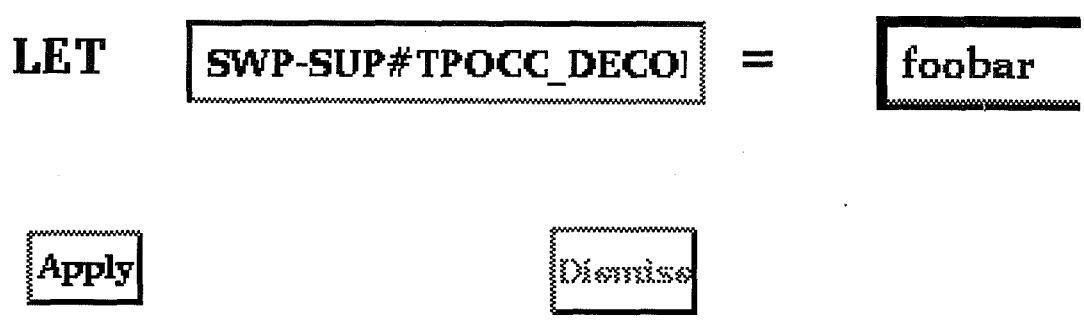


Figure 5: SCP Statement Builder - Parameter Form for "Let" Directive

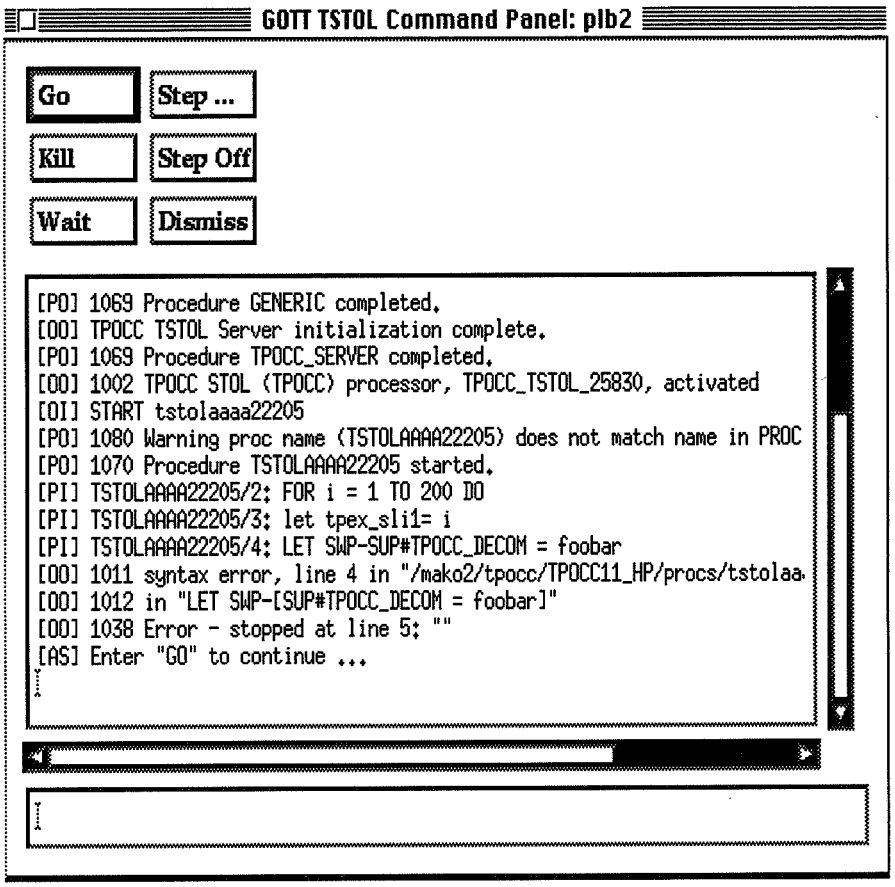


Figure 6: SCP - TSTOL Script Execution Panel

GMIS Software Design

The design for GMIS has several distinguishing features that are illustrated in Figure 7. A major design decision was to provide and maintain two copies of the system variables. The simulator modules read from one copy and write to a second. At the end of the time step, any variable that has changed is forwarded to the data server. At the same time, the first copy is updated from the second. The point of this feature is that it allows multiple modules to contribute to the next system state. The full state is written out at once after all the simulation modules have taken a turn.

The current version allows any number of internal, compiled simulation modules. In addition, the SCP can interpret TSTOL procedures and send new data values to the GMIS. The SCP is running asynchronously with the GMIS, but the internal modules are synchronized to a strict clock. The internal timing is maintained by examining the system clock, computing the amount of idle time between cycles, and then programming the Xt System software for the required delay. The basic timing cycle is illustrated in Figure 8.

The current version has the feature - or the implementation restriction, depending upon your viewpoint - that it is single threaded. Thus, the software cannot receive from the SCP when it is running a model, nor will it start its scheduled time step while a read operation is in progress. In future versions, we would like to be able overlap the data server communication with the other operations by providing two or more separate threads.

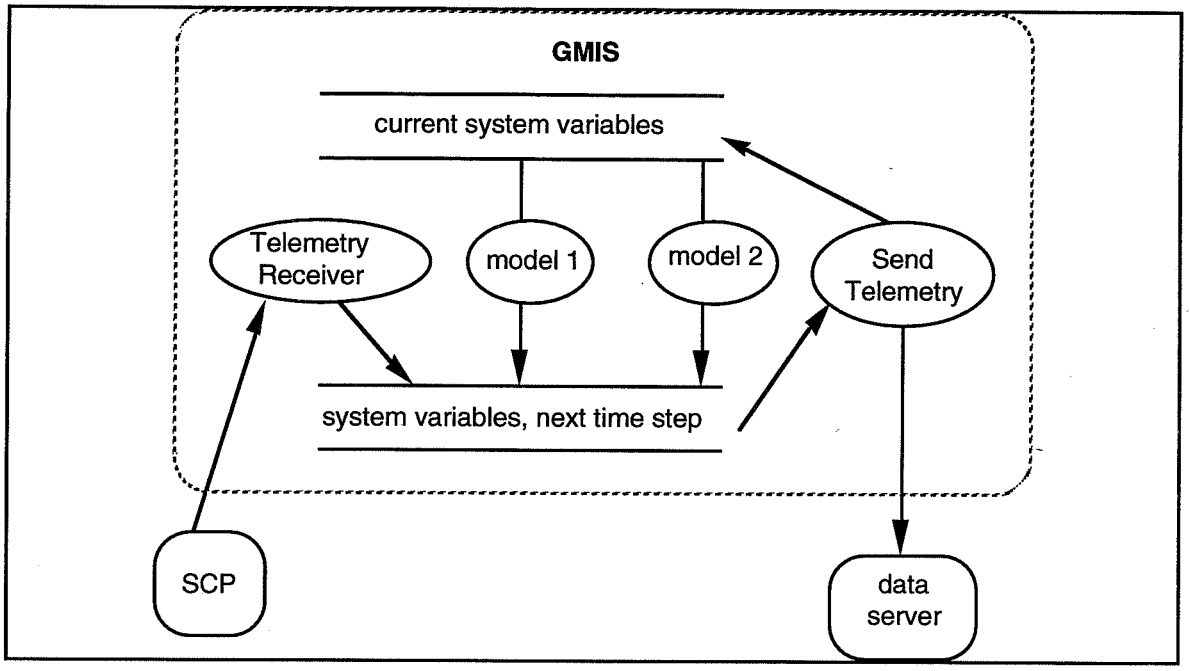


Figure 7 - The Data Flow in GMIS

A programmer who wants to extend GMIS with compiled modules must use C++ in the current version of GMIS. C++ classes provide most of the routine simulator functions such as timing and data communications. The functions are inherited by the simulator modules in the following way. To write a new module, the programmer defines it first as a class that inherits from the *ModelEngineInterface* class. Then the programmer makes a single instance of the class, which is the actual simulator module. When the instance is constructed, it will connect itself with the GMIS infrastructure automatically.

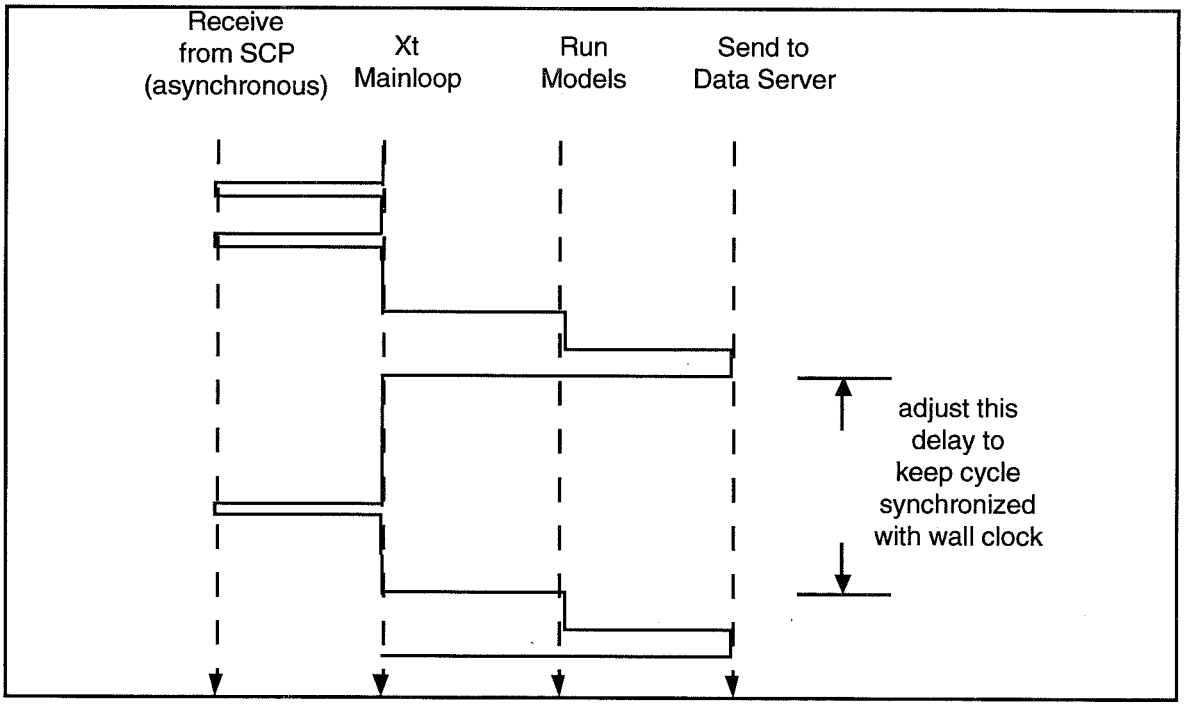


Figure 8 - GMIS Timing Diagram

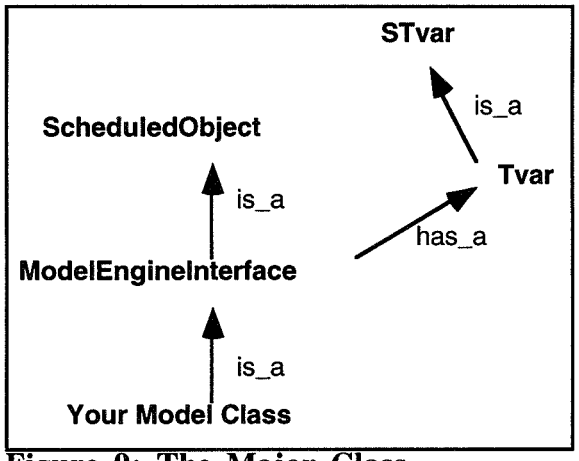


Figure 9: The Major Class Relationships in GMIS

The *ModelEngineInterface* class inherits its timing properties from a utility class, called *ScheduledObject*, and it acquires access to the system variables by a composition operation. In simple terms, it contains a pointer to a list of system variables. There is a full set of utility functions that come with the list as part of a class called *Tvar*. The GMIS effort borrowed the *Tvar's* from another project. Because the*Tvar's* didn't do everything we needed, we made *Tvar* a subclass of a new class *STvar* that added the new functions. This programming trick avoided any modifications to *Tvar*, initially anyhow. We will discuss software maintenance experiences later. The class relationships are summarized in the Figure 9.

Features Provided, Features Used

The GMIS simulator has been in operation for about 18 months. It has been used in experiments in the GOTT and by three other projects. The feedback has been surprising because the features that are used the most were not the most important during the development. Also, some of the most important features during development are used little, or tend to be in the way of use. Finally, we had to add features for two projects because there was a reasonable need that was not satisfied by the baseline version. As a result, there is currently a baseline version and two variations. The feature sets of these versions are summarized in the Table 1.

As we expected, the built-in connection to the data server has made GMIS a valued tool for testing TPOCC related software. The interpreted TSTOL modules have proven more valuable than expected. On the other hand, the class libraries have fared poorly in practice. The classes are not simple enough to encourage reuse, although a lack of familiarity with C++ may also be a factor. Moreover, the C++ class for data server access, `Tvar`, has proven very hard to maintain. Also, the `Tvar` class only handles asynchronous data and it proved impossible to extend it for synchronous data. Consequently, we had to write a special version of GMIS for a project that requires synchronous data simulation . Our conclusion is that it may not be a good idea to wrap a C++ class around a C library that is not well understood. This is certainly the case with `Tvar` and the TPOCC data services library.

A surprising requirement has been the need for simulations in which the simulation calculation is performed off-line and written to tape. The GMIS must then read the tape and supply the data at a steady pace controlled by the simulation clock. In principle, there is no reason that an off-line calculation could not be performed on-line. Indeed, one could keep the simulation software wherever it is developed and provide simulated data over a network on demand. In practice, software does not move freely and networks are not always connected to each other. For now, this requirement is a real one for NASA and it required a special version of GMIS.

Table 1: Summary of GMIS Features and Their Extent of Use

Feature	Version 1.0	Version 1.X
1. Asynchronous Data to Data Server	✪	
2. Synchronous Data to Data Server	○	✪
3. Local Copies of System Variables	●	
4. Compiled Simulation Modules, Static Binding	●	
5. Compiled Sim. Modules, Dynamic Binding	○	
6. Reusable C++ Classes	●	
7. TPOCC Data Server Classes	●	
8. Interpreted TSTOL Simulation Modules	✪	
9. Simulation Tape Playback	○	✪

Key:

- ○ feature not supported
- ● feature supported but not used
- ✪ feature supported and used extensively

Note:

Version 1.0 is the baseline version; Version 1.X is a notation for the several specialized variants.

The Future of GMIS/SCP

As the preceding summary shows, there are valuable features in GMIS that complement other simulation facilities. However, there is still room for improvement in the three qualities we deem important: convenience, extendibility, and speed.

In the area of convenience, it has long been our goal to have fast, compiled, simulation modules that we can start on demand while GMIS is running. These dynamically loaded modules would give the experimenters in the GOTT laboratory more flexibility in their tests. This feature should appear in the next version. The overall architecture of GMIS will then realize the design shown in Figure 10. Presently, all the compiled modules are linked statically, but that will change to dynamic loading in the next version.

In principle, the module that connects GMIS and SCP is just another producer module. Similarly, the module that connects to the data server is just another data consumer. In practice, it may be difficult to build a dynamically linked module without revising the extensive body of TPOCC code that is reused in these modules. Consequently, these modules will always be statically linked, but compilation options will determine whether the modules are present or not.

The extendibility of the current version is based on C++ classes that offer a variety of scheduling features as well as an encapsulated access to TPOCC data services. Only the simplest scheduling features are used, however, and the encapsulated access is often more of an obstacle than an advantage. For this reason, the next version of may be written in C, without classes.

The speed of GMIS should be improved considerably if we could overlap simulation calculations with system variable output. We plan to explore this possibility as DCE - Distributed Computing

Environment - is phased in. DCE has a built-in capability for multiple threads that should support simultaneous network communication and computation.

For the future, the GOTT laboratory is not limited to TPOCC control center software. In the past, the laboratory has hosted OASIS software and we are currently experimenting with software from Storm Technology, Inc. For this reason, the GMIS should be independent from a particular control center.

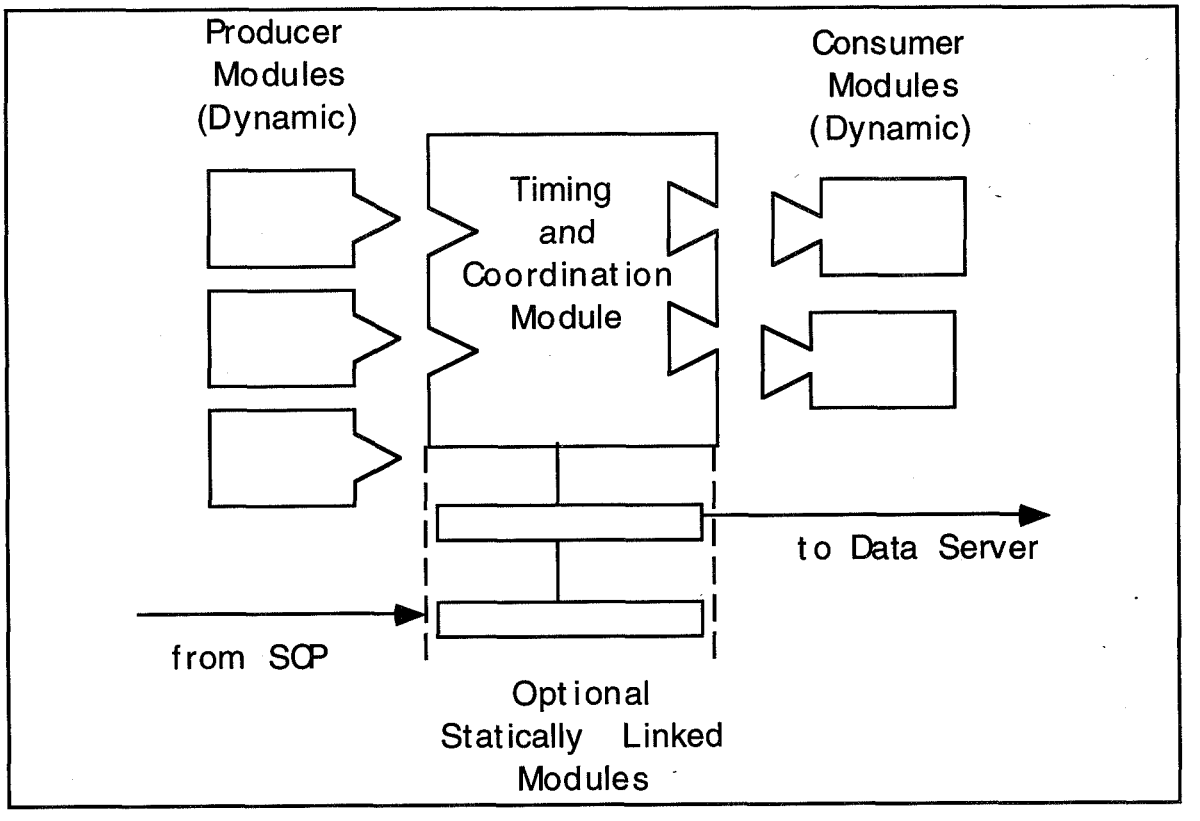


Figure 10: Overall Architecture for GMIS

Contact Information

Paul L. Baker
Global Science and Technology, Inc.
6411 Ivy Lane, Suite 610
Greenbelt, MD 20770

John Rosenberger
CTA Incorporated
6116 Executive Blvd.
Rockville, MD 20852

J. Michael Moore
Software Automation Systems Branch
Code 522
NASA Goddard Space Flight Center
Greenbelt MD 20771

Point of Contact

Paul L. Baker
Email: pbaker@gst.gsfc.nasa.gov
Tel. (301) 474-9696

A REUSABLE REAL-TIME OBJECT ORIENTED SPACECRAFT SIMULATOR

Eric Beser
AlliedSignal Inc.

Paper Not Available

111166 N95-17581

354261

TEST/SCORE/REPORT: SIMULATION TECHNIQUES FOR AUTOMATING THE TEST PROCESS

Barbara H. Hageman
Integral Systems, Inc.

Clayton B. Sigman
National Aeronautics and Space Administration
Goddard Space Flight Center

John T. Koslosky
National Aeronautics and Space Administration
Goddard Space Flight Center

ABSTRACT

A Test/Score/Report capability is currently being developed for the Transportable Payload Operations Control Center (TPOCC) Advanced Spacecraft Simulator (TASS) system which will automate testing of the Goddard Space Flight Center (GSFC) Payload Operations Control Center (POCC) and Mission Operations Center (MOC) software in three areas: telemetry decommutation, spacecraft command processing, and spacecraft memory load and dump processing. Automated computer control of the acceptance test process is one of the primary goals of a test team. With the proper simulation tools and user interface, the task of acceptance testing, regression testing, and repeatability of specific test procedures of a ground data system can be a simpler task. Ideally, the goal for complete automation would be to plug the operational deliverable into the simulator, press the start button, execute the test procedure, accumulate and analyze the data, score the results, and report the results to the test team along with a go/no go recommendation to the test team. In practice, this may not be possible because of inadequate test tools, pressures of schedule, limited resources, etc. Most tests are accomplished using a certain degree of automation and test procedures that are labor intensive. This paper discusses some simulation techniques that can improve the automation of the test process.

The TASS system *test*s the POCC/MOC software and provides a *score* based on the test results. The TASS system displays statistics on the success of the POCC/MOC system processing in each of the three areas as well as event messages pertaining to the Test/Score/Report processing. The TASS system also provides formatted *report*s documenting each step performed during the tests and the results of each step. A prototype of the Test/Score/Report capability is available and currently being used to test some POCC/MOC software deliveries. When this capability is fully operational it should greatly reduce the time necessary to test a POCC/MOC software delivery, as well as improve the quality of the test process.

1. INTRODUCTION

1.1 TASS Background

The Transportable Payload Operations Control Center (TPOCC) Advanced Spacecraft Simulator (TASS) system has been designed to support the development, test, and operational aspects of Payload Operations Control Center (POCC) and Mission Operations Center (MOC) software deliverables. TASS is designed to test the majority of POCC/MOC low-level requirements. The TASS system simulates spacecraft telemetry and command

PRECEDING PAGE BLANK NOT FILMED

PAGE 1102 INTENTIONALLY BLANK

functions. TASS takes advantage of the TPOCC architecture by using the backup POCC/MOC system configuration hardware for the simulator, or TASS can be separately hosted on a streamlined version of the POCC/MOC. This eliminates the need to schedule hardware or Nascom lines during various test configurations. In essence, the user has a simulator on call at all times.

TASS has the capability to simulate the Nascom link protocols required to support satellites and generate simulated spacecraft telemetry streams using the POCC's/MOC's operational data base (ODB). TASS validates spacecraft commands and alters the real-time telemetry stream in response to those commands. The user can alter the telemetry stream either by data base mnemonic or by specifying individual bits in the telemetry frame or packet. Similar telemetry display pages at both the simulator workstation and the POCC/MOC workstation help identify telemetry processing irregularities. As part of the system design, software hooks are available so more complexity can be added by providing various dynamic models for the telemetry generating function.

In the POCC/MOC test environment, the TASS system provides a means for saving and restoring predefined test scenarios and results, telemetry stream contents, and data structures to allow the user to accurately repeat specific tests, retest with known data, or continue testing from a given point in the test scenario. These features allow the user to perform regression tests on new software deliverables in the shortest possible time.

TASS records all received Nascom blocks and all received spacecraft commands in history files that can be viewed for detailed analysis through the use of an offline utility program. All system events, errors, operator input, procedure input recorded in the event log; and spacecraft memory images that are saved can be viewed by using the offline utility programs. After completing the test, the user generates test reports using the report generation subsystem. These reports can later be used to evaluate the test results during the analysis process.

Unique implementations of spacecraft memory load and dump capabilities are provided as well as an NCC communications protocol when TDRSS support is required.

1.2 TASS System Design

A typical POCC/MOC system string is used to host the TASS software. The hardware configuration to support TASS consists of two computers connected by Ethernet and associated peripherals as shown in Figure 1. These computers are a real-time front-end computer or processor (FEP) in a Versa Module European (VME) bus enclosure and a general-purpose computer or workstation. The real-time FEP is used to process spacecraft commands and to build and transmit telemetry streams. The Hewlett-Packard HP 9000-715 workstation allows the user to configure, control, and monitor the FEP from one or more user terminals.

TASS makes extensive use of the same TPOCC reusable software that the POCC/MOC developers use, mainly the user interface (display and TPOCC Systems Test and Operations Language (TSTOL)) and the Nascom interface. The display system is based on X Windows and fully adheres to the industry-standard OSF/Motif principles. TSTOL is the user script language which is used to control the TPOCC application system (either the POCC/MOC system or the TASS system). TSTOL is also used to develop operational scenarios and test procedures. Presently, TPOCC reusable software comprises approximately 78% of the TASS system. Another 16% of TASS is reusable from mission to mission, such that only about 6% of TASS needs to be newly developed with each added POCC/MOC mission.

1.3 Control Center Configuration

Because of the methodology chosen for the overall ground system design, no special equipment or system configuration is required for TASS. TASS uses the POCC/MOC backup system string and communicates with the primary POCC/MOC system string thru the local TPOCC switch. In a test configuration, the TASS input/output data flow at the switch interface looks like the Nascom interface to the primary POCC/MOC system string. This

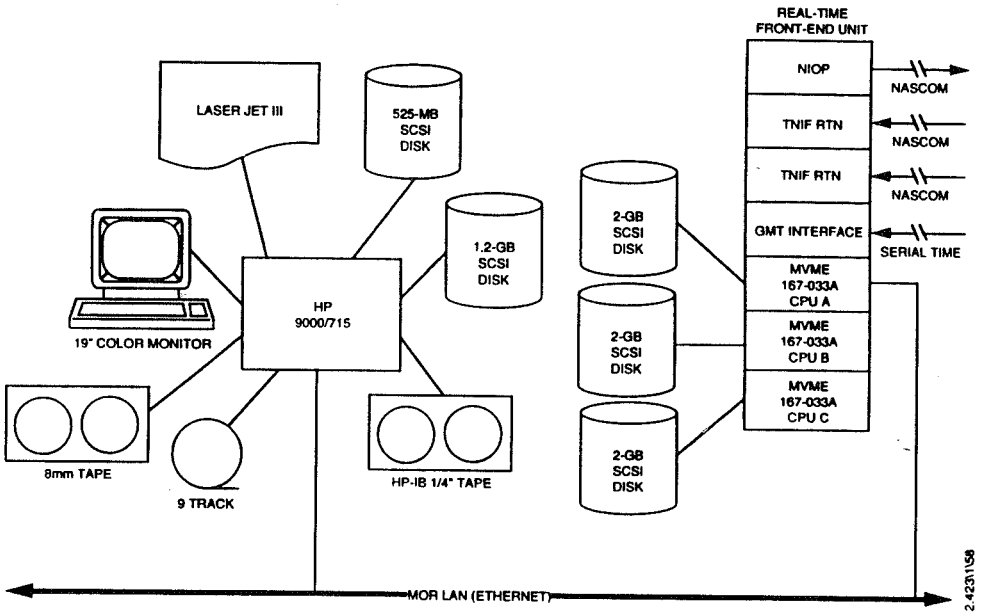


Figure 1. TASS Hardware Configuration

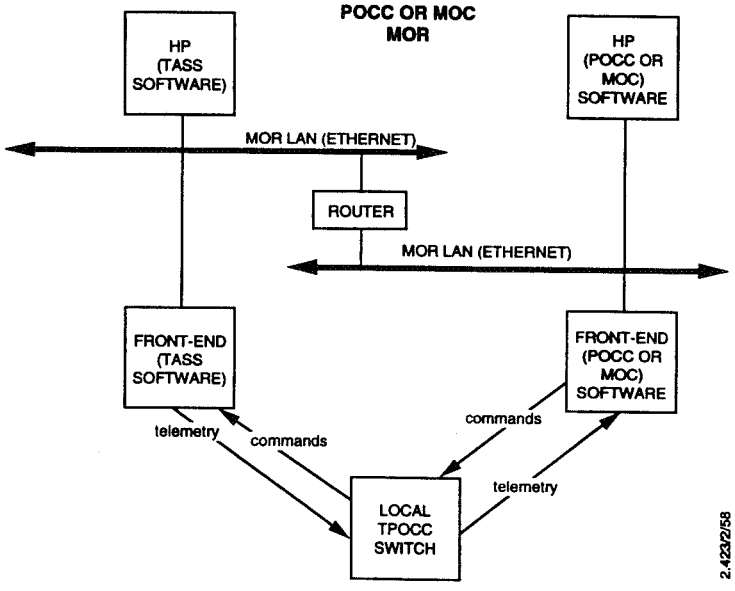


Figure 2. Control Center Configuration

architecture is shown in Figure 2.

The TASS system accepts spacecraft commands from the POCC/MOC and transmits telemetry to the POCC/MOC via Nascom connections on both FEPs. The workstations show displays generated by the TASS and POCC/MOC system.

2. GROUND DATA SYSTEM TESTING

2.1 Software Delivery Test Process

GSFC Control Center Systems Branch is responsible for testing the software deliverables for the POCC/MOC systems in the TPOCC environment. TPOCC based POCCs/MOCs support the WIND, POLAR, SAMPEX, FAST, SWAS, SOHO, XTE, TRMM, and ACE missions. Testing of these POCC/MOC systems consists of unit testing, integration testing, and finally acceptance testing.

Unit testing is performed throughout the software implementation phase by the POCC/MOC developers. Unit testing is completed prior to delivery of the unit for system integration.

Integration testing is performed before delivery to the test team. This testing verifies integration of TPOCC generic software and POCC/MOC unit software into the POCC/MOC system and is performed by an integration manager who is supported by the development team.

Acceptance testing is performed by the test team before delivery to GSFC according to a comprehensive test plan and procedures. This testing verifies the functional and performance requirements and is completed prior to delivery of the system to operations/user community.

2.2 Automating the Test Process

To achieve the goal of automating the test process, several test methodologies have been prototyped. The most promising concept is Test/Score/Report. TASS and POCC/MOC simularities in system architecture, user interface, script language, and project data base files are some elements that support this system concept approach to automate the test process.

3. TEST/SCORE/REPORT

3.1 Automated Testing in Three Areas

The Test/Score/Report capability currently being developed will automate testing of the POCC/MOC software in three areas: telemetry decommutation, spacecraft command processing, and spacecraft memory load and dump processing. TASS takes advantage of the ground system attributes in designing the Test/Score/Report capabilities. Both the TASS and the POCC/MOC systems are using some of the same reusable building blocks of TPOCC software and running on the same hardware architecture. By using this approach, TASS can easily add features which enhance the automated test process.

Figure 3 shows the data and control flows between the two systems. The TASS system simply establishes a socket connection with the POCC/MOC system in order to make requests for data and to receive the data. This connection is transparent to the POCC/MOC system and requires no new software be written on the POCC/MOC side. The TASS system also reads the POCC's/MOC's system variable dump file and the ground image file which both reside on the workstation's disk. The system variable dump file contains all of the telemetry parameters located in the operational data base as well as counters and status information. This file is needed for initialization purposes before requests for data can be made. The ground image file is used to validate spacecraft memory load and dump processing.

Telemetry decommutation is tested in two ways. The first way is by comparing the values of telemetry parameters decommutated by the POCC/MOC against the telemetry parameters commutated by TASS. Ideally, the decommutated values should match the commutated values. The second way is by comparing the limit specifications previously set with the status words of decommutated telemetry parameters. Every telemetry parameter located in the operational data base is automat-

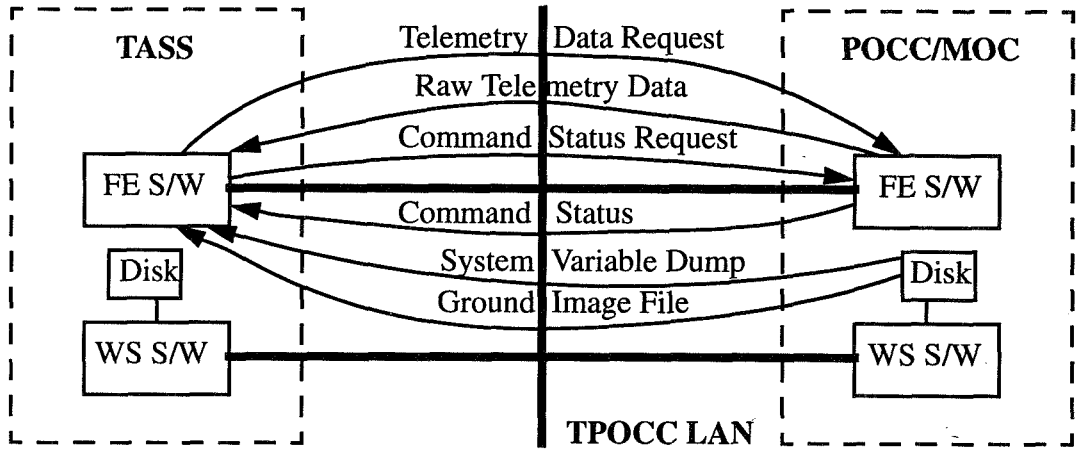


Figure 3. Data/Control Flow Between TASS and the POCC/MOC

ically checked. Each discrepancy is displayed as an event message which gives the value of the decommutated telemetry parameter and the value of the commutated telemetry parameter. Summary event messages for the two telemetry decommutation tests state the total number of telemetry parameters which decommutated correctly, the total number of telemetry parameters checked, and the percentage of which were correctly decommutated.

For spacecraft commanding, the test process validates the various fields in the Nascom block header for all spacecraft command blocks received by TASS and validates the individual spacecraft commands received in valid command blocks. TASS also checks whether the POCC/MOC verified the commands after they were executed by TASS. This is accomplished by making data requests to the POCC/MOC for values of command status parameters and counters. Summary event messages are displayed which give the number of valid command blocks, the number of valid commands, the number of commands verified by the POCC/MOC, and the percentages for each of the above tests.

The spacecraft memory load and dump processing is tested by comparing the spacecraft image maintained by TASS against the ground image file maintained by the POCC/MOC. This test is performed after memory load data is sent to TASS from the POCC/MOC via a spacecraft command and after TASS transmits a memory dump to the POCC/MOC via the telemetry stream. Ideally, the memory values maintained by TASS should match values in the POCC/MOC ground image file. A summary event message informs the user of the number of bytes that miscompared, the total number of bytes in the spacecraft image, and the percentage of bytes which had the same value.

3.2 Scoring the Ground Data System

The initial scoring method will be in terms of percentages and raw counts. More experience in testing and interpreting the test results will be required to develop a better scoring methodology.

3.3 Reporting the Test Results

The TASS system displays statistics on the success of the POCC/MOC in each of the three areas as well as event messages pertaining to the Test/Score/Report processing. The Test/Score/Report display page shown in Figure 4 is broken up into two sections. The top section gives the summary counts and scores for the various tests. The bottom section is a scrolling region which displays all event messages generated during Test/Score/Report processing.

The TASS system also provides formatted reports documenting each step performed during the tests and the results of each step. The user can issue reports for each type of test (i.e., telemetry, limits, command, memory loads/dumps). Report options include showing all

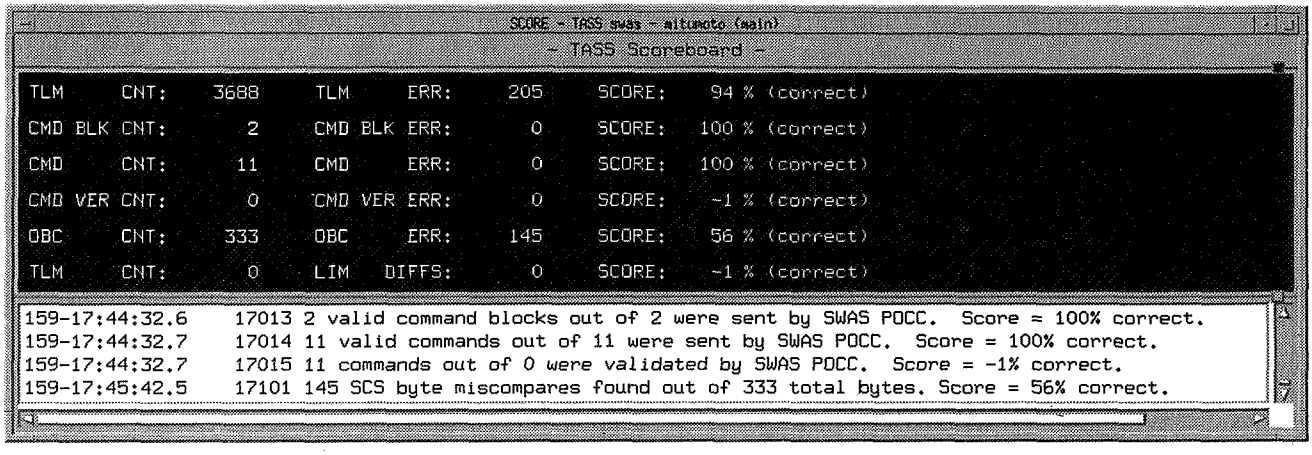


Figure 4. Test/Score/Report Display Page

parameters tested or just those parameters which miscompared or were in error. For telemetry tests, the report shows the telemetry mnemonic, the commutated value, and the decommutated value. For limit tests, the report shows the telemetry mnemonic, the commutated value, the limit specification and the status word of the decommutated parameter, and the decommutated value. For command tests, the report shows numbers of: Nascom block header errors, command errors, and commands verified. Finally, for memory load/dump tests, the report shows the memory address, TASS's spacecraft image value, and the POCC's/MOC's ground image value.

4. PRESENT STATUS

A prototype of the Test/Score/Report capability is available and currently being used to test the WIND, POLAR, SAMPEX, FAST, SWAS, and SOHO POCC software deliveries and the XTE MOC software delivery. This prototype includes the "test" and "score" features described in this paper. The "report" features (other then the Test/Score/Report display page) are currently being developed by the TASS development team and are planned to be released by early next year. The TASS development team is actively working with the POCC/MOC test team and software developers to obtain feedback on the Test/Score/Report prototype.

5. FUTURE DIRECTIONS

Basically we have completed our prototyping stage of this project. We have been successful in implementing this methodology in several projects as mentioned in the previous section. Future objectives are to automate more of the test process by: 1) including additional subsystem testing such as attitude, events, packet extraction, history, NCC, and database testing; 2) improving the scoring methodology; and 3) providing more functionality and options for the report process.

REFERENCES

NASA GSFC, (June 1994). *Transportable Payload Operations Control Center (TPOCC) Advanced Spacecraft Simulator (TASS) System Requirements Document (Revision 4).*

NASA GSFC, (May 1994). *Transportable Payload Operations Control Center (TPOCC) Advanced Spacecraft Simulator (TASS) System User's Guide for Release 7.*

NASA GSFC, (May 1994). *Transportable Payload Operations Control Center (TPOCC) Advanced Spacecraft Simulator (TASS) Detailed Design Specification for Release 7.*

NASA GSFC, (February 1994). *Transportable Payload Operations Control Center (TPOCC) Detailed Design Specification for Release 10.*

Acknowledgements -- We wish to extend special thanks to the following personnel for ideas and review of the report: Carroll Dudley (NASA GSFC), Luan Luu (Integral Systems, Inc.), Nancy McCluer (Integral Systems, Inc.), and Darlene Riddle (Integral Systems, Inc.).

11167 N95-17582

354263

SPACECRAFT DATA SIMULATOR FOR THE TEST OF LEVEL ZERO PROCESSING SYSTEMS

p.10

Jeff Shi, Julie Gordon
RMS Technologies, Inc.
Code 520.9

Chandru Mirchandani, Diem Nguyen
Loral AeroSys
Code 521

NASA, Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

The Microelectronic Systems Branch (MSB) at Goddard Space Flight Center (GSFC) has developed a Spacecraft Data Simulator (SDS) to support the development, test, and verification of prototype and production Level Zero Processing (LZP) systems. Based on a disk array system, the SDS is capable of generating large test data sets up to 5 Gigabytes and outputting serial test data at rates up to 80 Mbps. The SDS supports data formats including NASA Communication (Nascom) blocks, Consultative Committee for Space Data Systems (CCSDS) Version 1 & 2 frames and packets, and all the Advanced Orbiting Systems (AOS) services. The capability to simulate both sequential and non-sequential time-ordered downlink data streams with errors and gaps is crucial to test LZP systems. This paper describes the system architecture, hardware and software designs, and test data designs. Examples of test data designs are included to illustrate the application of the SDS.

1. INTRODUCTION

Simulation of spacecraft data is a basic function of any space and ground data system. Over the years, many simulation systems have been developed to support spacecraft and ground system Integration and Test (I&T). However, very few are capable of testing LZP systems that are based on CCSDS recommended data formats [1][2] and require large unique test data sets with complex data scenarios.

In 1992, the Microelectronic Systems Branch (MSB) at GSFC developed a Spacecraft Data Simulator (SDS) to support it's Very Large Scale Integration (VLSI) LZP system prototype phase one (VLSI LZP-1) development [3]. The VLSI LZP-1 is capable of performing LZP functions for CCSDS packet telemetry at rates up to 20 Megabits per second (Mbps). In order to test this system, it was necessary to simulate realistic streams of spacecraft data, including a variety of errors and data gaps. The SDS was used to simulate data streams of 750 Mbytes each (i.e., 5-minute sessions at 20 Mbps) with all valid timecodes and sequence counts. To emulate onboard tape recorders, reversed data streams were generated for playback sessions. To test the overlap deletion function, the forward real-time sessions and reversed playback sessions were simulated with regions of overlap at the beginning and end of sessions. Many types of errors were inserted into selected frames and packets in both overlap and non-overlap regions to fully exercise system capability of handling errors.

The SDS was significantly enhanced through the development and deployment of Fast Auroral Snapshot Explorer (FAST) Packet Processing System (PPS), that is based on the VLSI LZP-1 architecture [4]. The use of Solid-State Recorders (SSR) onboard FAST created many

PRECEDING PAGE BLANK NOT FILMED

PAGE 1110 INTENTIONALLY BLANK

complicated data scenarios never before encountered in conventional tape recorder-based missions [5]. New functions were added to the SDS to simulate the scenarios, such as, downloading thousands of data fragments in a single session, interleaving real-time and playback data with identical packets being present on different Virtual Channels (VC), overlap occurring anywhere in a data stream, and packet sampling that resulted in non-contiguous packet sequence counts. These enhancements enabled the SDS to be used successfully in the FAST PPS system development, integration, and acceptance testing.

Early in 1994, the SDS was upgraded further to support the development of VLSI LZP prototype phase two (VLSI LZP-2) [6]. Simulation software was modified to support CCSDS AOS data formats, and key hardware components were upgraded increasing the maximum data output rate from 25 to 80 Mbps.

2. SYSTEM OVERVIEW

The Spacecraft Data Simulator consists of a hardware subsystem and a software subsystem. The hardware subsystem provides processing power, data storage, a network interface, and a telemetry interface. The software subsystem takes in user specifications and generates simulated CCSDS telemetry data accordingly.

The hardware subsystem, shown in Figure 1, contains a simulation processor and a 5.5 Gbytes disk farm. The simulation processor is contained in a Versa Module Eurocard (VME) equipment rack; it consists of a Master Controller card, an Ethernet Interface card, a Disk Controller card, a Memory card, and a MSB-developed Data Generator (DG) card. All cards except the DG card are Commercial Off-the-Shelf (COTS) components that provide system base functions for data processing, buffering, and network interfacing. The DG interfaces with the disk farm and is used to store simulated telemetry data in the disk farm during offline data generation. During a test session, the DG retrieves the data from the disk farm, serializes it and outputs it through an RS-422 interface or an Emitter Coupled Logic (ECL) interface. The data rate is user-selectable from 0 to 80 MHz. The disk farm parallel architecture enables data transfer at rates up to 128 Mbps. The disk farm capacity is configurable from 5.5 to 40 Gbytes.

The software subsystem consists of two major components: the Test Pattern Generator (TPGEN) and the Large Volume Data Generator (LVGEN). TPGEN is a menu-driven software package that can generate small sets of Nascom blocks, and conventional and AOS CCSDS frames and packets for up to 32 VCs and 256 Application Processes (AP). Packet and frame placement is user-defined. The data can be either Cyclic Redundancy Checked (CRC) or Reed-Solomon encoded, and many types of errors can be inserted in the data. TPGEN has been used to support many missions including the Topographic Explorer (TOPEX); Solar Anomalous and Magnetospheric Particle Explorer (SAMPEX); and FAST.

LVGEN is a script-based package. It uses TPGEN to configure "base sets" of frames or blocks when the ratio and placement of packets from all sources are defined. It then switches among the "base sets" while generating the test data set, with all timecodes and sequence counts increment correctly during switches. The only restriction is that all "base sets" must share the same source list.

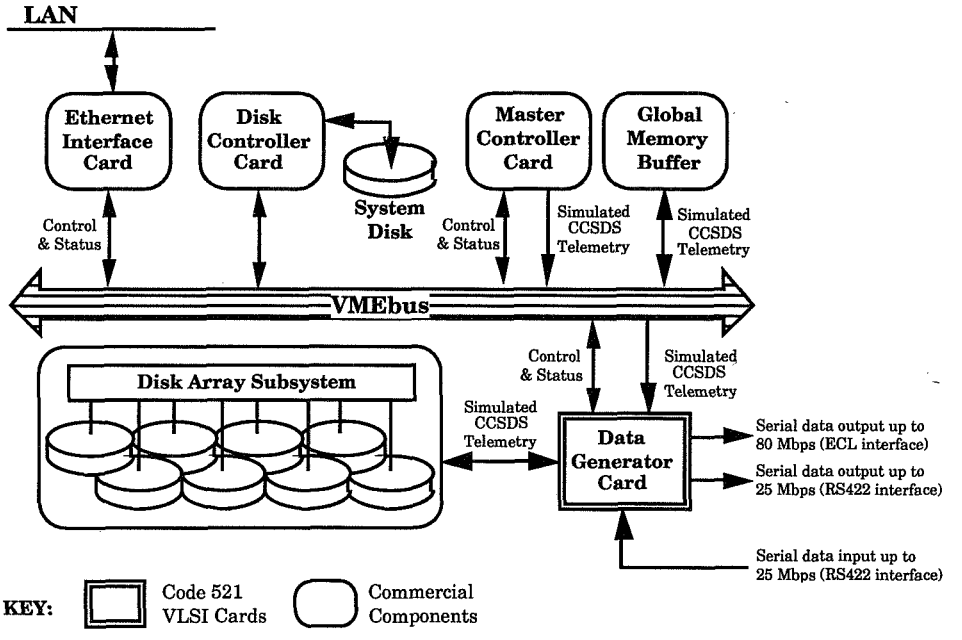


Figure 1. System Block Diagram

Figure 2 illustrates the information and process flow in SDS data set generation. Spacecraft needs are interpreted as requirements and operational scenarios. Requirements are used to design the data set and to specify the errors to be injected. The data set design includes simulating the expected order, frequency, and content of the data from the spacecraft. This step produces external files defining the complexity of the data, scripts encompassing the order and rules for error injection and data generation, and the TPGEN catalogs (configuration files) defining the repeatable sets of frames. LVGEN uses the products from the data design and error injection stage to generate the data. The user interface downloads the test setup, and activates the DG to output the simulated data stream using the LVGEN-generated data.

When a "base set" must contain a larger number of frames than can be defined in a single TPGEN configuration file, it is broken down into "subsets," each with a separate configuration file. Different versions of the same "subset" may be used to simulate the forward playback scenario where identical packets from one source can be transmitted on more than one VC. The start count for each file created by LVGEN defines the regions of overlap between the sessions.

For "complex data," where the transmission of packets out of time order by a spacecraft must be simulated, LVGEN fills the sequence count and timecodes for each packet from external files that are generated by a Packet Header Definition (PHD) utility based on the test scenarios. The simulated science or engineering source data can be simple repeated patterns, or can be read from files that may contain real spacecraft data, still images, or other interesting data patterns.

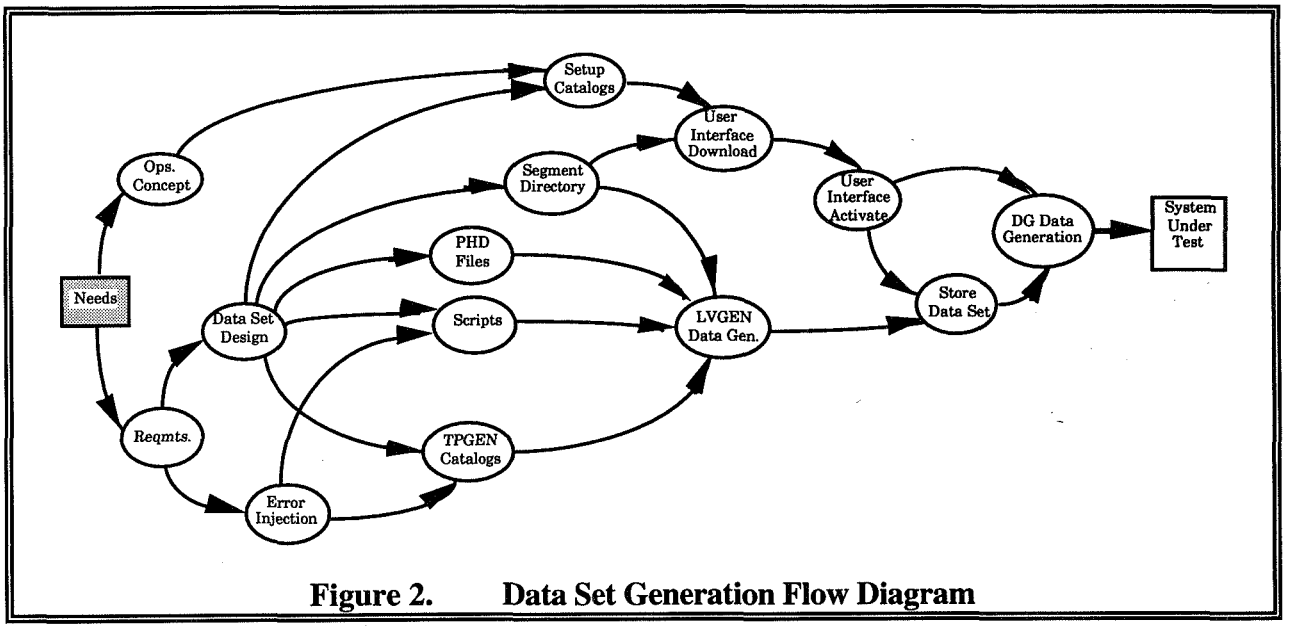


Figure 2. Data Set Generation Flow Diagram

3. SDS APPLICATION

The SDS was used to functionally test the FAST PPS system through system development, integration, and acceptance testing. This section will describe the strategy used to test the FAST PPS which included exercising the functionality and performance of the PPS. The strategy was to initially develop the minimum number of data sets which would base-line the functionality of the FAST PPS; these data sets will have varying degrees of complexity. The next step was to place errors in the initially-generated data sets that would cover the range of errors and performance metrics to fully stress the PPS.

3.1 CALIBRATING DATA SET GENERATION

The first data set that was generated was referred to as 'clean' or 'calibrating,' i.e., with no errors. It consisted of a single session of block-encoded frame data, with 564,131 blocks containing 300,600 telemetry frames that simulated a 30-minute session at 1.5 Mbps.

The calibrating data set simulated the expected proportions of the VCs as closely as possible. Within each VC, relative proportions of Application Process Identifiers (APID) were configured as specified in the FAST Data Management Plan. In order to expedite simulation, frames of the same VC occurred in groups of not more than 1280 frames within the telemetry stream. The data set consisted of playback and realtime interleaved data, bound on both sides by VC 7 fill data (see Figure 3).

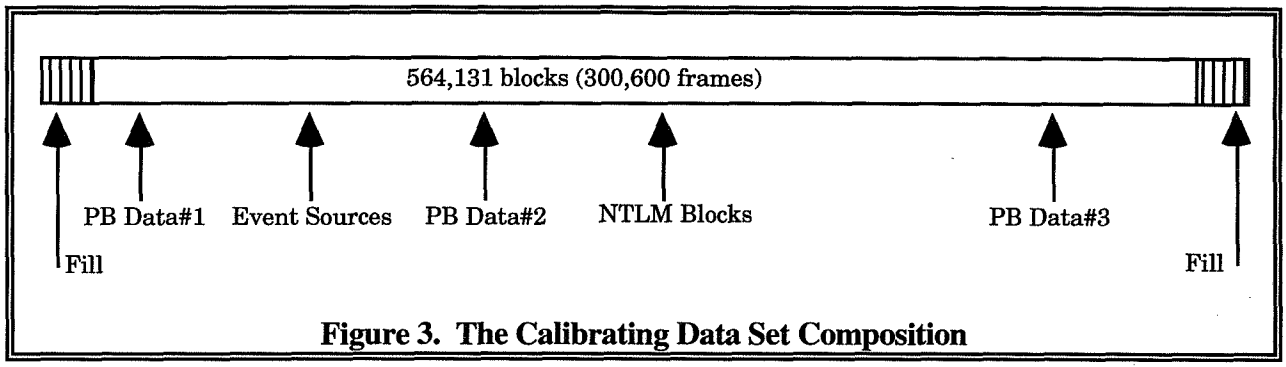


Figure 3. The Calibrating Data Set Composition

The playback data, VC 1, was dumped three times during the session at a rate of 1 out of every 5 frames downloaded. During the time VC 1 data was output, VC 2, 3, and 4 data continued to be read out in a similar order as when no VC 1 data was being dumped, although the contents of each VC were slightly different due to simulation limitations. The first VC 1 dump occurred at the beginning of the session; the second dump occurred in the middle of the session; and the last dump was a partial dump that occurred near the end of the session. VC 0 was output at a rate of 1 frame every 8 seconds (approximately 1 frame every 1394 frames at 1.5 Mbps). VCs 2, 3, and 4 were distributed throughout the scenario in alternating groups of each VC. For this data set, it was decided that VCs 2, 3 and 4 would have relative densities 8:0.125:1. One APID had all its packets in VC 3. Several other APIDs had packets in both VC 3 and 4; these APIDs were in VC 4 most of the time but switched to having small numbers of packets in VC 3 no more than 20 times during the course of the session.

A group of "non-telemetry" blocks was introduced in the middle of the data stream, to test telemetry/non-telemetry filtering. These blocks had different destination codes or message type codes.

The calibrating data set was generated by LVGEN using TPGEN catalogs, PHD files and user developed scripts. Variances in the data content were produced by switching between base sets while generating the data. Although the data was Nascom block encoded, the base sets were defined by the number of frames and the content of the frame data per base set.

3.2 DATA SET COMPLEXITY

Different types of complex data containing multiple segments were simulated to fully test the capability of the FAST PPS. "Segment" is a concept referring to a group of packets of the same source which has a consistent time order, either forward or reversed. Data is "simple" if there is only a single segment received per session per source; data is "complex" if there are multiple segments received from a source during a session. For the FAST spacecraft, there are three ways in which complex data may be transmitted; each of these data types were simulated in the calibrating data set.

For all sources in VC 2, the FAST spacecraft uses a "sampling" storage algorithm to write data into the onboard SSR. Sensor (source) data is compared against a preset threshold. If the data value is less than the threshold, the data is filled sequentially into a small buffer storage area on the SSR. If the data value meets or exceeds the preset threshold, a sample of the subsequent sensor data is sequentially stored in a partition within a larger buffer storage area on the SSR. After the larger buffer storage area is full, samples are still taken and compared to all the stored samples

according to pre-defined criteria. If a new sample is better, it will overwrite an old sample with the best value, that may be anywhere from the 1st partition to the Nth partition. As more observations are made and the sensor value fluctuates, this scheme will result in segmented data in the SSR. When the SSR is dumped sequentially from low to high address, the PPS will receive and reassemble data segments that are completely out of time order.

In LVGEN, the FAST sampling process was simulated by placing the first packet of the defined "sample" at the start of the second segment; and the remaining packets were then used to fill the second, the first, and the third segments, in that order. This means, among other things, that if the playback list, defined by external files, used by LVGEN specified adjacent forward-time-order samples, the last packet of the third segment of the first sample was continuous with the first packet of the first segment of the second sample. The proportion of the first segment to the second segment for each sample was controllable within LVGEN. In the calibrating data set, VC 2 contained "complex data," with 8 simulated "samples" received in a mixed time order.

The second source of multiple segments in FAST data was the VC 1 engineering playback data. During a pass, the FAST spacecraft may re-transmit stored engineering data several times, resulting in several wholly or partially identical groups of packets which must be overlap processed to delete redundant data. Multiple transmissions of playback engineering data and the mixture of realtime and playback data (albeit on separate VCs) was also simulated in the calibrating data set. The third type of FAST data segments result from the splitting of survey science data between VC 3 and 4. On the FAST PPS, these two streams, which contain no overlapping data, will be processed separately and merged, similar to the way VC 0 and 1 data were processed.

The calibrating data set was a sequential data set, using a sequence count increment of 1. Some complexity was introduced to base-line the functionality of the FAST PPS. In this data set, only some VC 2 APIDs were fully simulated as sampled. The remaining sources were "simple," i.e., a single segment per session. The order of groups of packets within the data stream was specified by means of listing the playback order of samples for VC 2 sources in the calibrating data set. Two additional calibrating data sets were designed with greater complexity and used to further stress the capability of the FAST PPS. They were also used to test non-sequential packet data processing.

3.3 DATA SET ERRORS AND THEIR IMPLEMENTATION

The SDS was also used to simulate data streams with errors. Three scenarios were needed to fully test the FAST PPS: (1) Introduce various types of errors, (2) Introduce a large number of errors, 5% and 10%, and (3) Introduce errors in sequential and non-sequential data, both partially and completely overlapped, to exercise the capability to segment and locate questionable, or 'fuzzy' packets.

The first scenario introduced errors in the calibrating data set to simulate realistic cause and effects of data and transmission errors. The errors that can be generated include: (1) Noise in the transmission from space-to-ground, and (2) Noise in the ground-to-GSFC transmission. The errors that can be generated, and the resultant effects at block, frame, and packet levels are shown in Table 1.

Table 1. Errors, Results, and Statistics

ERRORS	RESULTS	STATISTICS
Block Synchronization Pattern Errors	Dropped Blocks, Missing Frames, Missing Packets	Number of Blocks, Number of Block Synchronization Errors, Number of Block Sequence Errors, Number of Frames, Number of [Missing, Back to Search, Lock, Check] Frames, Number of Packets, Number of Missing Packets, Number of Packet Gaps
Block & Frame CRC Errors	Tagged Blocks, Bad Frames, Bad Packets	Number of Block Polynomial Errors, Number of Frame CRC Errors, Number of Packet Errors
Block CRC Errors	Tagged Blocks, Good Frames, Good Packets	Number of Block Polynomial Errors, Number of Frames from CRC Error Blocks, Number of Packets from CRC Error Blocks
Frame Synchronization Errors	Missing Frames, Missing Packets	Number of Frames, Number of [Missing, Back to Search, Lock, Check] Frames, Number of Packets, Number of Missing Packets, Number of Packet Gaps
Frame First Header Pointer Errors	Rejected Frames, Missing Packets	Number of FHP Errors, Number of Missing Frames, Number of Missing Packets, Number of Packet Gaps
Frame CRC Errors	Bad Frames, Bad Packets	Number of Frame CRC Errors, Number of Packet Errors
Frame Bit Slip Errors	Bad Frames, Good Packets	Number of Bit Slips, Number of Missing Frames, Number of Missing Packets, Number of Packet Gaps
Packet Length Errors	Bad Packets	Number of Missing Packets, Number of Packet Length Errors, Number of Packet Sequence Errors

To introduce these errors, TPGEN catalogs for the calibrating data set were modified to include the errors. To ensure the number of times an error is to be repeated and its location in blocks, frames, and packets, base sets were split into smaller sets and the subsequent modified catalogs were used in conjunction with LVGEN to generate data with induced errors. For example, to generate an error in the 10th frame of a 40 frame base set within a specific repetition of the base set, (i.e., frame location,) it was necessary to break the base set into two smaller base sets of 9 and 31 frames, respectively. The sequence of the frames is still maintained, but to inject the error, the LVGEN script will treat these two smaller base sets separately. For example, a data set requires repeating a particular base set (BS3) 200 times, within which the 102nd repetition will have some error introduced in the 10th frame. BS3 is split into BS3a and BS3b, and the script will specify repeating BS3 101 times, repeat BS3a once, BS3b once, and subsequently repeat BS3 98 times.

Three types of simulated block errors were adequate to test the requirements and statistics at block, frame, and packet levels; Block Synchronization Error - by a flipped or incorrect bit in the block synchronization pattern; Block Polynomial Error with Frame CRC Error - by an error of one or more bits in the Nascom Block; and Block Polynomial Error with No Frame CRC Error - by flipped or incorrect errors in the Nascom block header or trailer.

Four types of simulated frame errors were adequate to test the requirements and test statistics at frame and packet levels; Frame Synchronization Error - flipped or incorrect bit in the frame synchronization pattern; Frame CRC Error without Block Polynomial Error - by one or more incorrect bit in the telemetry data field; Frame Bit Slip Error - by dropped or extra bit(s) in the frame; and Frame First Header Pointer Error - error in the first header field of the frame.

Packet length error was simulated to test the system requirements for detection and qualification of packet level errors. It was simulated by flipping the last significant bit in the packet length field specified in the packet header of the selected packet.

The next scenario tested the performance of the system and evaluated its capability to process data with large quantities of errors. Two data sets with 'High Volume Errors' were designed to test these capabilities. They consisted of the FAST VC 2 sources only, with frames and packets similar to previous data sets, to simplify generation and predicted results. High-rate sampling was not simulated; all sources were a single segment per session. Frame CRC, First Header Pointer Errors, Frame Synchronization, Block Synchronization, and Packet Length errors were distributed throughout the data set to bring the total percentage of packets with errors or gaps to 5% and 10%, respectively.

The final scenario tested the capability of the system to process the maximum number of sources with non-sequential and questionable packet sequence numbers. A data set called SRC200 was designed to test the requirement that the FAST PPS was able to process up to 200 sources. It was not intended to otherwise simulate the expected FAST telemetry format, neither in APIDs nor in packet sizes. The opportunity presented by this non-realistic scenario allowed the SRC200 data set to contain several "complex" sources with a full range of segment processing tests, including non-sequential and sequential overlap cases. This data set had no induced errors or gaps, but was used as the template for a later test set in which the complex data was used to test segment processing with fuzzy packets and non-sequential segment processing with errors and gaps. The 200 sources were generated by distributing 128 APIDs over VCs 0-6. The frame and packet format for SRC200 was the same as for previous data sets. Packet size was specified as 1054 bytes for all packets, resulting in one packet per frame for all sources.

4. CURRENT AND FUTURE DEVELOPMENT

During 1994, the Spacecraft Data Simulator was upgraded to support development and I&T of VLSI LZP prototype phase 2 that performs LZP functions at rates up to 50 Mbps. The Data Generator card was outfitted with a new mezzanine that increases output data rates from 25 to 80 Mbps. The TPGEN and LVGEN data generation packages have been modified to simulate CCSDS AOS data for up to 32 VCs and 256 APIDs. To test the system's ability to provide AOS services, the SDS simulated AOS Coded Virtual Channel Data Units (CVCDU) consisting of Private Data Units for Virtual Channel Access Service, Insert Zone Data Units for Insert Service, Bitstream Data Units for Bitstream Service, fixed and variable length packets within Multiplexed Packet Data Units (MPDUs) with repeated APIDs on different VCs. The CVCDUs contained incomplete and fill packets, and included the full range of errors as described in the FAST PPS test scenarios.

In the current implementation, all error injection scenarios and the foibles introduced in the data sets are designed in advance and manually catalogued through the TPGEN menu before the data generation process is initiated. As data scenarios get more and more complicated, this can become

a tedious and time consuming process. Each time a scenario changes, even just adding or removing one error, the entire data set has to be regenerated.

MSB is currently engaging in the development of a second generation data simulation package. Based on UNIX platform, this package will be highly modular and script-driven, and can be ported to different UNIX workstations. CCSDS telemetry data will be simulated by first generating a set of basic data units, then piping it through a series of simulation modules, each of which is responsible for one layer of CCSDS protocol or one type of data manipulation. Data or error characteristics will be specified through scripts. Graphical-based user interfaces will be provided to help users design, generate, modify, and examine their test data sets. While maintaining all SDS capabilities as described in this paper, this layered and modularized architecture will greatly improve efficiency.

5. SUMMARY

SDS has demonstrated its versatility and flexibility by supporting the LZP project through all phases of development. Its unique capabilities to simulate realistic spacecraft CCSDS data streams, especially SSR data scenarios, proved to be invaluable for system I&T of the VLSI LZP prototype phase 1 and 2 systems as well as the FAST PPS. The knowledge and expertise gained in the development of the current SDS will be used to develop the new generation of data simulators capable of testing systems running at speeds in excess of 300 Mbps.

6. REFERENCES

1. "Packet Telemetry," CCSDS 102.0-B-3, Blue Book, Consultative Committee for Space Data Systems, November, 1992.
2. "Advanced Orbiting Systems, Networks, And Data Links," CCSDS 701.0-B-2, Blue Book, Consultative Committee for Space Data Systems, November, 1992.
3. Shi, J., Horner, W., Grebowsky, G., Chesney, J., "A Prototype VLSI Level Zero Processing System Utilizing the Functional Component Approach," Proceedings of International Telemetering Conference, 1991, pp. 519-531.
4. Shi, J., Horner, W., Grebowsky, G., Chesney, J., "Fast Auroral Snapshot Explorer (FAST) Packet Processing System (PPS)," Proceedings of International Telemetering Conference, 1993, pp. 445-459.
5. Shi, J., Mao, T., Clotworthy, T., Grebowsky, G., "Lessons Learned Supporting On-Board Solid-State Recorders," Space Ops 94.
6. Shi, J., Harris, J., Speciale, N., Bennett, T., "A Second Generation 50 Mbps VLSI Level Zero Processing System Prototype," Space Ops 94.

7. NOMENCLATURE

AOS	Advanced Orbiting Systems
AP	Application Process
APID	Application Process Identifier
CCSDS	Consultative Committee for Space Data Systems
COTS	Commercial Off-the-Shelf
CRC	Cyclic Redundancy Check
CVCDU	Coded Virtual Channel Data Units
DG	Data Generator
ECL	Emitter Coupled Logic
FAST	Fast Auroral Snapshot Explorer
GSFC	Goddard Space Flight Center
I&T	Integration and Test
LVGEN	Large Volume Data Generator
LZP	Level Zero Processing
MPDU	Multiplexed Packet Data Units
MSB	Microelectronic Systems Branch
Nascom	NASA Communications
PED	Polynomial Error Detector
PHD	Packet Header Definition
PPS	Packet Processing System
SAMPEX	Solar Anomalous and Magnetospheric Particle Explorer
SDS	Spacecraft Data Simulator
SSR	Solid-State Recorder
TOPEX	Topographical Explorer
TPGEN	Test Pattern Generator
VC	Virtual Channel
VME	Versa Module Eurocard
VLSI	Very Large Scale Integration

Systems Engineering

* *Presented in Poster Session*

111168 N95-17583

354265

P-9

RE-ENGINEERING THE MULTIMISSION COMMAND SYSTEM AT THE JET PROPULSION LABORATORY

SCOTT ALEXANDER
JEFF BIESIADECKI
NAGIN COX
SUSAN MURPHY
TIM REEVE

Operation Engineering Lab
Jet Propulsion Laboratory
California Institute of Technology
MS 301-345
Pasadena, California 91109-8099
{salex, jeffb, nagin, sooz, timr@devvax.jpl.nasa.gov}

ABSTRACT

The Operations Engineering Lab (OEL) at JPL has developed the multimission command system as part of JPL's Advanced Multimission Operations System. The command system provides an advanced multimission environment for secure, concurrent commanding of multiple spacecraft. The command functions include real-time command generation, command translation and radiation, status reporting, some remote control of Deep Space Network antenna functions, and command file management. The mission-independent architecture has allowed easy adaptation to new flight projects and the system currently supports all JPL planetary missions (Voyager, Galileo, Magellan, Ulysses, Mars Pathfinder, and CASSINI).

This paper will discuss the design and implementation of the command software, especially trade-offs and lessons learned from practical operational use.

The lessons learned have resulted in a re-engineering of the command system, especially in its user interface and new automation capabilities. The redesign has allowed streamlining of command operations with significant improvements in productivity and ease of use. In addition, the new system has provided a command capability that works equally well for real-time operations and within a spacecraft testbed. This paper will also discuss new development work including a multimission command database toolkit, a universal command translator for sequencing and real-time commands, and incorporation of telecommand capabilities for new missions.

INTRODUCTION

The Jet Propulsion Laboratory has a long history of building multimission ground data systems that are designed to be easily adaptable to new projects. The mainframe-based systems of the 1970s have been replaced by distributed, workstation-based systems as part of JPL's advanced Multimission Ground Data System (MGDS). The new MGDS provides flexible, extensible components that are easily adapted for new missions, but more importantly, can also support multiple missions concurrently. However, as these ground systems have evolved, it has become apparent that providing advanced tools that help simplify and automate the old way of doing business is not enough to support the small, low-cost missions of the future. In particular, the uplink process has been very labor intensive for planetary missions and it must be re-engineered to provide the simple command capabilities that will be needed for missions with cheaper, more autonomous spacecraft and for operators wanting remote telescience capabilities.

PRECEDING PAGE BLANK NOT FILMED

PAGE 1122 INTENTIONALLY BLANK

The Operations Engineering Lab (OEL) has developed and refined the MGDS Command Subsystem to be an adaptable, low-cost, multimission component of the overall uplink process. As part of our development work, the OEL is working with the sequencing teams and developers at JPL to re-engineer the uplink process so it can provide seamless, easy-to-use capabilities for spacecraft commanding. The goal is to provide an off-the-shelf command package that can support large to small missions that need to command through the Deep Space Network (DSN).

MGDS COMMAND SYSTEM DESCRIPTION

The MGDS Command functions include real-time command generation, command translation and radiation, status reporting, remote control of DSN antenna functions, and command file management. A distributed, network-based, graphical interface is provided to give real-time command radiation status to users at remote sites. This interface was implemented in X/Motif. The Command System provides security functions including authentication for two user privilege levels, internal security checks, a central node for controlling all command radiation processing, a configuration control environment for command files, and a mode for non-interactive Command viewing.

The primary control function of the Command system is to permit real-time transmission of command files and memory loads from the ground to a spacecraft. The Command Control Graphical User Interface (GUI) (Figure 1) provides real-time, interactive control of the command transmission and radiation to the spacecraft. The connection between Command and the DSN is a secure process controlled by the Data System Operations Team at JPL. These operators allocate the connection resources to a project mission control team after ensuring a clean commanding interface.

Command files are first transmitted to the DSN and held at the receiving end until the completeness and integrity of the file transfer can be verified. Once there, the user is free to put the files in the queue of the Command Processor Assembly for radiation to the spacecraft either at that moment or some later specified command window. The user also uses the Command GUI to remotely control the configuration of the antenna in terms of when actual radiation of commands.

Any time the user is connected to a DSN station, the station returns monitor data which is displayed in the Command GUI for inspection by the operator. Monitor data contains information about the current antenna configuration, acknowledgments of command file radiation, and constant status information including alarms, files at the CPA, and receipt of command blocks.

Command files are generated prior to transmission using the Command system or the Sequence software system. Both processes are similar. A spacecraft command sequence is formulated and constraint-checked and then the actual commands are entered as command mnemonics, encoded abbreviations (with parameters) that tell the spacecraft what commands to perform. The command mnemonics are translated into a spacecraft-ready file that contains binary translations of the mnemonics, spacecraft identification information, start and acquisition codes, and file integrity and error-detection information. Once the command files are prepared, they are stored and made available to the Command system through a secure database that checks command formats and user permissions. Before transmission, the command files are reformatted for recognition and radiation by the DSN (Figure 2).

LESSONS LEARNED

When the MGDS Command System was completed, existing projects were required to transition from the mainframe MCCC Command System. Voyager was chosen as the first project to transition since it had entered its interstellar cruise phase. Their experience provided multiple lessons learned about simplifying the user interface and reducing the number of steps in the uplink tasks.

When the Mars Observer (MO) Project came on line as a new project, they had no prior system for comparison. Their experience was different since they had a much higher command rate than the Voyager mission. They had also decided not to implement the real-time command translation capability in the MGDS Command System as a cost-cutting measure. This meant that all of their command files, even those with only a single non-interactive command, had to be prepared off-line using the more complex Sequence software. As a result, the project was having difficulty keeping up with its command rate, even in the early cruise phase. When the spacecraft went into emergency mode, commanding became a 24-hour activity with many engineers required in the process.

There were two lessons learned from the MO use of the command system. First, eliminating the real-time translator during the mission planning phase resulted in increased costs in the mission operations phase. Second, the number of steps needed to prepare commands had to be reduced. In particular, the use of the security-controlled Command GUI had to be re-evaluated. The GUI was required to perform even simple file reformatting functions, with no options for a command-line interface or batch-mode. This reliance on a graphical interface prevented automating some steps with simple scripts because a user had to be sitting at the computer, pushing each button in turn. It became apparent that we had to provide an off-line command generation capability that was based on non-graphical, less restricted, command-line interfaces. The secure GUI was still essential for transmission and radiation of commands. With this re-design, DSN resources are only required for the final transmission and radiation of the Command files to the spacecraft. The impact of this off-line capability on required network resources is significant.

Thus, the Command system interface was redesigned to allow users to generate command files in an off-line environment without requiring a connection to the command control GUI. First, the translation and reformat functions were developed into separate, stand-alone programs. The translation program translates text mnemonic commands into an intermediate SpaceCraft Message Format (SCMF) file containing binary commands expected by the spacecraft. The reformat program packages the binary commands into the form expected by the DSN. These programs can be started up by the user on the UNIX command line or script, as well as by the central command system. The off-line capabilities have also allowed script automation to reduce the number of manual, interactive steps involved in the generation of command files. A graphical interface shell was built using the JPL-developed PERL scripting language and OELSHELL interface building tool.

This off-line translation toolkit also found extensive use in spacecraft flight testbed facilities where no connection to the DSN was allowed. Testbeds provide an environment for testing and validating commands on a mock spacecraft. The testbed command system sends commands directly to the ground support equipment.

Another lesson learned was the need to streamline and simplify the end-to-end uplink process. The uplink process involves multiple operations and development teams. This creates a system with multiple tools and interfaces, forcing the user to learn how to operate across several different boundaries. From a project perspective, there should only be a single interface to the uplink process that would allow a single user to perform all functions including spacecraft sequence generation and translation, ground sequence of events schedule generation, real-time command preparation, mnemonic translation, and command transmission and verification. The OEL has worked closely with the Mars Surveyor Project to implement an integrated, graphical interface tool that allows a single user to seamlessly perform end-to-end functions in the uplink process.

The successful experience of the early projects using the MGDS Command system eased the transition of the remaining projects. All of the JPL planetary missions have now transitioned successfully to the MGDS Command System and the mainframe-based

Command system was decommissioned a year earlier than originally planned.

RE-ENGINEERING COMMAND TRANSLATION

Since both the Sequence and Command software provided capabilities for a user to generate command files, there were common translation capabilities duplicated in both systems. The OEL has worked closely with Sequence developers to re-engineer the translation process and develop a universal command translator that can be used by both subsystems. The redesigned system includes the use of advanced graphics and object-oriented techniques.

The translation functions in the Sequence system were based on manually building mnemonic-to-bit translation information in each project's unique command macro language. These project-specific adaptations were time-consuming and error-prone. The command translation process in the Command software was based on a multimission Command Definition Language (CDL) that can be used to specify command mnemonic-to-bit definitions and constraints. The CDL file is compiled into a project's Command Database. A command database is built for each project, but the language compiler, database interpreter, and translator software is multimission. In the re-designed uplink process, the command database interpreter and translator software was re-built as generic, universal libraries that could be called by both Command and Sequence software. This multimission, common approach will significantly reduce uplink costs.

An illustrative example of CDL code follows:

```
! define a memory load message
MESSAGE: memload-msg(buf: 200)
    FIELDS
        data: 160    ! 160 bit local variable
    END FIELDS

    ! declare the kinds of arguments that will
    ! be entered by the user
    LOOKUP ARGUMENT: name
        ! lookup value below in hex
        CONVERSION: HEX
        LENGTH: 8
        'MEMLOAD' = 'A9'
    END LOOKUP ARGUMENT
    NUMERIC ARGUMENT: address
        ! user to enter number in hex
        CONVERSION: HEX
        LENGTH: 16
        ! acceptable range
        '00FF' TO 'FFFF'
    END NUMERIC ARGUMENT
    NUMERIC ARGUMENT: aword
        ! user to enter number in hex
        CONVERSION: HEX
        LENGTH: 16
    END NUMERIC ARGUMENT

    ! read mnemonics from user input
    READ ARGUMENT name
    READ ARGUMENT address
    REPEAT 1 TO 10 TIMES
                (COUNTING WITH nwords)
        READ ARGUMENT aword
        data := data // aword
    END REPEAT

    ! combine converted input into a message
    ! counters like "nwords" are 16 bits
    buf := name // address // nwords // data
END MESSAGE
```

It defines a memory load message that can load up to ten words, sixteen bits each, into a certain area of memory. If the user's mnemonic input was, for example,

MEMLOAD; 0A48; 1; 22; 333

the resulting hex output would be:

A9 0A 48 00 03 00 01 00 22 03 33

where the first byte is an op code that signifies a memory load instruction, the next two bytes are the address to load the data into, the next two bytes are the number of words in the data, and the remaining six bytes are the data itself.

Since CDL files can become very complex, a command generation toolkit is being developed to facilitate their creation and browsing. The CDL toolkit will include a graphical CDL editor, a CDL parser and compiler, and various report generators. In the future, some text based on-line reference tools and a smart editor to help a user create mnemonics are planned.

The first step taken in the development of the toolkit was to determine the data structures for holding the information contained in a CDL file. These structures are accessed through a library that is used by all tools in the toolkit. Here, an object-oriented approach was used. For example, CDL has several types of processing routines. So, one of the classes was for that of a general processing routine. A subclass of the general routine is a message routine. Arguments are also objects, with lookup arguments and numeric arguments derived from a common, more general, argument class. For the CDL code above, there is one instance of a message routine, **memload-msg**. There are instances of both kinds of argument objects: **name**, **address**, and **aword**.

When an object is created, a parent object is specified. Whenever an object is destroyed, all of its children are automatically destroyed as well. For the example above, **name**, **address**, and **aword** are children of **memload-msg**. So if the user of the graphical editor chooses to delete the **memload-msg** routine, the code for the editor is simply one call to destroy the appropriate parent object and all of the child objects (which are not useful by themselves) are automatically cleaned up.

CDL objects can refer to each other. For an easy example, the **READ ARGUMENT address** statement is itself an object (in this case, of class input processing statement and child of **memload-msg**). It contains a reference to the object corresponding to the argument to be read. Thus, if the CDL editor user changed the name of the **address** argument, when the CDL code was saved the **READ ARGUMENT address** statement would automatically be written with the new name. Note that for this example, the editor will not allow the **address** argument to be destroyed until the reference to it in the **READ ARGUMENT address** statement is changed or the statement removed altogether. It is easy to get a list of references to any object. There are many constructs in CDL not shown in the example that lead to a single object being referred to in several places.

The CDL language was designed years ago as part of the old mainframe-based command system. It is missing some important functionality such as arithmetic and comparison operators. CDL was also written before the Telecommand standard, so some of its constructs are outdated and intended for tasks such as embedding error polynomials into the binary commands. In the new Command system, any functionality not present in the CDL language must be added as hard-coded 'user hooks' to the command translation software, creating additional expense for development and testing. Thus, as part of our re-engineering efforts, we are incorporating important enhancements and simplifications to the CDL language. For some of these enhancements, we are investigating the use of other process control languages such as Spacecraft Control Language (SCL) in the uplink process. With the object-oriented approach taken and the goal of reducing class-specific code, we expect it to be easier to make changes to the language.

We are also investigating extending CDL to include information that would typically be found in a command dictionary such as telemetry verification points and flight rule constraints. The graphical CDL toolkit is also being enhanced to provide a complete command definition and dictionary toolkit with hypertext references to other mission documentation.

Other recent development work includes porting our code to multiple UNIX hardware platforms, ANSI-C, and XPG-4 open standards. In addition, we are incorporating the 1987 Consultative Committee for Space Data Systems (CCSDS) "Telecommand" standards into the MGDS Command System. All future JPL missions will comply with this standard.

TELECOMMAND IMPLEMENTATION

The Telecommand service model is a layered model which more or less parallels the ISO Open Systems Interconnect model. The highest two layers of this model, the Application Process layer and the System Management layer, have not yet been

specified in detail. It is still up to the individual project to define procedures and data structures in these layers. The layers below this, however, *have* been specified in detail. Our response to the standard addresses the Packetization, Segmentation, Transfer, Coding, and Physical Layers

In the Command subsystem, the Telecommand (TC) standard is being implemented as a generic, batch-mode "wrapping service." Clients of the service supply the data to be wrapped in ASCII formatted files called Command Packet (CMD_PKT) files. The service takes multiple, one or more, CMD_PKT files as input, wrapping the data from each file record and time-order merging the results into an SCMF file.

CMD_PKT file format

The format of the CMD_PKT file follows the CCSDS Standard Formatted Data Unit (SFDU) standard. The I-data (user) section of the file is organized as a header section followed by a series of data sections. The section boundaries are defined with special markers, and the information within these sections is organized in a "keyword = value" format. An example of a header section follows:

```
$$MPF   COMMAND PACKET FILE
*CMDPKT        SEQTRAN.CMDPKT/JOB001
*OPERATOR      Frank Zappa
*PROGRAM       SEQTRAN - MARS
        PATHFINDER V19.0 APR 29, 1994
*CREATION      JPL 94-131/09:58:59
*BEGIN          *****  NO DATA  *****
*CUTOFF         *****  NO DATA  *****
*TITLE          *****  NO DATA  *****
*ZERO           *****  NO DATA  *****
*CMDFIL         *****  NO DATA  *****
*FILSIZ          6
*SISVER          04/27/94
*FRMVER          1
*CDUACQLEN  22
*CDUACQ        55
*CLTUSSQLEN 2
*CLTUSSQ        EB 90
*CLTUTSQLEN 8
*CLTUTSQ        55
*CLTUDLY        ENDSTART/BITS/0
*FRMPERCLTU 1 $$EOS
```

The header section contains global file information. For example, the value of the 'FILESIZ' keyword tells you the number of data sections which follow. 'CDUACQLEN' and 'CDUACQ' together form a specification of the acquisition sequence to be used for this file. 'CDUACQLEN' is the number of octets in the acquisition sequence and 'CDUACQ' is the smallest repeat pattern. Using the above record, the Telecommand wrapping service would generate 22 octets of 55 hex.

Each data section contains ASCII hexadecimal data to be wrapped, along with enough information to fill in the Telecommand headers. Here is an example:

```
$PKT            SCGNLD
PKTVER          1
SEQFLGS         FIRST
CHECKSUM        947D
VC              1
LENGTH          12
APPID           0
OPENWIN         82-080/11:40:00.000
CLOSEWIN        82-080/12:00:00.000
FRMSEQ          0
FEC             EACSUM55AA
CTRLCMD         NO
BYPASS          YES
PACKETIZE       N
FRAMING         YES
SEGMENTING      NO
DATA
        0A01 0000 0200 0001 0002 0003
        0004 0005 0006 0007 0008 0009
$EOP
```

Following the DATA keyword is a sequence of ASCII hexadecimal words. This represents the binary data to be wrapped. The format and structure of this data is known to the higher layers of the CCSDS Telecommand service model (system management and application layers). The values of other keywords enable the wrapping service to fill in the TC headers. For example, the value of the VC keyword tells the wrapping service what to put into the 6-bit 'virtual channel ID' field of the TC transfer frame header.

The creator of this file also has control over which layers of wrapping are applied to the data. The wrapping service concerns itself with the following layers:

- TC packetization layer (TC packets)
- TC segmentation layer (TC segments)
- TC transfer layer (TC transfer frames)
- TC coding layer (Command Link Transmission Units (CLTUs), consisting of TC codeblocks)

For example, consider the keywords PACKETIZE, SEGMENTING, and FRAMING. PACKETIZE and SEGMENTING are both set to NO, while FRAMING is set to YES. This means that the TC wrapping service will consider the data to be the contents of a TC frame, and will only prepend a TC frame header (and may also append a Frame Error Control word, if the FEC keyword is set to a value other than NONE), before creating a CLTU. If PACKETIZE were set to YES, the wrapping service would consider the data to be the contents of a TC packet, and would apply a TC packet header. Then, if SEGMENTING and FRAMING were both set to YES, the TC packet would be broken into TC segments, and then each TC segment would be wrapped as a TC frame, before creating a CLTU. Currently, all eight permutations of (PACKETIZE, SEGMENTING, FRAMING) are allowed by the wrapping service, though only three may be legal: (NO, NO, YES), (YES, NO, YES), and (YES, YES, YES). This flexibility makes the name 'CMD_PKT' something of a misnomer; perhaps 'CMD_TC' would have been a better choice.

Currently, each data section of this file will result in one or more CLTUs. Normally, only one CLTU will be created per data section; the only thing which can affect this is the setting of the FRMSPERCLTU keyword in the CMD_PKT header section. If this is set to a value N, where N > 0, then no CLTU may contain more than N TC frames. So, if the amount of data in the data section is large enough that when it is segmented, more than N TC frames are created, more than one CLTU will result.

Each data record contains a timestamp as well. This may be specified as either a window (OPENWIN, CLOSEWIN) or an execution time (EXECTIME). Times are expressed in GMT relative to the spacecraft (SpaceCraft Event Time, or SCET). For a given data record, this means that the data in that record will be at the spacecraft, ready to be processed, at the given (EXECTIME), or within the given window (OPENWIN, CLOSEWIN).

TC Wrapping Service

This service is implemented as a single process which consumes one or more CMD_PKT files and produces a single SCMF (SpaceCraft Message Format) file. Each data record of the SCMF file contains a single 'spacecraft message', which in this case is a CLTU.

Each CLTU within a record may be preceded by an acquisition sequence, depending upon the PLOP (Physical Layer Operation Procedure) in use by the project. Currently two PLOPs are defined in the TC standard. In PLOP 1, CLTUs are individually radiated, meaning that the physical telecommand channel is deactivated after each transmitted CLTU. In this case every CLTU in the SCMF file must have an acquisition sequence prepended. In PLOP 2, the physical channel is not deactivated until the last CLTU in an 'upload' has been transmitted. For our purposes, this means that only the first CLTU of the SCMF file will be preceded by the acquisition sequence.

The TC wrapping service places the resultant CLTUs in ascending time order within the SCMF. Further, the timestamp in each record of the SCMF is the time of radiation of the first bit in the record. This means that in going from execution time in CMD_PKT file(s) to an SCMF, all times have to be backed off by the number of bits in the record (multiplied by the time of one bit at the current uplink rate), plus any inherent spacecraft delay time, plus the appropriate one-way light time. All of this is a fairly complex operation, since we are merging multiple CMD_PKT files, each of which can

have a mixture of window and execution time records.

TC Wrapping Service Design

A modular approach was taken in the design of the wrapping service. It is decomposed into five primary modules, as follows:

1. CMD_PKT file I/O module.
2. SCMF file I/O module.
3. Light time module.
4. Telecommand module.
5. Main module.

The first four modules are implemented as libraries. The main module calls functions in these libraries. The CMD_PKT file module depends upon the Telecommand module as well, mainly for validation of TC header field values.

The CMD_PKT file I/O module isolates all of the knowledge of the format and structure of CMD_PKT files. Its set of exported functions allow record-oriented I/O (both reading *and* writing) of CMD_PKT files.

The SCMF file I/O module is directly analogous to the above, for SCMF files.

The Light time module contains functions which perform conversion between ground transmission times (TRM) and spacecraft event times (SCET). This module reads a LIGHTTIME files in order to perform its function.

The Telecommand module isolates all of the knowledge of the TC data structures. It contains a set of functions for validating all of the TC header fields values, as well as a set of functions for performing TC wrapping. This module also maintains a table of project-dependent Telecommanding data. Items such as default acquisition, start, and tail sequences, virtual circuit and application id mnemonics, TC codeblock size, and PLOP are included in this table. The main module is responsible for the overall control of the wrapping process, and deals directly with the time-ordering issue.

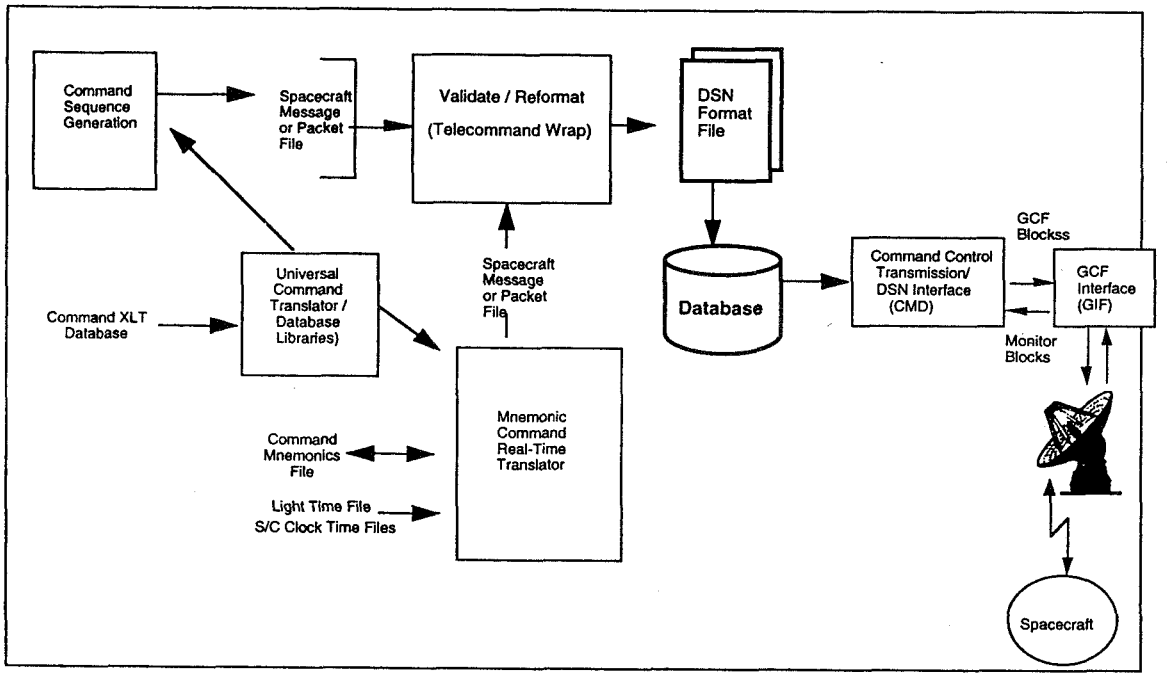
CONCLUSION

The Operations Engineering Lab has developed the JPL multimission command system to provide low-cost, adaptable, extensible uplink capabilities to new and existing flight projects. The goal in the ongoing re-engineering of the command subsystem is to create a set of independent tools to allow more flexibility for the user and to make any necessary customization faster and easier for future, low-cost missions.

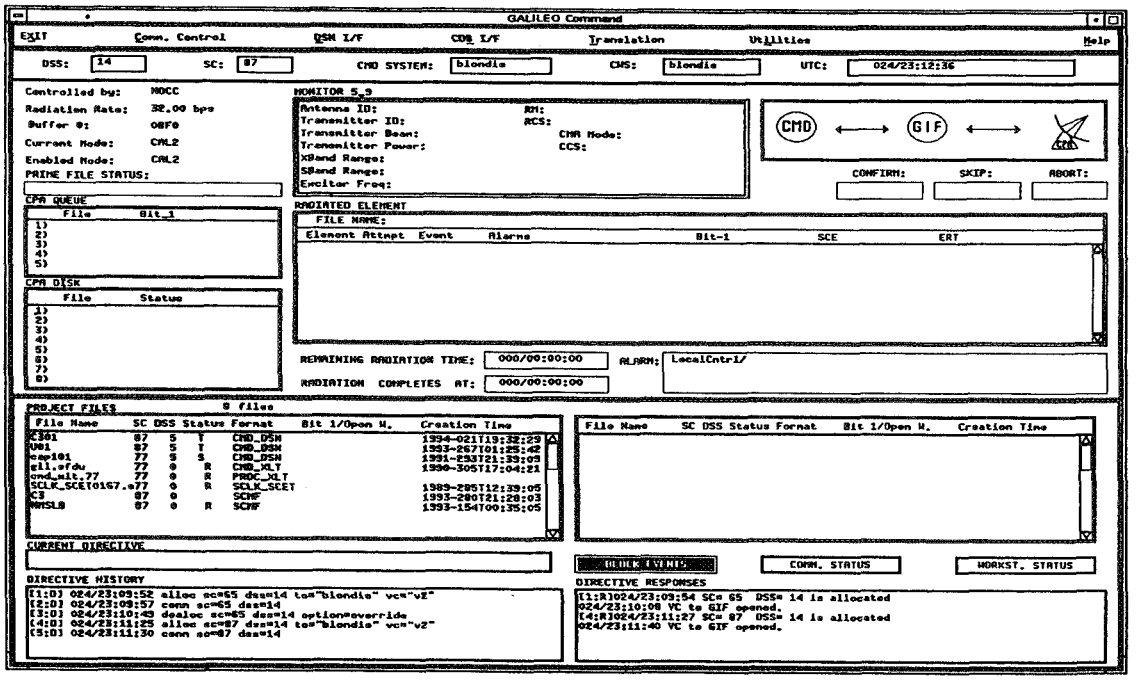
ACKNOWLEDGMENTS

This work was done at the Jet Propulsion Laboratory, California Institute of Technology, under a contract from the National Aeronautics and Space Administration. Thanks to the technical staff in the OEL, the early MGDS development teams, and the JPL Mission Operations Teams for their enthusiasm and support.

Data Flow for Commanding Process



Command GUI



ORIGINAL PAGE IS
OF POOR QUALITY

N95-17584

111169

354269

P. 9

RE-ENGINEERING NASCOM 'S NETWORK MANAGEMENT ARCHITECTURE

BRIAN C. DRAKE
GODDARD SPACE FLIGHT CENTER (GSFC)
CODE 541.2
GREENBELT, MD 20771
&
DAVID MESSENT
COMPUTER SCIENCES CORPORATION
7700 HUBBLE DRIVE
LANHAM-SEABROOK, MD 20706

ABSTRACT

The development of Nascom systems for ground communications began in 1958 with Project Vanguard. The low-speed systems (rates less than 9.6 Kbs) were developed following existing standards; but, there were no comparable standards for high-speed systems. As a result, these systems were developed using custom protocols and custom hardware.

Technology has made enormous strides since the ground support systems were implemented. Standards for computer equipment, software, and high-speed communications exist and the performance of current workstations exceeds that of the mainframes used in the development of the ground systems.

Nascom is in the process of upgrading its ground support systems and providing additional services. The Message Switching System (MSS), Communications Address Processor (CAP), and Multiplexer/Demultiplexer (MDM) Automated Control System (MACS) are all examples of Nascom systems developed using standards such as, X-windows, Motif, and Simple Network Management Protocol (SNMP). Also, the Earth Observing System (EOS) Communications (Ecom) project is stressing standards as an integral part of its network. The move towards standards has produced a reduction in development, maintenance, and interoperability costs, while providing operational quality improvement.

The Facility and Resource Manager (FARM) project has been established to integrate the Nascom networks and systems into a common network management architecture. The maximization of standards and implementation of computer automation in the architecture will lead to continued cost reductions and increased operational efficiency. The first step has been to derive overall Nascom requirements and identify the functionality common to all the current management systems. The identification of these common functions will enable the reuse of processes in the management architecture and promote increased use of automation throughout the Nascom network.

The MSS, CAP, MACS, and Ecom projects have indicated the potential value of commercial-off-the-shelf (COTS) and standards through reduced cost and high quality. The FARM will allow the application of the lessons learned from these projects to all future Nascom systems.

PAGE 1132 INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

INTRODUCTION

The development of NASA Communication (Nascom) systems for ground communications began in 1958 with Project Vanguard. The low-speed systems (less than 9600 bps) were developed following existing standards. However, the existing communication standards could not be used to meet the higher data rates and transmission reliability requirements demanded by next generation of NASA projects. As a result, custom protocols and associated hardware had to be developed to meet the needs of the growing space agency.

The Nascom protocol that was developed in 1968 consisted of a 1200-bit block with header information, data field, and polynomial error control field at the end. Over time, user communication requirements increased and the Nascom block size was increased to 4800 bits. The basic structure remained the same, but provided an expanded user data field.

In the 1970's, development began on a system of geosynchronous satellites to relay data around the world instead of many earth stations. As the Tracking and Data Relay Satellite System (TDRSS) grew, it became evident the manual network control and monitoring would be impossible. Nascom began to automate its portion of the network.

Once again, network management standards did not exist to meet NASA's requirements. Nascom began custom development for automated network management, using the existing Nascom protocol with 1200-bit blocks as the management protocol.

With today's reality of budget reductions and the high cost of custom development, the use of standards has become as important as high-speed and data integrity. Fortunately, communication standards for both transmission and management had been developing over the last few years to keep up with user demands.

Nascom began its move towards standards with development of an X.25 network for the Pacor/Gamma Ray Observatory (GRO) project and the Mission Operations and Data Systems Directorate (MO&DSD) Operational/ Development Network (MODNET)/ Nascom Operational Local Area Network (NOLAN) for high-speed local data distribution. Now, Nascom is developing a high-speed Asynchronous Transfer Mode (ATM) network for the EOS project and re-engineering its automated network management architecture to integrate these standards-based networks and the existing customized network into a centralized operations area.

EXISTING ARCHITECTURE

Nascom currently has three distinct networks with separate management systems: the Data Distribution and Command System (DDCS), MODNET, and the 4800-bit block network.

The DDCS X.25 network has two nodes located at GSFC, a node at Marshall Space Flight Center (MSFC) and a node at the University of California at Berkeley (UCB) (figure 1), distributing X.25 packets at data rates up to 224 Kbps. Each node has a redundant backup for automatic fail over. It is composed of COTS hardware and software with integrated network management software. The controlling node is located in the Nascom computer facility at GSFC.

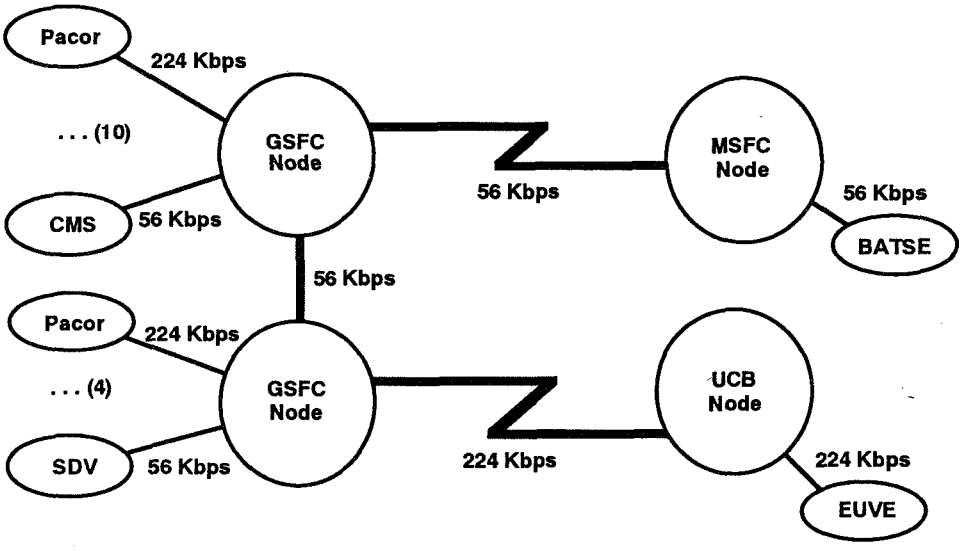


Figure 1. The DDCS

The second network is the Mission Operations and Data Systems Directorate (MO&DSD) Operational/ Development Network (MODNET). MODNET was formed in May 1986 through the integration of three division-level networks: the MOD&DSD Network (MNET), Mission Operations Division (MOD) Local Area Network (MODLAN), and the Information Processing Division LAN (InfoLAN). This network provides high-speed, 50 Mbps, interconnectivity between operational LANs and computer systems at GSFC (figure 2).

The LAN technology used by MODNET is HYPERchannel.

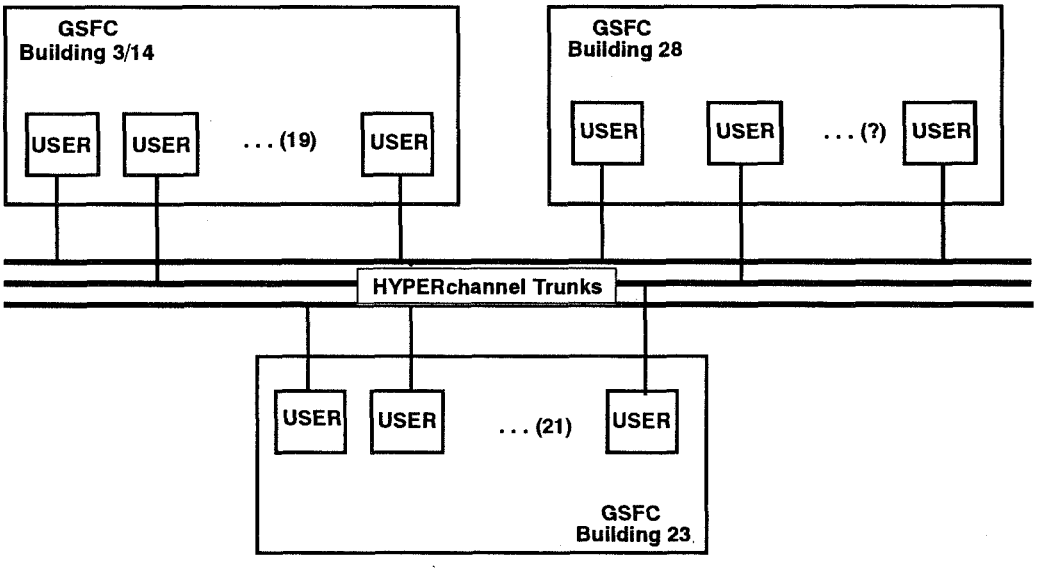


Figure 2. MODNET

HYPERchannel, a registered trademark, is a generic name for the various network components and protocols developed by Network Systems Corporation (NSC) to provide networking capabilities between entities within Nascom to handle the data requirements each with their own control capabilities. The MSS is the original Nascom block star network hub (figure 3). The MSS provides packet switching

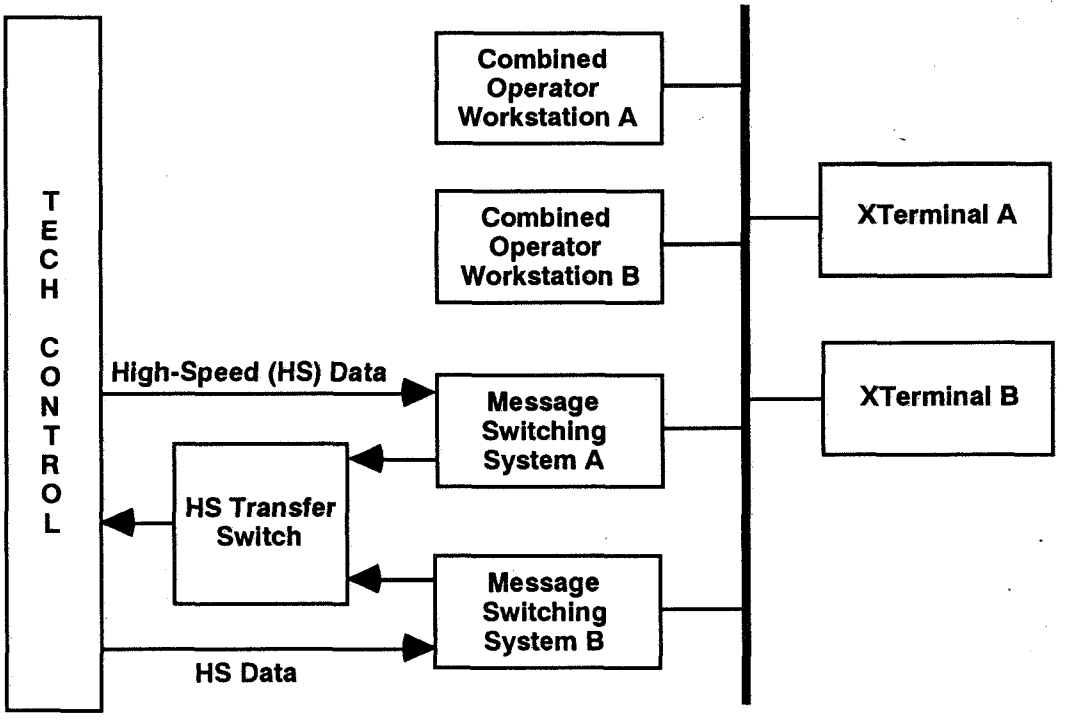


Figure 3. The MSS

networked computers. The fastest alternative protocol at the time MODNET was formed was the 10 Mbps Ethernet.

The NOLAN will expand upon MODNET's capabilities using a Fiber Distributed Data Interface (FDDI) backbone with a larger distribution area. MODNET/NOLAN will be managed by Nascom operations using Hewlett-Packard's HP Openview network management system with ISICAD for automated trouble ticketing.

The final network is the 4800-bit block network that distributes data throughout the world with data rates ranging from 50 baud to 2 Mbps. There are two network capabilities for hundreds of 4800-bit block users with data requirements ranging from 9600 bps to 1.544 Mbps. The management control consists of custom developed software operating on a Sun workstation.

The TDRSS network portion of Nascom consists of custom built Multiplexer/Demultiplexers (MDMs) and controllers located at the Johnson Space Center (JSC), NASA Ground Terminal (NGT), MSFC, and GSFC, a custom Digital Matrix Switch (DMS) and controller at GSFC, and custom Data Link Monitoring Systems (DLMS) located at GSFC, JSC, and NGT. This network equipment is being constantly

reconfigured according to the network schedule supplied by the Network Control Center at GSFC. In order to meet the reconfiguration requirements of the network, the Control and Status System (CSS) was a custom development to provide automated network management (figure 4). The CSS uses the Nascom 1200-bit block to send commands and receive status from the network equipment.

and engineer an efficient network management architecture using new technology and standards. Computer workstations today can process the same data that required mainframes ten years ago. Standard protocols can send data at higher rates and provide interconnectivity between different hardware platforms. COTS software packages can be found to meet almost every need. Custom systems are not an effective application of limited resources any more.

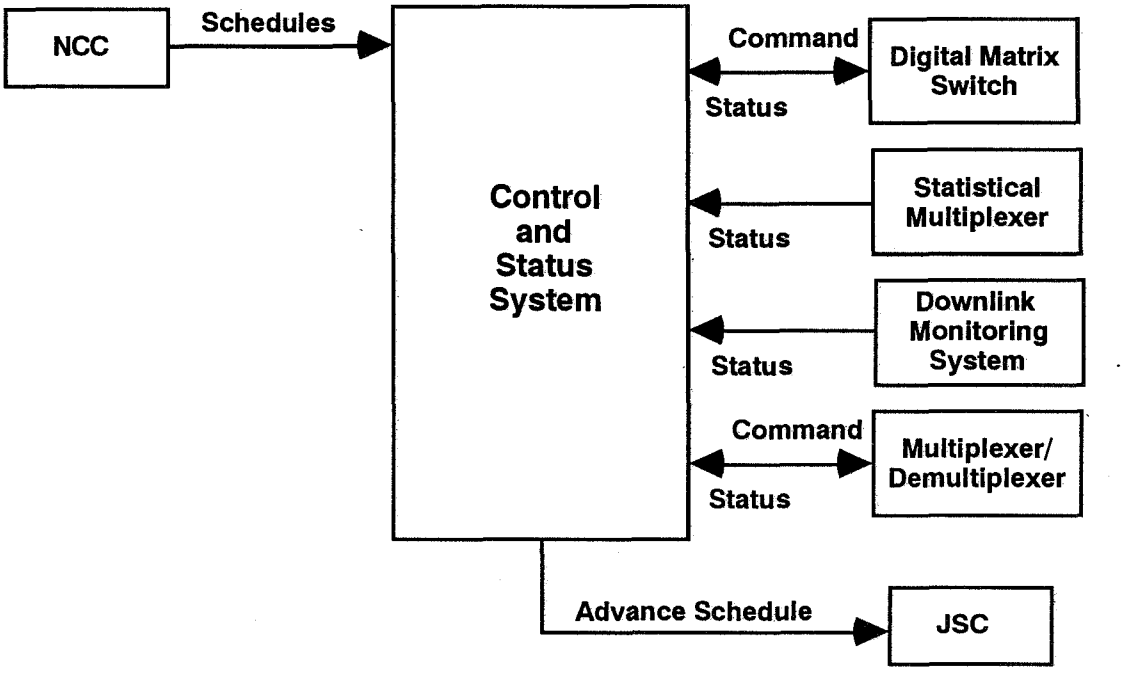


Figure 4. The CSS

MORE WITH LESS

Do more with less is the motto of the world today. Every year, the demand increases and the budget decreases. In order to maintain communication services for the Agency, Nascom has begun a project to re-engineer its network management architecture.

The Facility and Resource Manager (FARM) is an attempt to step back, analyze Nascom's management requirements, with an eye to the future, and engineer an efficient network management architecture using new technology and standards. Computer workstations today can process the same data that required mainframes ten years ago. Standard protocols can send data at higher rates and provide interconnectivity between different hardware platforms. COTS software packages can be found to meet almost every need. Custom systems are not an effective application of limited resources any more.

The basic goal of the FARM is to meet today's motto, more with less. More network operational demands with less budget cause additional capabilities and requirements to be engineered into the systems with less engineering budget. Standards, COTS, reuse, and new technology are the FARM's tools.

Operational Costs

The biggest cost to Nascom operations is the large number of personnel required to run the networks. The 4800-bit block

network is very manually intensive and operates in a reactive mode for troubleshooting. MODNET, NOLAN, and the X.25 networks are a little more automated, less personnel dependent; but being separate networks, they still require additional operations personnel. In the near future, Nascom will implement and operate the Federal Telecommunications System (FTS)-2000 network and the EOS Communications (Ecom) ATM network. Although these are both highly automated networks, they also have their own network management systems that have to be operated, requiring additional personnel.

from text menus to windowed graphical interfaces. Comprehensive operator training for each system has to be developed, which is time consuming and expensive.

THE FARM ARCHITECTURE

The FARM is developing the functional requirements of all the management systems and designing a workstation-based distributed network management architecture using SNMP between the systems. By down-sizing from mainframes, using COTS products, and Object Oriented (OO) development methodologies for software reuse, the

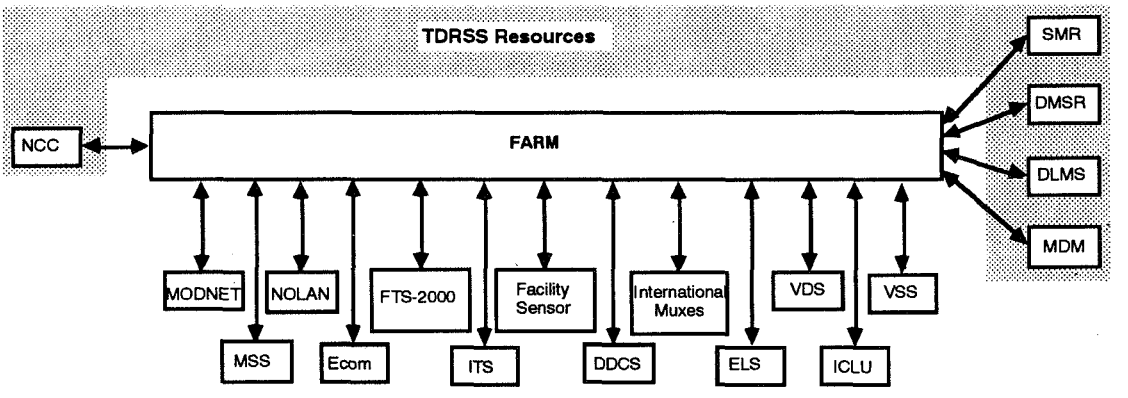


Figure 5. The FARM

Engineering Costs

Nascom engineering costs largely come from developing and maintaining specialized hardware and software. Until recently, to perform real-time processing of large amounts of data required the use of mainframes in conjunction with mini-computers and large amounts of assembly code. Add to that a unique protocol, requiring specialized hardware, and you have a recipe for a marching army of software and hardware engineers to keep Nascom operational.

Another cost is training personnel on every new system or network. Every one of the management systems operated by Nascom has a different user interface,

FARM will reduce engineering costs and development time. By consolidating multiple network management systems into a consolidated environment, the FARM will provide a consistent operator interface into the different networks, reducing training time and the need for separate operators for each system (figure 5).

The FARM has five functions: operator interface, data management, resource scheduling, system automation, and a non-SNMP gateway (figure 6). The operator interface will be designed using X-window and Motif standards to provide a comprehensive graphical user interface. COTS development tools, such as TAE+, will be used to provide rapid

screen prototyping. This will allow the FARM the capability to provide the most efficient user interface in the shortest amount of time.

The data management function will provide storage and retrieval of network configuration, management, and statistical information for the FARM. This functional will be performed using a Structured Query Language (SQL)-based database management system, such as Oracle or Sybase.

The resource scheduling function provides the FARM with the ability to configure TDRSS network equipment according to the network event schedule supplied by the Network Control Center. A large portion of this function can be performed by COTS network management software. The FARM project is evaluating several vendor packages for most comprehensive adherence to requirements.

System automation will be provided by using expert systems. Several of the COTS management systems also provide expert system development tools. The expert systems will analyze alarms and passive circuit monitoring equipment to perform pro-active troubleshooting for the network.

The Nascom networks have equipment and interfaces with specialized protocols. These non-SNMP interfaces will require gateways to convert between protocols. Again, several of the COTS management systems provide the capability to define unique interfaces.

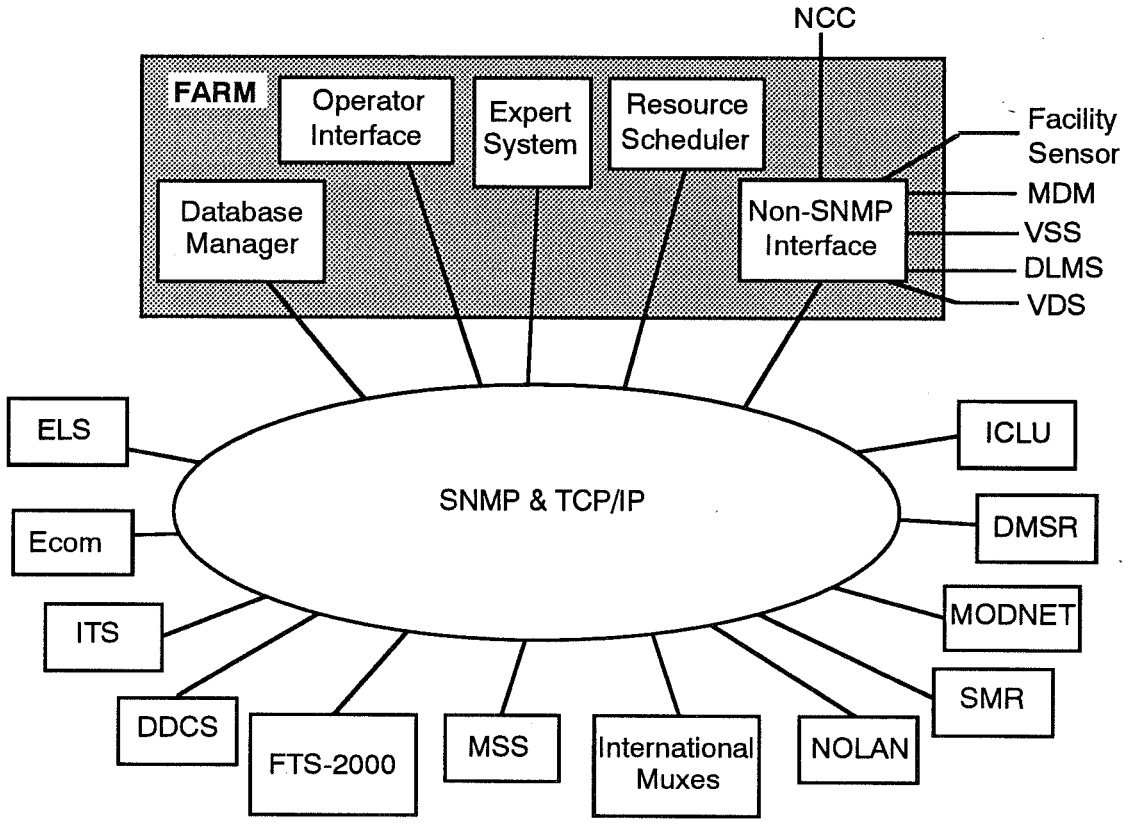


Figure 6. The FARM Functions

DEVELOPMENT METHODOLOGY

Development costs will also be reduced by applying some of the ideas from Object-Oriented (OO) methodologies to enhance reuse of already developed components and COTS products.

The FARM will not use an OO implementation language, as the project's short schedule, the long learning curve for OO methodologies, and the questionable success of OO projects are incompatible. Instead, the ideas within OO development that have been designed to enhance reuse are being employed.

The first of these OO development ideas is to perform a problem domain analysis. A domain is a problem area defined by the user interface, external interfaces and interaction of a proposed system. Its extent depends, in large, on the definition of what a user is. In the case of the FARM, users are operations personnel and developers. However, the developer is a user only to the extent necessary to safeguard reuse.

The problem domain analysis concludes with the specification of a development framework. The framework consists of the components identified in the problem domain analysis that appear to be candidates for reuse or COTS products in the system.

TESTING METHODOLOGY

COTS and reusable components are integrated into system development following System Engineering and Analysis Support (SEAS) System Development Methodology (SSDM). There are different levels of development testing required for COTS and reusable components depending on their level of reusability. The basic idea is that the supplied component is assumed to be fully tested and working. The task of development testing is to ensure it works within the system, that the interfaces are functioning as stated. System testing remains the same: the system is still tested to ensure requirements are met.

The level of testing, and when testing begins, depends on the component's position in the hierarchy of system components. When an entire function is reusable or COTS, the lowest level of testing required is module testing to ensure that the interface matches expectations. If the supplied function is more-or-less a client/server function, then integration testing is required to ensure that the client/server interface is correct.

Using the definition of a process as the smallest stand-alone system entity; testing should start at the level of testing in the following table:

Table 1. Testing Requirements

Level of Reuse/COTS	**Testing Required**
One or more processes	Integration testing
Complete function	Integration testing or Module testing
One or more units	Module testing
Modified unit	Unit testing

IMPLEMENTATION PLAN

The integration of management systems will be done in a phased approach, attacking the manually intensive legacy systems first, then moving to the more automated standards-based networks in later phases. This will allow Nascom to replace the expensive custom control systems, reducing the software maintenance costs.

The CSS and its associated control systems in the TDRSS network will be re-engineered to provide the framework for the consolidated network operations center. These systems are the most manually intensive, least automated systems. By consolidating these systems and implementing expert systems technology, operational costs will be greatly reduced.

Future phases of the FARM will integrate the remaining COTS management systems through SNMP interfaces and increasing operational automation.

A FARM lab is being developed which will be used to evaluate and test different hardware platforms and software packages, perform network performance analysis, and prototype new technologies and configurations for use in the FARM. By performing rapid prototyping and evaluation on every aspect of the FARM development, road blocks and design flaws will be identified early in the project life cycle.

CONCLUSION

The Nascom networks have evolved over the last thirty years into what they are today. Each system and network were developed to meet a specific set of requirements to support NASA's communications needs. Today, technology and standards have matured to the point that it makes engineering sense, as well as budgetary sense to start over from the beginning and re-engineer Nascom's network management architecture.

111170 N95-17585
354270
7.8

REENGINEERING NASA's SPACE COMMUNICATIONS TO REMAIN VIABLE IN A CONSTRAINED FISCAL ENVIRONMENT

Rhoda Shaller Hornstein
Office of Space Communications
National Aeronautics and Space Administration (NASA)
Washington, DC 20546

Donald J. Hei, Jr., Angelita C. Kelly, and Patricia C. Lightfoot
NASA Goddard Space Flight Center (GSFC)
Greenbelt, Maryland 20771

Holland T. Bell
NASA GSFC Wallops Flight Facility
Wallops Island, Virginia 23337

Izeller E. Cureton-Snead and William J. Hurd
NASA Jet Propulsion Laboratory
Pasadena, California 91109

Charles H. Scales
NASA Marshall Space Flight Center
Marshall Space Flight Center, Alabama 35812

ABSTRACT

Along with the Red and Blue Teams commissioned by the NASA Administrator in 1992, NASA's Associate Administrator for Space Communications commissioned a Blue Team to review the Office of Space Communications (Code O) Core Program and determine how the program could be conducted faster, better, and cheaper. Since there was no corresponding Red Team for the Code O Blue Team, the Blue Team assumed a Red Team independent attitude and challenged the status quo, including current work processes, functional distinctions, interfaces, and information flow, as well as traditional management and system development practices. The Blue Team's unconstrained, non-parochial, and imaginative look at NASA's space communications program produced a simplified representation of the space communications infrastructure that transcends organizational and functional boundaries, in addition to existing systems and facilities. Further, the Blue Team adapted the *"faster, better, cheaper"* charter to be relevant to the multi-mission, continuous nature of the space communications program and to serve as a gauge for improving customer services concurrent with achieving more efficient operations and infrastructure life cycle economies. This simplified representation, together with the adapted metrics, offers a future view and process model for reengineering NASA's space communications to remain viable in a constrained fiscal environment.

Code O remains firm in its commitment to improve productivity, effectiveness, and efficiency. In October 1992, the Associate Administrator reconstituted the Blue Team

PAGE 1142 INTENTIONALLY BLANK
PRECEDING PAGE BLANK NOT FILMED

as the Code O Success Team (COST) to serve as a catalyst for change. In this paper, the COST presents the chronicle and significance of the simplified representation and adapted metrics, and their application during the FY 1993-1994 activities.

Key Words: Blue Teams, Complexity Reduction, Economies of Scale, *"Faster, Better, Cheaper,"* Life Cycle Effectiveness, Mission Operations, Operations Concepts, Operations Technology, Process Improvement, Reengineering, Reusability, Reuse, Simplicity, Space Communications, Systems Engineering

1. INTRODUCTION

In addition to but separate from the Red and Blue Teams commissioned by the NASA Administrator, NASA's Associate Administrator for Space Communications commissioned a Blue Team to review the Office of Space Communications (Code O) core program and advise, within six weeks, how the program could be conducted faster, better, and cheaper. With no corresponding Red Team for the Code O Blue Team, the Blue Team was empowered to take an unconstrained, non-parochial, and imaginative look at the program and to explore strategic options for change. The Blue Team met during June-July 1992 and filed its report on July 15, 1992. The report contained findings, recommendations, and initiatives in three areas: (1) People, (2) Technical, and (3) Financial. (Hornstein et al., 1993)

At the heart of the technical initiative is a simplified representation and characterization of the space communications infrastructure that transcends organizational and functional boundaries, as well as existing systems and facilities. This simplified representation results from the Blue Team's discovery that the numerous and seemingly diverse infrastructure systems and facilities can be represented by only two functional categories. These categories are (1) **Information Handling** and (2) **Resource Management and Control.** Information Handling is the universe of activities associated with data/information receipt, processing (RF and digital), storage, retrieval, formatting, distribution, and transmission, including sensing of nominal and fault conditions. Resource Management and Control is the process of making decisions about which resources will be used for which activities at which times; control of operations; and assuring the allocation decisions are executed properly through all life cycle phases, including execution of recovery from unplanned events and circumstances, to satisfy operations goals and objectives.

The fulcrum of the simplified representation is the set of *"faster, better, cheaper"* metrics as adapted to fit the multi-mission, continuous nature of the space communications program. The non-conventional adapted metrics are **realistic, credible, responsive** (*"faster"* and *"cheaper"*), **simpler and smaller** (*"better"*) and are to be employed over both customer and infrastructure life cycles, rather than optimizing, for example, over the development phase to constrain initial development costs at the expense of the operations and maintenance phase.

The technical initiative is designed to reverse the trend of planning, designing, developing, maintaining, and operating costly one-of-a-kind systems and facilities. The initiative, as submitted on July 15, 1992, reads as follows: Create, evaluate, and select a wholly integrated operations concept, leading to an end-to-end systems architecture, with full participation of Code O service providers and customers. The concept is to be applicable across organizational and functional boundaries, and not limited to the in place infrastructure or configuration of existing systems and facilities. The evaluation factors and selection criteria will focus on customer satisfaction, life cycle effectiveness, and the adaptation of the *"faster, better, cheaper."*

The simplified space communications representation and adapted metrics are shown in Figure 1. In October 1992, the Code O Blue Team was reconstituted as the Code O Success Team (COST) to serve as a catalyst for changing the process from engineering systems to systems engineering. The focus of this paper is on laying the groundwork for process change, promoting the teamwork to accomplish change, and highlights of the FY 1993-1994 activities.

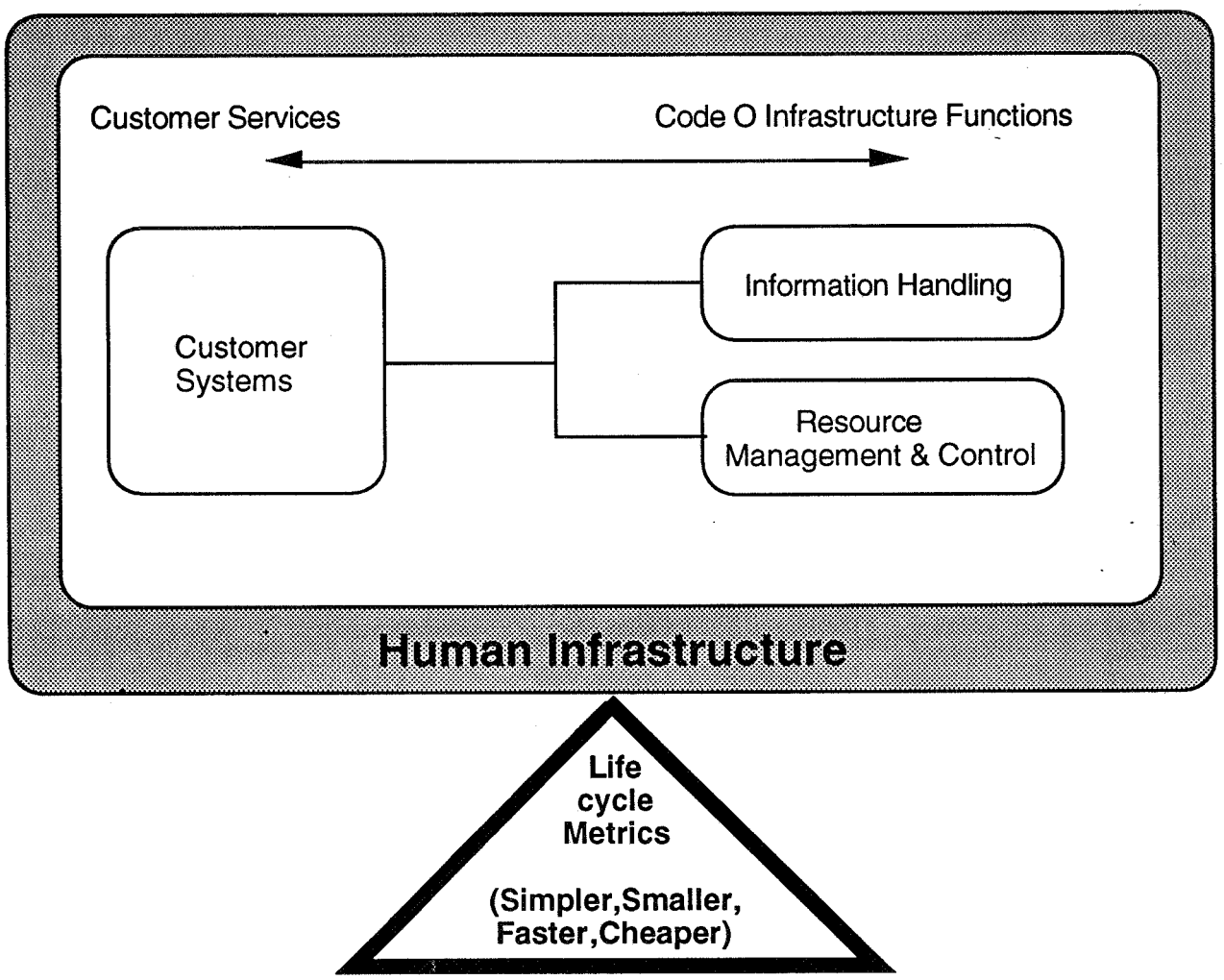


Figure - 1: Space Communications Representation and Metrics

2. LAYING THE GROUNDWORK FOR INNOVATION

In less than six weeks during June-July 1992, the Code O Blue Team examined a plethora of unique systems and facilities, all engaged in providing space communications services to space communications customers, some of whom are now labeled NASA Strategic Enterprises. The systems and facilities were unique in their names and titles,

organizations and architectures, technical components, and budget line items. To facilitate the examination of what appeared to be vastly distinct and divergent entities, the Blue Team diagrammed the systems and facilities to seek and formulate comparative relationships. However, substantive progress did not occur until the team stopped scrutinizing how the systems and facilities were engineered and started to question their purpose(s). A simplified representation of the space communications infrastructure then began to emerge. Through repeated examination of *similarities in purpose* vice *differences in engineering*, the Blue Team was able to group the space communications infrastructure systems and facilities into five functional categories at first, and finally to collapse them into only two functional categories: (1) **Information Handling** and (2) **Resource Management and Control.**

This discovery by the Blue Team (i.e., that the numerous and seemingly diverse infrastructure systems and facilities can be represented by only two functional categories) led to the recognition that there are considerable economies of scale to be gained and problems to overcome. We, organizationally, have become shaped by our emphasis on uniqueness. We tend to engineer systems rather than conduct systems engineering. This practice produces locally optimized, narrowly focused, and somewhat short-sighted solutions that contribute to overall infrastructure complexity through the accumulation of these many special solutions for similar purposes. Our heritage of unique solutions (i.e., systems and facilities) has fragmented our perspective and created barriers. One clearly visible barrier is language or vocabulary. Heretofore, when defining requirements for space communications services, the requirements have been described in terms of an implemented, or to be augmented, system or facility. This approach tends to limit the field of potential and available solution sets, and continues to perpetuate the proliferation of unique solutions. This entrenchment is adversely impacting mission operations through ineffective analysis of trade space alternatives. This entrenchment is not restricted to the space communications program, but occurs throughout NASA's strategic enterprises and functions.

A second key Blue Team finding or 'eureka' deals with the adaptation of the *"faster, better, cheaper"* charter to reflect the multi-mission, continuous nature of the space communications program. Further, this adaptation may be used to guide the improvement of customer services while accomplishing more efficient operations and infrastructure life cycle economies. The derivation of **realistic, credible, responsive, simpler and smaller** from *"faster, better, cheaper"* is as follows: The Blue Team began its examination with *"faster" and "cheaper."* Both concepts seemed obvious, i.e., do whatever in less time and with less money. Without giving it a second thought, 'doing whatever in less time' was related to accelerating system delivery schedules or receiving data more rapidly at higher rates. On second thought, *"faster"* was less obvious. *"Faster"* was meaningful only in the context of being responsive to customer needs. Delivering a system capability two years prior to need (e.g., launch or encounter) may not be advantageous if the implementation of that capability creates restrictions in current operations or introduces additional costs in maintaining and operating the capability. *"Cheaper"* was looked at in terms of the agency flight program model and the multi-mission, continuous nature of an infrastructure model. Using the flight program model, costs are tracked, on a per mission or spacecraft basis, from beginning to end or womb to tomb. However, an infrastructure (now labeled as strategic functions in the NASA Strategic Plan) spans multiple missions and spacecraft. Economies are achieved by leveraging the needs of multiple customers and accommodating these needs through modifications to the infrastructure. The Blue Team probed and found that emphasis was placed on engineering systems, on a mission by

mission basis, with restrained and controlled implementation costs. Although economies were gained, the results were sub-optimal when one considers that the majority of a system's life cycle is spent in the maintenance and operations phase, not in the design and development phase. Investigating the meaning of *"better"* proved to be both enlightening and revealing. In the minds of many good engineers, *"better"* readily translates to more whiz bang, e.g., state of the art, advanced technology, enhanced performance, and inevitably more complexity. However, complexity often translates into cost and schedule risk throughout a system's life cycle.

From these deliberations and non-conventional exchanges of views, it became evident that *"faster, better, cheaper"* was being discussed in terms of their units of measure. Traditional units of measure (i.e., time and dollars) were being used for *"faster" and "cheaper,"* but non-traditional units of measure (i.e., complexity and size) had surfaced for *"better."* In all cases, value had to be measured across life cycles, both customers and service providers alike. Value for *"faster and cheaper"* became **realistic, credible, responsive**. Value for *"better"* became **simpler and smaller**.

The two Blue Team discoveries described in this section -- (1) that the numerous and seemingly diverse infrastructure systems and facilities can be represented by only two functional categories and (2) that *"faster, better, cheaper"* is more appropriately portrayed as **realistic, credible, responsive, simpler and smaller**, across life cycles -- constitute a new working paradigm for preparing and delivering space communications services. This new paradigm strongly suggests that we depart from our legacy of engineering systems to establish the practice of true systems engineering. Additionally, it must be acknowledged that focusing on similarities, rather than on differences, expands the solution set for achieving economies of scale, while creating opportunities for reducing infrastructure complexity.

3. PROMOTING TEAMWORK TO ACCOMPLISH CHANGE FY 1993

In October 1992, the Code O Blue Team was reconstituted as the Code O Success Team (COST) to serve as a catalyst for change. The COST role was further clarified to include complementing line management and fostering cooperation (not competition) across the Code O Family. In addition, the COST accepted the challenge to include a tactical emphasis in its planning and to maintain a balance between this tactical emphasis and the strategic (1) People, (2) Technical, and (3) Financial Blue Team initiatives. Consistent with this clarified role, the COST set out to:

- Promote cooperation and collaboration across traditional boundaries
- Nurture innovation by seeking ideas for near term across-the-board opportunities for savings
- Advocate **realistic, credible, responsive, simpler and smaller** solutions
- Actively solicit customer participation
- Encourage use of the simplified space communications representation and adapted metrics in evaluating system implementation projects

The COST entered into dialogues within the Code O Family of service providers and customers at Headquarters and the field centers. These dialogues disclosed that many of us were in violent agreement on the need to change. While we recognized that we shared a heritage of success, we also acknowledged that the challenge was to reduce the cost of success. This challenge, "to reduce the cost of success" became the hallmark of the FY 1993 activities.

The FY 1993 activities included strategic and tactical elements. Formulation of the Code O Family vision was a strategic endeavor to integrate the simplified space communications representation, the adapted metrics, the Blue Team initiatives, and the challenge "to reduce the cost of success." The one-page statement and illustration was signed by the Associate Administrator for Space Communications in June 1993. The Code O Family vision statement is to reduce the cost of success through (1) increased cooperation, (2) improved service to and partnership with customers, and (3) decreased cost and complexity of the space communications infrastructure.

Two tactical activities were conducted during FY 1993. They centered on remaining viable in a constrained fiscal environment, while pursuing the vision. The COST organized and hosted the first Space Communications Programmatic and Tactical Planning Workshop. Attendees included space communications service providers and their customers. There were two key objectives:

- Develop a technical foundation and teamwork base to facilitate work on operations concepts
- Develop an investment strategy that included areas for near term savings

The workshop teams presented their results and recommendations to Code O Senior Management, including the Associate Administrator. The recommendations were synthesized into six areas for tactical savings and incorporated in the budget guidelines. In retrospect, the workshop set the stage for teaming across organizational boundaries and for testing the value of proposed infrastructure modifications in terms of customer benefit.

Also incorporated in the budget guidelines, was the announcement of the Code O Investment Program. This program provided an open channel for the field centers to submit proposals that reduced costs, improved services to customers, or otherwise contributed to making the Code O infrastructure of systems and operational services simpler, cheaper, or more customer friendly. The proposal selection criteria were based on (1) Contribution to the Code O Family vision and (2) Rapid Investment Payback - within three years. The Code O Investment Program was intended to build on the teaming started at the workshop and create near term across-the-board opportunities for savings. In retrospect, it was difficult to shed local institutional perspectives. Only ten tactical winners were selected from sixty-three proposal abstracts and twenty full proposals.

4. REDUCING THE COST OF SUCCESS FOR NASA's MISSION OPERATIONS FY 1994

Early in the year, the COST determined that going after strategic and tactical results concurrently was not leading to the desired state. Line management was proceeding with business as usual, i.e., working within organizational boundaries and motivating their employees to do the same. Meanwhile, circumstances had overtaken the tactical program.

Process changes to realize substantive economies of scale would have to be strategic, and technical innovation to reduce cost and complexity would have to be strategically motivated. Further, "reducing the cost of success" actually meant reducing the cost of success for mission operations for NASA's strategic enterprises and strategic functions. With active and intense involvement by the Associate Administrator for Space Communications, the COST role was broadened to encompass agency mission operations.

The COST organized and hosted a mission operations workshop. Participants were a non-parochial cross section of mission operations experience from NASA, industry, academia, and another government organization. They had been selected using the same criteria used to select members of the initial Code O Blue Team:

- Independent Thinkers, yet Team Players
- Recognized for Technical Expertise and Professional Integrity
- Prepared to Challenge the Status Quo
- Able to Resist Engineering the Solution Before Understanding the Problem
- Experience in engineering systems and Systems Engineering; able to distinguish
- Willing to be Unconstrained, Non-parochial, and Imaginative
- Empowered to Explore Strategic Options rather than Producing a Quick Fix
- Perseverance and Commitment Post-Workshop (1-year continued teaming)

The workshop theme was NASA Teaming Across Organizational Boundaries to Reduce the Cost of Success for End-to-End Mission Operations. The workshop goal was to expressly begin work on changing the culture to stimulate innovation and promote cooperation and collaboration across traditional boundaries for the good of NASA. The workshop was to be considered as the kick-off meeting for building relationships and creating teaming arrangements to step up to the challenge of reducing life cycle costs for NASA mission operations. The principal workshop objective was to articulate a common baseline for services and functions necessary to conduct end-to-end mission operations. In order to emphasize *similarities in purpose* vice *differences in engineering*, the descriptions of these services and functions were to be independent of existing systems, facilities, technologies, organizations, and personalities.

The workshop announcement stated that many of us believe the big payoff will come from reversing the trend of engineering special solutions for similar problems, through the identification, development, and deployment of reusable components that simplify engineering (building and maintaining) and operating systems for end-to-end mission operations. Achieving an agreed to baseline of services and functions was seen as a mandatory first step on the road to payoff.

The opening session of the workshop was a dialogue between the participants and the NASA Administrator. His presence reinforced the priority of cooperating and collaborating across organizational boundaries. At the conclusion of the session, he invited workshop representatives to continue the dialogue at the next Senior Management Meeting to be held June 9, 1994. The invitation was accepted.

Preparation for the Senior Management Meeting energized the representatives to form a non-standard alliance of **C**ode **O** **S**uccess **T**eam/**L**ifecycle **E**ffectiveness for **S**trategic **S**uccess (COST LESS) for Mission Operations. This alliance established the following goals, technical approach, and people process:

Goals
Redefine Success in a Constrained Fiscal Environment
Reduce the Cost of Success for End-to-End Mission Operations

Technical Approach
Reverse the Trend of Engineering Special Solutions for Similar Problems

People Process
Break Down Barriers and Team Across Traditional Boundaries

The alliance presented to NASA Senior Management that the goals would be met, and significant savings could be realized by improving processes and incorporating them into the line organizations. The COST LESS for Mission Operations alliance also reported that "across traditional boundaries" included life cycles, functions, programs and projects, as well as organizations. The effort envisioned would be multi-dimensional and multi-disciplinary in order to achieve example

results such as (1) Common Vocabulary, (2) Reusable Solutions to Simplify Engineering and Operations, and (3) Operations Concepts to Maximize Value.

5. NEXT STEPS FOR FY 1995

In the NASA Senior Management Meeting of June 9, 1994, the Administrator noted that the key to success is the cross-cutting nature of the COST LESS team for Mission Operations. It allowed the group to review NASA objectively versus as individual organizations. With this endorsement, the team is reconvening during August-September 1994 to prepare for the next steps between NASA's strategic enterprises and functions.

6. REFERENCE

Hornstein, R.S. et al. (October 1993). A Systems Engineering Initiative for NASA's Space Communications. Proceedings of AIAA Computing in Aerospace 9 Conference (AIAA-93-4696) (pp. 1282-1289). San Diego, CA.

111171 N95-17586

35427/ p-7

A System Study for Satellite Operation and Control in Next Generation

K. Nakayama, T. Shigeta, / Tracking and Data Acquisition department, National Space Development Agency of Japan

T. Gotanda, K. Yamamoto, Y. Yokokawa / LTCB Systems Co.,Ltd

1. Abstract

Ever since the first satellite, ETS-1, in 1975, 28 NASDA satellites in total have been launched. With regards to satellite operations, NASDA has developed realtime TLM/CMD processing systems which could be commonly used for different types of satellite. Presently the third generation system is operational. Meanwhile, the recent trend of satellite operations is becoming more complicated, for example, CCSDS-adapted Satellites are emerging and computer technology is developing quite rapidly. Moreover, NASDA's role in satellite operations is changing from mainly Satellite Bus operations to experimental/whole satellite mission operations. Considering these circumstances, NASDA has initiated a study for the next generation system which is suitable for operations of future satellites keeping in mind the following viewpoints.

- Demands from mission support
- Trend of satellite design
- Progress of computer environment

This is an interim report of the study.

2. NASDA's Present system

2.1 Tracking and Control System

The present Tracking and Control System is shown in Fig.-1. It consists of the followings:

- Network system(tracking stations, network control, etc)
- Satellite Operation and Control system(TLM and CMD operation)
- Support system(orbit determination,planning of operation,etc)
- Space Network system(network via Data Relay Satellite; experimental)

2.2 System Configuration of Satellite Operation and Control system

NASDA's Satellite Operation and Control system, based upon the "system applying to all satellites", consists of the following elements.

- Satellite system
  The realtime on-line subsystem and off-line subsystem which are composed of satellite functions and satellite unique functions.
- Database Manager
  Single system, common to all satellites, to manage database for parameters.

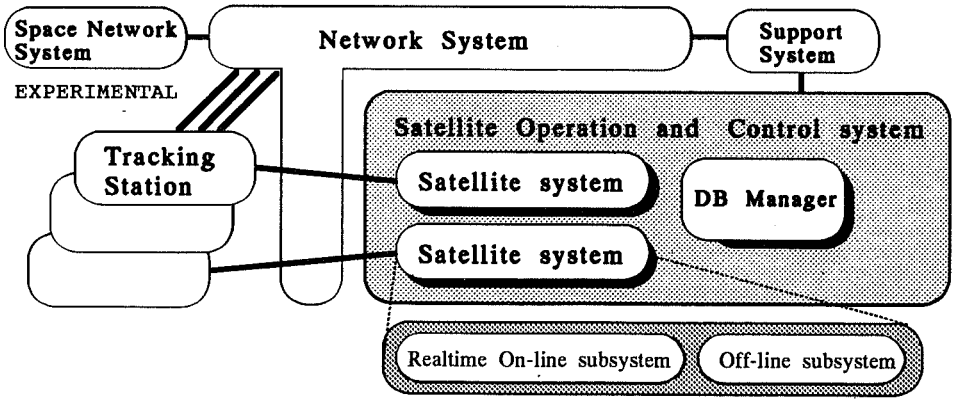


Fig.-1 NASDA Tracking and Control System

3. Objectives of the Study

In this section, necessity of the next generation system (named "SOC-X") is introduced, addressing the following viewpoints:

- Demands from mission support
- Trend of satellite design
- Progress of computer environment

3.1 Demands from mission support

- Supporting mission

The present system is made up and used for satellite operations without making clear distinction of housekeeping operation or activity of supporting missions.
Hence, NASDA needs a new concept for SOC-X, which is "Providing interface to support mission" on user's side.

- Variety of missions

The present systems, based upon the "concept applying to all satellites", have difficulty supporting various missions entirely, because of a shortage of mission support functions.

A variety of support functions as well as further study on how to implement the functions on the common system are required.

3.2 Trend of Satellite design

- Accommodation to CCSDS

NASDA satellites are adapting the CCSDS recommendation. Therefore, SOC-X is requested to support the corresponding CCSDS communication environments. SOC-X will deal with both conventional and CCSDS satellites.

- Satellite autonomy and autonomous operation

High performance and autonomy of onboard equipments will change the tasks of satellite operations on ground as follows.

- Simplification of Command data generation scheme
- Monitoring equipments through onboard autonomous supervision function in parallel with the conventional monitoring of all telemetry information

In addition, operating procedures will change over to the new way from ground-based actions to onboard autonomous control.
NASDA is in now a transition phase, and it is necessary to cooperate with satellite design.

3.3 Progress of computer environment

Computer environment is rapidly advancing more than the time the present system was designed.

- Progress in computer technology includes:
  - Enabling more complicated process
  - Minimizing the cost of computer
  - Displaying high value added information to the operator
  - Improving man-machine interface

- Progress in network environment
  - Network technology of LAN
  - Use of common resources and distributed function via LAN
  - High speed WAN, which is suitable for LAN
  - Standardization of LAN environment

4. Consideration of SOC-X

4.1 Concept for satellite operation

Cooperative operation with the mission users becomes more important in the future satellite operations. Moreover, satellite operation not only housekeeps the satellite, but also provides functions to utilize satellite payloads to the mission users. Apparently, it is required to be more "mission " and "end-user" oriented.

4.1.1 Definition of SOC-X

SOC-X directly interfaces with satellite according to the concept of "mission" and "end-user oriented". It could be defined as a "Provider of data between satellite and mission users". SOC-X provides mission support functions as described below and in Fig.-2.

- Providing operation environment to users for housekeeping of payload and mission execution
- Controlling the satellite safety

4.1.2 Providing mission environment

The mission operating environment provided by SOC-X is for the housekeeping of satellite resources, including mission equipments, and for the real-time mission data interface between the satellite and users.

- Housekeeping the satellite resources

There are many items of resource to be managed on satellite. These resources are controlled by SOC-X while mission users would operate their equipment under this controlled environment.
In this configuration, mission users are allowed to control only each mission part individually.

- Data interface with satellite

Data interface function provided by SOC-X is the real-time data transfer of TLM and CMD including transparent data interface, engineering data conversion of TLM, and generation of CMD.
However, it is undesirable for SOC-X to have direct interface with the user, in terms of system security and satellite safety. Accordingly, it becomes important to provide a system which satisfies the users' requirements and protects the Satellite Operation and Control system.
NASDA is considering a method for user interfaces taking into account the above aspects, which also include non-realtime data interface.

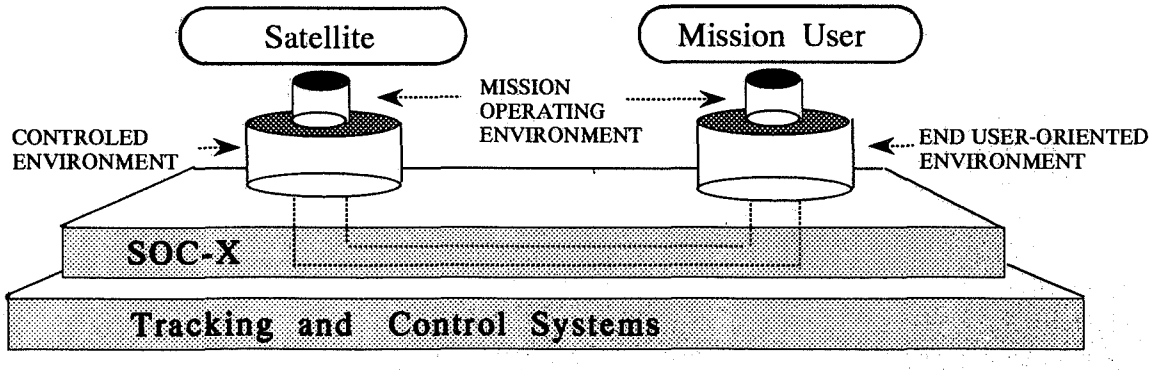


Fig.-2 Environment provided by SOC-X

4.1.3 Satellite Safety Operation

SOC-X must provide protection to satellite resources.

- Supervising satellite status

The present operation takes necessary recovery action after detecting of anomaly by monitoring TLM. However, "prediction of anomaly by inference" might be the goal. This requires the satellite operation planning information in order to know how the resources will be used.

- Command control of satellite

Here, the validity of command and timing of its transmission are verified. This process also takes into account the conditions of satellite resources. However, to what extent and method of these checking should be used by SOC-X needs further study.

4.2 Satellite design trend

4.2.1 Approach to Mission-oriented and Commonality

The important aspect of this study is to overcome the contradiction between diversification of mission and commonality of satellite operation. Needless to say, these studies must be made with a close cooperation with satellite design and development.

4.2.2 Adaptation to CCSDS

CCSDS causes no big impact on Satellite Operation and Control system, however, it brings about some new concept of communication protocol between the satellite and ground.
In NASDA cases, the responsibilities of the following two subsystems in terms of CCSDS environment are unclear;

- Network system is responsible for the control of all equipments on ground and all systems of tracking station.
- Satellite Operation and Control system is responsible for the control of the satellite system.

Since it is still new for NASDA to develop the CCSDS equipments for onboard and ground, an issue on how to design s system most suitable for CCSDS environment was brought up.

A conclusion has not been reached, but the scope of new Satellite Operation and Control system must conform to the concept of the CCSDS standards.

4.2.3 Automatic and autonomous

Onboard subsystems will be more automatic and autonomous for high level mission operation, which result in reduction of operational load. These will cause a change in operation leadership on the ground so it is important to assign responsibilities to both onboard and ground systems respectively.
The operational load will be reduced by automatic and autonomous satellite operations, but the task of SOC-X increases because of the followings;

- Analysis on the cause of anomaly and countermeasure actions executed by autonomous onboard programs
- Verification of pre-arranged actions in automatic and autonomous onboard programs, prior to its execution
- Control of automatic and autonomous programs and supervision of the operating status
- Backup for onboard CPU failure

4.3 Utilization of new technology

4.3.1 LAN-based system

A use of new technologies, which were not available at the time of the present system design, is one of the key issues of this study. A LAN-based system is one of the concepts for SOC-X, which supports activities as follows.

- Common use of resources
- Distributed processing
- Reduction of duplicated functions

4.3.2 Reduction of operator's load

One of the subjects for reducing operational load is the telemetry monitoring. Currently the operator needs plenty of information to judge satellite conditions, thus it is quite important to develop a method where the system can predict anomaly. Artificial Intelligence(A/I) technology may help reduce this task.

4.3.3 Upgrading the operational environment

It is most desirable to provide more practical and condensed information to the operators. One of them is to utilize the latest computer technologies. It would provide a suitable operation environment to support NASDA's mission by:

- Visual information
- Multi Role terminal
- Unified man-machine interface

5. Summary of the study result

This section is an interim report about the concept of SOC-X.

5.1 Concept of SOC-X

NASDA's Tracking and Control system, now functioning only as the operator of a satellite, would also become a provider of satellite mission support environment.
In this concept, SOC-X would provide the following environment to mission users:

- Mission-oriented operational environment
- User-oriented interface environment
- Satellite configuration management

5.2 Element of SOC-X

5.2.1 Concept

The satellite operations consist of housekeeping and mission operations. Hence, the Satellite Operation and Control system is provided with Bus-control and Mission-control functions.

5.2.2 Bus-controller and Mission-controller.

Bus-controller(B-ctrl), which is a system applying to all satellites, is for the operation of whole satellite and bus equipment. B-ctrl is a front-end system to manage the whole satellite.

Meanwhile,Mission-controller (M-ctrl) is a customer-made system for each mission operation. M-ctrl has TLM and CMD functions for specific missions, and a back-end system which performs the housekeeping for mission equipment and control of mission equipments.

5.2.3 Interface between B-ctrl and M-ctrl

Telemetry data

In case of copious telemetry from a conventional satellite, B-ctrl delivers necessary telemetry to M-ctrl. With regards to the CCSDS cases, its telemetry is distributed to B-ctrl and M-ctrl individually from the CCSDS telemetry handling systems, but exchange of information between them would be done as follows:

- B-ctrl provides information needed to proceed the mission.
- M-ctrl provides information needed to manage the whole satellite system.

Command data

Mission commands from users are generated and checked by each M-ctrl for payload safety. At this point, satellite safety is managed between B-ctrl and M-ctrl by exchanging information.
In case of conventional satellite, commands generated by each M-ctrl would be transmitted through B-ctrl. In other cases, for example, the distributed commanding of CCSDS, B-ctrl and each M-ctrl generate and transmit their individual commands under planned conditions.

5.3 Mission Support

5.3.1 Service environment

SOC-X provides the management function of satellite configuration. This configuration is independent of other missions and bus operation, and it assures the mission users with operation safety.
Furthermore, operations including the housekeeping of onboard mission equipment by B-ctrl is also possible.

5.3.2 End-user oriented service

Further services are considered:

- Data transfer
  Realtime communication with satellite(e.g.:TLM,CMD)
- Value added Data transfer
  - Delivery of TLM, which is converted to engineering data
  - Generation of CMD, which is converted from a descriptive information to actual command data
- Rental terminal

5.3.3 Interface Configuration

For the interface configuration provided to the users, are there three types as follows and also shown in Fig.-3:

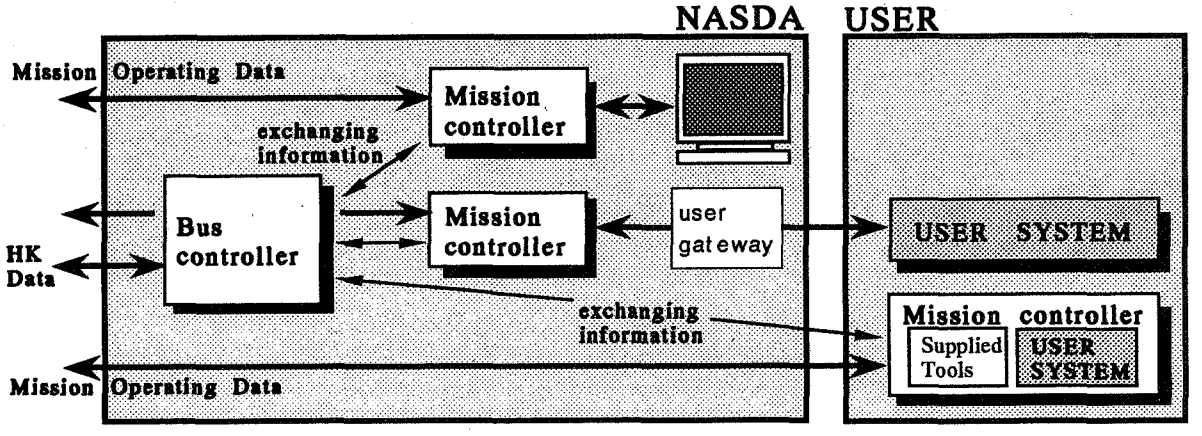


Fig.-3 Interface configuration (CCSDS type satellite, for example)

* **Utilization of NASDA supplied tool**

This tool supplied by NASDA, is provided with functions necessary to perform the mission operations for the users.

* **Access to Data GW system**

This Data GW system supplied by NASDA, functions as a data gate-way for the satellite and user's system, and links these two to perform mission operation.
The system provides the basic functions to relay the processed telemetry data to user's system and convert a descriptive information from the user to actual command data.

* **Development of User Own System**

A mission user develops each system by utilizing the functional tools supplied by NASDA.
As the user integrates these tools in his system, he is able to perform a mission control on his own.

6. Future subject

6.1 Future Study

This paper introduced an interim study result.
So far, several issues have been identified. We, for example, have some study items with the network system, and a certain extent of responsibility for SOC-X in terms of "Basic concept of NASDA satellite operation in future".
Indeed, the key element in developing system is to keep in mind conformity of satellite design and ground systems.
We hope to write a final report after having the major problems examined.

6.2 Development schedule

The current schedule is not yet approved. So far, the following schedule is being considered.

Fy	
1994	Conceptual Study
1995	System Study
1996	Preliminary Design
1997	Detail Design
1998	Development
1999	Integration Test

Systems Engineering

2. Reusable Systems

* *Presented in Poster Session*

PRECEDING PAGE BLANK NOT FILMED

N95-17587

111172

354273

TRANSPORTABLE PAYLOAD OPERATIONS CONTROL CENTER REUSABLE SOFTWARE: BUILDING BLOCKS FOR QUALITY GROUND DATA SYSTEMS

p. 9

Ron Mahmot[1] , John T. Koslosky[1] , Edward Beach[2] , Barbara Schwarz[3]
[1] NASA/Goddard Space Flight Center, [2] Computer Sciences Corporation (CSC), [3] Integral Systems Inc.

ABSTRACT

The Mission Operations Division (MOD) at Goddard Space Flight Center builds Mission Operations Centers which are used by Flight Operations Teams to monitor and control satellites. Reducing system life cycle costs through software reuse has always been a priority of the MOD. The MOD's Transportable Payload Operations Control Center development team established an extensive library of 14 subsystems with over 100,000 delivered source instructions of reusable, generic software components. Nine TPOCC-based control centers to date support 11 satellites and achieved an average software reuse level of more than 75%. This paper shares experiences of how the TPOCC building blocks were developed and how building block developer's, mission development teams, and users are all part of the process.

1. INTRODUCTION

The TPOCC is a control center architecture which takes advantage of workstation based technology to improve mission operations and reduce development costs for Payload Operations Control Center's and Mission Operations Center's in GSFC's MOD. The TPOCC architecture is characterized by it's distributed processing, industry standards, commercial off-the-shelf (COTS) hardware and software products, and reusable custom software. The reusable TPOCC software is integrated with mission applications to provide the health and safety monitoring, and commanding capabilities for various NASA satellites. This includes the capability to process and display telemetry, build and send commands, and perform special processing. In addition, TPOCC provides a graphical user interface and a procedural command language that automates ground system and spacecraft control.

The TPOCC development team established an extensive library of 14 subsystems with over 100,000 delivered source instructions of reusable, generic software components. By encapsulating the basic control center functionality into reusable building blocks, the control center design and implementation becomes a much easier task. The nine TPOCC-based control centers to date support 11 satellites and achieved an average software reuse level of more than 75% (see Table 1). The challenges involved in establishing and maintaining a cost-effective library of software components for ground system development include: making the initial investment, getting mission teams to adapt a reuse paradigm, configuration control, and staying current with technology.

This paper describes the MOD's approach to establishing and maintaining the TPOCC reusable software.

2. BACKGROUND

NASA's Goddard Space Flight Center (GSFC) Mission Operations Division (MOD) provides

PAGE 1160 INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

Ground Support Systems for a variety of scientific satellites. The MOD designs, implements, tests, and delivers control centers - both traditional Payload Operations Control Centers(POCCs) and the newer, functionally expanded Mission Operations Centers (MOCs) - that provide command management and mission planning functions. Using an operations control center, the Flight Operations Team (FOT) provides around-the-clock support to its spacecraft, typically establishing communications contact every few hours for a period of about 20 minutes. During these brief contacts, the control center system functionality allows the FOT to quickly evaluate the current condition of the spacecraft and transmit the commands necessary to maintain its health and control the spacecraft and its science instruments.

Recognizing the many similarities among missions, the MOD always has made software reuse a priority. Prior to TPOCC, the most successful reuse effort was the Multisatellite Operations Control Center (MSOCC) Applications Executive (MAE), which achieved a maximum software reuse level of about 40%.

The TPOCC project began in 1985 as a Control Center System Branch research and development effort examining the new workstation based technology to further reduce mission development cost and improve operations support. The ICE / IMP Control Center was successfully prototyped using a TPOCC system in little more than six months. Following this proof of concept POCC software was developed for the SAMPEX, WIND / POLAR, and ICE / IMP missions simultaneously with the implementation of the TPOCC core reusable software. The TPOCC core reusable control center software was completed with the delivery of TPOCC release 6.3, in July 1992, and successfully supported the SAMPEX launch. In addition to the numerous MOD control center applications which TPOCC now supports other non-control center applications utilize TPOCC software to reduce their development costs (see Table 1). TPOCC software continues to evolve based on mission needs and new technology opportunities.

TABLE 1 - APPLICATIONS INCORPORATING TPOCC

CATEGORY	APPLICATION
NASA/GSFC Control Centers:	International Cometary Explorer (ICE) / Interplanetary
	Monitoring Platform (IMP) Payload Operations Control
	Center (POCC), Interplanetary Physics Laboratory
	(WIND) POCC , Polar Plasma Laboratory (POLAR)
	POCC, Solar and Heliospheric Observatory (SOHO)
	POCC, Solar Anamalous & Magnetospheric Particle
	Explorer (SAMPEX) POCC, Fast Auroral Snapshot
	Explorer (FAST) POCC, Submillimeter Wave Astronomy
	Satellite (SWAS) POCC, X-Ray Timing Experiment (XTE)
	Mission Operations Center (MOC), Tropical Rainfall
	Measuring Mission (TRMM) MOC, Advanced
	Composition Explorer (ACE) MOC
Alenia Spazio Control Center:	X-SAR Mission Planning Operations System
NASA/GSFC Spacecraft Operations Tools:	Shuttle Payload Interface Facility (SPIF),

NASA/GSFC Spacecraft Operations Tools:	Generic Trend Analysis System, Generic Spacecraft
	Analysist Assistant, Ground Operations Technology
	Testbed,TPOCC Orbital Signature Analyzer, Space Views,
	FAST Level Zero Processor (LZP), Real time Attitude
	Display System (RTADS), Heads Up Display (HUD)
NASA/MSFC Spacecraft Operations Tools:	Gravity Probe B Project
NASA/GSFC Spacecraft Simulators:	TPOCC Advanced Spacecraft Simulator (TASS) (TASS
	systems have been tailored and delivered to WIND,
	POLAR, SOHO, FAST, SWAS, XTE, and TRMM)
	SOHO Simulator, XTE Simulator
NASA/GSFC Science Center:	SOHO Experimentor Operations Facility
Cal Tech Science Center:	ACE Science Operations Center (SOC)
NASA/GSFC Spacecraft Integration and	
Test (I&T) Facilities:	SAMPEX I&T, FAST I&T, XTE I&T
Martin Marietta Spacecraft I&T Facility:	WIND/POLAR I&T
Johns Hopkins University Applied Physics	
Laboratory (APL):	ACE I&T

3. BASELINING THE BUILDING BLOCKS

Prior to baseling TPOCC software and committing to mission deadlines an investment was made in a system development concept, technology evaluation, and prototyping POCC subsystems. The System development concept resulted in a series of mandates for both the architecture and the management approach for mission's which will use the TPOCC architecture.

The management approach taken on TPOCC was to institutionally fund and manage the TPOCC reusable software development and architecture. Each mission would then fund an effort to build a dedicated control center based on TPOCC building blocks integrated with mission specific software. Enhancements to the core reusable software would be evaluated on a case by case basis using project funding when necessary.

There were four key TPOCC architecture mandates: distributed processing; widely accepted industry standards; reusable software with clearly defined interfaces; and commercial off-the-shelf (COTS) products.

TPOCC adapted distributed processing as a means to scale hardware and software to the requirements and complexity of a specific mission. Implemented through a client/server method, functional data processing capabilities are developed as independently executing software tasks distributed across a group of networked inexpensive heterogeneous computers.

Although all TPOCC software components can execute on a single workstation, the current generation of TPOCC based control centers are hosted on a heterogeneous set of processors to meet mission data throughput needs. The main hardware components are a front end processor (FEP) and multiple UNIX workstations connected to an Ethernet-based local area network (LAN). The FEP, a VME chassis containing one or more single-board computers using a real-time UNIX-like operating system, performs the time-critical control center functions. The VME bus

provides high data throughput between the processes hosted within the FEP, while reducing data volume on the external LAN. The FEP also contains mass data storage, as well as additional NASA communications (NASCOM) interface hardware for receiving spacecraft telemetry data and for sending commands and data to the spacecraft.

Another major contributor not only to software reuse across control centers but also to the extendibility of TPOCC to other non-control center applications is its clearly defined and tightly controlled interfaces. Because all TPOCC based systems produce software that complies with these interfaces, capabilities developed by mission development teams can be easily shared, either by subsuming the new capability into the TPOCC reusable software base (if the capability is truly generic) or directly into other applications that require the specific capability. These well defined interfaces also make it possible for TPOCC compliant applications to share data. TPOCCs defined data service protocol allows a loosely coupled application to link into a TPOCC based system, access its telemetry stream, receive updated parameter values in real-time, act on that data in its own software, and return data to the TPOCC based system for display.

Adhering to widely accepted industry standards was seen as an essential part of TPOCCs approach to building a software library of components which could be compatible with advances in hardware technology. The standards used for TPOCC hardware components include VME, Ethernet, RS 232, RS422, and SCSI. TPOCC software standards adheres to open systems communications standards such as the Transmission Control Protocol/Internet Protocol (TCP/IP), external data representation (XDR), and network file system (NFS). TPOCCs user interface standards include the X-Window System and the Open Software Foundation(OSF)/MOTIF software.

Following widely accepted industry standards also puts TPOCC in a position to maximize the use of COTS products wherever possible. Graphical User Interface, database, development tools, and network management are a few of the areas where TPOCC has utilized COTS products in lieu of costly in house development efforts. As a result, system enhancements have become more dependent on COTS vendors, making vendor reliability an important aspect of technical evaluations.

A significant amount of time was spent on doing technical evaluations and prototyping in the early phases of the TPOCC project. High risk requirements were prototyped and workstation based hardware and software components were evaluated for its application to the control center environment. Prototyping continues to play an important role in evolving the TPOCC reusable software base with technology.

4. ESTABLISHING A REUSE PARADIGM AMONG MISSION TEAMS

System development based on TPOCC software building blocks and TPOCC architecture introduced technical and management challenges. Initially, TPOCC development was localized within a research group and focused primarily on developing concepts, identifying applicable technology, and prototyping high risk areas. Mission POCC development teams were accustomed to developing, implementing, and testing each POCC individually using FORTRAN on a IBM or DEC like minicomputers. Both the TPOCC development group and mission development teams needed to change their perspective. Mission developers could no longer view themselves as developers of separate and

unique control centers but rather as contributors to the larger body of common control center software.

The first step toward this paradigm shift was to introduce the mission development teams to the principles and mindset necessary to make large-scale subsystem reusability a reality. The TPOCC development group conducted classes on the TPOCC development methodology, system design, and system capabilities. Also development teams attended vendor training classes on the new technology and software implementation environment. The third method of training was having the mission teams build prototype POCC systems utilizing configured TPOCC generic subsystems and adding their mission specific code. The complete telemetry processing path was prototyped for SAMPEX & WIND/POLAR in approximately two months .

Communication and feedback between the TPOCC staff and mission development teams becomes increasingly important as the number of TPOCC-provided capabilities and the number of TPOCC-based applications increase. A Configuration Review Board (CRB) and working group, with a variety of representatives including mission development teams(contractors and GSFC personnel for each mission), the TPOCC staff, users, and representatives of other (i.e. non-control center) TPOCC based systems was established. The working group ensures that everyone is kept abreast of the efforts, needs and schedule of the collective group. As mission development teams identify candidate capabilities for generic implementation, the CRB determines which candidate capabilities are truly generic, schedules their implementation, and provides configuration control.

As the number of missions increase, requirements for new capabilities grew but the TPOCC budget has remained constant. In order to make this effort successful while controlling cost, temporary teams with members from the TPOCC group and mission development teams are formed to design and implement new generic software. In addition to providing additional staff to implement generic capabilities, TPOCC staff gets insight into what portions of a capability are generic, and help identify likely differences among missions. For the missions, these teams provide insight into designing software for reusability and experience in differentiating the mission-unique elements of a problem from the generic.

Using TPOCC software and architecture framework, mission development teams have also been able to support unique needs of individual spacecraft requirements. The WIND/POLAR POCC, supporting two spacecraft scheduled to launched within a year, handles two physical channels of time division multiplexed (TDM) telemetry, while other TPOCC missions use the CCSDS standard. The SOHO POCC, scheduled to launch in mid-1995, must handle a unique combined TDM and CCSDS telemetry format.

To date, six TPOCC-based applications systems have been configured to support eight different spacecraft. The first three POCCs developed for the Small Explorer (SMEX) program, SAMPEX, FAST, and SWAS, also serve as a case study in determining the levels of reuse that can be attained with the TPOCC architecture within a series of missions. The SAMPEX POCC consists of COTS products, 67% TPOCC-generic software and 33% mission-specific software (see Table 2). By reusing both SAMPEX mission-specific software and the expanded TPOCC-generic software, the FAST POCC achieved an 81% reuse level. The SWAS POCC, expanding this

reuse base to include FAST mission specific software, achieved a reuse level of 92%.

Two latter missions, XTE and TRMM chose a TPOCC-based-MOC approach, integrating the traditional POCC, mission planning, and command management functions together into one system by sharing functionality and reusable software which had previously been implemented separately. Since the TRMM and XTE spacecraft have significant similarities, much of the mission specific code in XTE will be reusable in TRMM.

Another indicator of TPOCC reuse across missions is the number of new generic capabilities asked for by mission development teams and users. Totals by mission of new generic capability requests, documented as Internal Configuration Changes Requests (ICCRs), are listed in Table 2.

TABLE 2 - NASA/GSFC Control Center TPOCC Reuse Data

Project Name	Mission Specific DSI*	Reused DSI	Total System DSI	Percent Reuse	Total*** Generic Change Requests(ICCRs)
ICE/IMP**	25407	72552	97959	74%	17
ISTP series: WIND	43372	81895	125267	65%	23
POLAR	7414	124934	132348	94%	0
SOHO	43690	94840	138530	68%	8
SMEX series: SAMPEX	38308	77125	115433	67%	16
FAST	22707	96534	119241	81%	11
SWAS	9800	114434	124234	92%	1
XTE	133340	128520	261860	49%	14
TRMM	14450	235360	249810	94%	2

*DSI = Delivered Source Instructions

**Following its initial prototype, ICE/IMP was completely rehosted

***Number of approved ICCRs from TPOCC Release 2 through Release 12

5. CONFIGURATION CONTROL

The TPOCC CRB oversees the reusable software configuration management as defined in the TPOCC Project Support Plan. Because the TPOCC Reusable Software forms the core of several control centers, configuration control is necessary across development efforts and it is essential that:

- Proposed changes [i.e., internal configuration change requests(ICCRs)] to the TPOCC software be visible to TPOCC missions before implementation and that TPOCC applications be given the opportunity to analyze the proposed changes and assess the impact to the TPOCC applications
- Proposed changes [i.e., configuration change requests (CCRs)] to mission software be visible to the TPOCC project and other application teams for analysis, and possible reusable implementation recommendations.
- Problems detected during testing of the TPOCC release be documented as TPOCC internal discrepancy reports

(IDRs) and made visible to all TPOCC applications
- Problems detected during testing of TPOCC applications be documented as discrepancy reports (DRs) and made visible to all TPOCC applications
- DRs written against TPOCC applications are analyzed; assigned to either generic TPOCC or a mission development team based on the results of that analysis; and if assigned to the generic TPOCC, made visible to all TPOCC applications.
- The reusable TPOCC capabilities are accurately incorporated in the TPOCC Capabilities Document

5.1 Configuration Baselines

Two levels of software baselines are established for TPOCC reusable software. The TPOCC Capabilities Document, analogous to a typical system requirements document, contains a structured list of TPOCC capabilities. A build release plan maps the capabilities to specific releases. This permanent baseline will be updated only in response to ICCRs written against TPOCC software or CCRs written against a TPOCC application but implemented as a generic TPOCC capability.

The second configuration baseline documents the as built TPOCC release. This release product baseline consists of the software release, design documentation, the TPOCC implementation guide, generic user's guide sections, test procedures, data, and results.

The TPOCC Detailed Design Specification describes the current design of each generic TPOCC subsystem and includes a summary of each subsystems functions, a subsystem architecture diagram, high level descriptions of each task, unit prologs, a calling hierarchy within each task, network interface specifications, file definitions, and data structure specifications.

The TPOCC Implementation Guide is the programmers manual for using and maintaining TPOCC generic software subsystem. It includes a description of the fundamental UNIX concepts that are used in building a TPOCC application system, a description of the functions available in the TPOCC software library, and a description of how to build and use each generic TPOCC subsystem.

The generic user's guide sections describe operational aspects of the TPOCC reusable software which are incorporated into the system user's guides of each TPOCC mission. Also the TPOCC Display Page User's Guide is provided which describes the methods of creating display pages with each release.

5.2 Software Support Policy

The TPOCC approach used in software support is similar to that of many operating system vendors. The development team makes each release of reusable software available to application development teams for incorporation into their deliverable. If an application group elects not to incorporate the new TPOCC release then software support from the TPOCC development team will not be guaranteed. Any reusable software changes made by the application teams without following the CRB standard procedures for configuration control will nullify TPOCC's support agreement with that application.

New capabilities are scheduled in TPOCC releases based on mission need dates, with an average of two software releases a year. Frequently the TPOCC development team is over booked with new capabilities and something has to give. Based on priority and the required expertise, some ICCRs get

delegated to application development teams. ICCRs that can wait get deferred. An applications implementation of a generic capability is initially delivered as part of its mission specific software and then folded into the TPOCC release at a latter time. Larger subsystems such as the NASCOM interface, packet processing, or Network Control Center (NCC) Interface has been implemented with TPOCC and mission support staff.

New TPOCC releases are tested in a mission independent testbed environment and then made available to mission development teams for integration with mission specific software. A test cycle is completed when an independent acceptance test team tests the complete application system delivery. Problems identified in the application may, after analysis, be found to reside in either the application software or in TPOCC. The CRB reporting mechanism makes generic problems known to all applications.

Changes to generic TPOCC get initiated as an Internal CCR on TPOCC reusable software itself or as a CCR written against the missions requirements document and presented to the Configuration Control Board. Generic TPOCC ICCRs are reviewed by both the generic TPOCC project and the application groups for technical feasibility, cost, and schedule impact. Having the user's who initiate the requirements attend the CRB has been an important part of the success enjoyed thus far.

6. TECHNOLOGY INFUSION

New technology advances are incorporated into the TPOCC architecture by: integrating new/upgraded COTS products, integrating new ground system tools/components developed by NASA organizations external to the MOD, and enhancing TPOCC software. An institutional prototyping and technical evaluation group looks at various products from each area for it's applicability to the TPOCC architecture based on mission requirements and industry trends. As members of the TPOCC Working Group, users and mission development teams provide signifcant input into the prototyping and technical evaluation activities. New systems for trend analysis, s/c subsystem monitoring, and s/c visualizing have been integrated in TPOCC's loosely coupled architecture. The TPOCC User Interface continues to be enhanced by taking advantage of new MOTIF GUI capabilities.

The workstation era has accelerated the obsolesce of computer systems. A characteristic of the mainframe & minicomputer era was to replace systems every five years which effectively reduced any reusable software base to zero due to software dependencies on the hardware. TPOCC software has transitioned between two generations of FEP computers and ports to Sun, HP, IBM, and VAX workstations.

7. CONCLUSION

Averaging 75% of a missions control center deliverable, TPOCC software has reduced cost and shortened the control center development life cycle. Established standards both in industry and spacecraft development have increased the reuse factor to well over 90%. A successful process for evolving the TPOCC software with new capabilities and new technology is in place. As projects "Re-Engineer" the development life cycle by relying more on the TPOCC software maturity schedules will continue to be shortened and cost reduced.

REFERENCES

NASA/GSFC, MOD, CCSB, 511-1PSP/0190, CSC and ISI, TPOCC Project Support Plan (Revision 4), December 1993

--, 511-4SSD/0393, CSC and ISI, TPOCC Implementation Guide for Release 11, July 1994

--, 511-4DDS/0193, CSC and ISI, TPOCC Detailed Design Specification for Release 10, February 1994

--, 511-4SUG/0594, CSC and ISI, TPOCC Display Page Users Guide for Release 11, June 1994

--, 511-4SUG/0494, CSC and ISI, TPOCC Generic User's Guide Sections for Release 11, July 1994

--, 511-4SSD/0290, CSC and ISI, TPOCC Reusable Subsystem Capabilities (Revision 4), April 1994

--, CSC and ISI, "TPOCC: A Satellite Control Center System Kernel That Fosters High Reuse and Lower Cost"

--, CSC, "TPOCC Task Labor Projections Through FY95"

ACKNOWLEDGMENTS

We wish to extend special thanks to Carroll Dudley, Dolly Perkins, Judy Bruner, Dan Mandl, Mike Rackley, Jack Lauderdale, Dave Johnson, and Bill Stratton for their ideas which went into this paper.

N95-17588

111173

354274

p. 5

CUSTOMIZING THE JPL MULTIMISSION GROUND DATA SYSTEM: LESSONS LEARNED

SUSAN C. MURPHY
JOHN J. LOUIE
ANA MARIA GUERRERO
DANIEL HURLEY
DANA FLORA-ADAMS

Operation Engineering Lab
Jet Propulsion Laboratory
California Institute of Technology
MS 301-345
Pasadena, California 91109-8099

ABSTRACT

The Multimission Ground Data System (MGDS) at NASA's Jet Propulsion Laboratory has brought improvements and new technologies to mission operations. It was designed as a generic data system to meet the needs of multiple missions and avoid re-inventing capabilities for each new mission and thus reduce costs. It is based on adaptable tools that can be customized to support different missions and operations scenarios. The MGDS is based on a distributed client/server architecture, with powerful Unix workstations, incorporating standards and open system architectures. The distributed architecture allows remote operations and user science data exchange, while also providing capabilities for centralized ground system monitor and control. The MGDS has proved its capabilities in supporting multiple large-class missions simultaneously, including the Voyager, Galileo, Magellan, Ulysses, and Mars Observer missions.

The Operations Engineering Lab (OEL) at JPL has been leading Customer Adaptation Training (CAT) teams for adapting and customizing MGDS for the various operations and engineering teams. These CAT teams have typically consisted of only a few engineers who are familiar with operations and with the MGDS software and architecture. Our experience has provided a unique opportunity to work directly with the spacecraft and instrument operations teams and understand their requirements and how the MGDS can be adapted and customized to minimize their operations costs. As part of this work, we have developed workstation configurations, automation tools, and integrated user interfaces at minimal cost that have significantly improved productivity. We have also proved that these customized data systems are most successful if they are focused on the people and the tasks they perform and if they are based upon user confidence in the development team resulting from daily interactions.

This paper will describe lessons learned in adapting JPL's MGDS to fly the Voyager, Galileo, and Mars Observer missions. We will explain how powerful, existing ground data systems can be adapted and packaged in a cost effective way for operations of small and large planetary missions. We will also describe how the MGDS was adapted to support operations within the Galileo Spacecraft Testbed. The Galileo testbed provided a unique opportunity to adapt MGDS to support command and control operations for a small autonomous operations team of a handful of engineers flying the Galileo Spacecraft flight system model.

INTRODUCTION

The Multimission Ground Data System (MGDS) at NASA's Jet Propulsion Laboratory has brought improvements and new technologies to mission operations. The development of a generic data system to meet the needs of multiple missions was intended

PRECEDING PAGE BLANK NOT FILMED

PAGE 1170 INTENTIONALLY BLANK

to avoid re-inventing capabilities for each new mission and thus reduce costs. The traditional mainframe-based data systems of the past were expensive to modify and their proprietary architectures did not facilitate incorporation of new technologies. The MGDS is based on a distributed client/server architecture, with powerful UNIX workstations, incorporating standards and open system architectures.

The MGDS system provides a mature, relatively stable set of software for real-time command and control operations and for off-line engineering analysis. The system is based on a table-driven approach with simple user-oriented languages for specifying processing and display functions that allows the addition of new missions without extensive reprogramming. The standard Sun/HP/UNIX end-user workstations are part of a distributed operations system that places a powerful, flexible, and extensible set of operational capabilities at an analyst's fingertips. When properly configured, these workstations greatly increase the efficiency of spacecraft operations.

ADAPTABLE SYSTEMS

The Multimission Operations System Office's (MOSO) understanding of the MGDS design was that multimission capabilities would be delivered to allow the users to customize, adapt, and tailor the system for their individual use. MOSO was responsible for developing, installing, and maintaining the multimission hardware and software for the operations teams, but customizing its multimission software was up to the project. However, the system has become so powerful with over 1.5 million lines of code that its 'configurability' and 'extensibility' can potentially overwhelm users rather than benefit them. The MGDS user guides currently stand over one foot high on end. In addition, the users don't often refer to the user's guides because they don't want to know how to use a tool, they want to know how to accomplish their operations task within the MGDS environment.

The MGDS Workstation Training Group had been frustrated for several years trying to train users on workstations which bore little resemblance to the configuration the users would find in their operations environment. Often, there was no standard project configuration in the end-user environment and users were on their own to transform their blank screens into a mission operations system. Each user worked individually and project-specific files needed for telemetry processing and display were passed in an ad-hoc manner among team members. However, how well a system is tailored for end users is often the most important factor in determining the degree of system operability and efficiency improvements that come from new technologies.

It has become clear that the MGDS system and its documentation cannot simply be delivered to a project for them to adapt for their needs. Adapting the MGDS software has become a complex task with a high learning curve. This makes adaptation an expensive task for individual projects, especially since operators within the same project will have different needs and interfaces with the system. The adaptation of MGDS for a power subsystem engineer may benefit more from knowing how a power subsystem engineer on another project customized the multimission system rather than how an instrument engineer on the same project would do it. Thus, the learning curve can be made cost-effective if it can be re-applied to several projects, with an adaptation team supporting multiple missions simultaneously. As an additional benefit, a multimission adaptation team will bring knowledge and improvement ideas to bear on future development and customization of MGDS for new projects.

OPERATIONS ENGINEERING LAB

Automation and advanced user interfaces can help reduce costs only if they are focused on the people and the tasks they perform. New technologies may only bring minimal cost savings if the new system functions much like the old one. This often happens since the users who write the requirements aren't always familiar with the capabilities of new technologies and simply use their existing

system as a model. For example, the JPL mission controllers asked for a scrolling screen that displayed telemetry values representing the latest value of the spacecraft clock. This was the way the old system allowed them to determine whether there were any data outages. The developers gave them their scrolling display and operators continued to stare at these displays watching for outages. An important opportunity was lost to automate this process and improve the efficiency of operations.

To solve these types of communications problems, the Operations Engineering Lab (OEL) was created four years ago to merge operations and development activities for the Space Flight Operations Section. The OEL builds scheduling, command, control, and analysis software and currently delivers over 500,000 lines of code. The development philosophy is characterized by iterative development with active participation of the end-users. Our approach has been successful because we involve users and trainers throughout development, focus on automating essential, time-consuming operations tasks, and get implementations in the hands of users early. We also have operators work in the OEL and developers work in operations in order to maintain close contact with our users and understand the problems that need to be solved. By working closely with users, we have learned how to use new technology to change the way they do business, not just automate the old way of doing business. For example, we have built a smart alarm tool to automatically perform the data outage task described earlier and improved mission controller efficiency by over 30%.

CUSTOMER ADAPTATION TEAM (CAT)

At the request of the trainers and project teams, the OEL developers began to work closely with mission controllers and spacecraft engineers to adapt and configure the workstation and MGDS software to meet the individual user needs. The project configurations were then transferred to the trainer workstations to allow more meaningful training. This adaptation task, started as a grass-roots effort, has evolved into a more formal Customer Adaptation Team (CAT). A small team of OEL developers and operators have supported the adaptation of MGDS for the Voyager, Mars Observer, and Galileo Spacecraft and Instrument Operations Teams. The OEL CAT provides direct project support in developing workstation configurations, customized processing and display tables, automation and analysis tools, and a common user interface for the project.

The workstation configuration and user interface is designed to provide an integrated system view from which a project team can operate a mission. The approach was to provide the flexibility for both advanced and novice operators to run the system to meet their individual needs without their having to know how to integrate across multiple tools and interfaces. We knew that different operators would use the system in unique ways. For example, 24-hour mission controllers want a system that is oriented to an analyst monitoring real-time data, working interactively at their workstation. On the other hand, the spacecraft engineers seldom need to view real-time data. They typically want hard copy plots and tabular printouts of telemetry parameters available overnight.

When the CAT team first started customizing the ground system for the spacecraft team, it became obvious that the system design forced the user to learn many tools and software interfaces to perform their analysis task. For example, to plot telemetry data, they had to use database query tools to retrieve their telemetry files, process the data through the telemetry processing software, export a processed telemetry parameter file and import it into a graphical plotting tool, set the axis correctly, and print the hard copy plot. The operator needed a single, integrated user interface that minimized operator interaction with the workstation and allowed each subsystem engineer to automate their analysis tasks.

The CAT team built a non-real-time telemetry toolkit and user interface that integrated the existing generic tools in the MGDS. The interface design was based on providing

graphical and command-line interfaces that freed the users from knowing the intricacies of querying, retrieving, accessing, and processing telemetry parameters and eliminated the need to know the intermediate file interfaces across various tools. With one simple command line, a user can ask to plot a telemetry parameter for a given time period without any knowledge of the tools needed to perform that task. Command line interfaces are especially important for users who prefer to have their processing done off-line. Graphical interfaces are provided for users who prefer interactive tools. The spacecraft engineers would set up overnight queries that would produce plots automatically for their review when they arrived each morning. The cost to implement this system was minimal since it was built on top of existing multimission capabilities. The interface was built using a GUI-building tool (OELSHELL) and a powerful scripting language (PERL) developed at JPL. There are no licensing costs and no compilation of C code is required for the graphical or command line interfaces.

A new MGDS subsystem was designed by the OEL to deliver these types of end-user tools and interface shells. It provides tools to fill in the gaps in missing capabilities that are discovered after MGDS is delivered, including project-specific adaptations and unique processing requirements. As a result, it is a subsystem that is continually evolving and has grown to be one of the largest MGDS subsystems.

This effort has been very successful because the CAT team works in the operator's own environment, configuring the workstations on their desks, building scripts to automate their tasks, and designing interfaces to integrate and organize the many software tools. In addition, OEL developers do not have the significant learning curve facing analysts getting familiar with the use of workstations, Unix, and MGDS software. We also provide hot-line and on-site support services for end-users, emphasizing quick response time in order to meet the real-time operations needs. Our multimission experiences mean the lessons learned from one project will be transferred to benefit another. Also, if we find a missing capability in the system, we know who to contact to modify existing software or we will build and deliver a new MGDS capability ourselves.

LESSONS LEARNED

We have learned many lessons in adapting and customizing the MGDS system for end-users. The coordination between the OEL, the mission operations engineers, training personnel, and system administrators greatly improved system operability for the users. The following are some lessons learned in our adaptation activities.

Distributed systems are essential to provide the flexibility needed for incorporating new technologies and capabilities required for missions of the future. Compared to the mainframe-based centralized systems of the past, the distributed nature of modern systems require a more disciplined approach to configuration procedures to ensure consistency among all system nodes. End-users have control of their own workstations and can easily modify processing and display parameters. However, this flexibility can cause traditionally-structured organizations to adopt strong, centralized configuration management tools and procedures to prevent any potential problems. Often this leads to software deliveries that are monolithic, irreversible installations with too much bureaucratic overhead involved in making even small changes. The software delivery process needs to be amenable to simple improvements and fixes in non-critical software. Configuration control of end-user tables, scripts, configuration files, and simple tools for specific project use need to be handled separately from the core system. It must be flexible and be controlled by the operations teams.

The MGDS design recognizes that every user has a unique need and the system should allow for individual customization of tools. However, there was a lack of management understanding of the need to staff a CAT team for the extensive work required to customize a distributed system for users. Initially, there was no official follow-on support after a system was delivered to

operations and hence MGDS operability was rated poor by users. After the CAT team work began, the users' perception of operability was dramatically improved even though the core system was unchanged. We are viewed by projects as the group that makes the MGDS system work for users. There are big payoffs in providing project-specific customization, tools, and interfaces, supplemented with on-site support. Distributed systems require extensive customization to meet the specific needs of users and this should not be left to the device of each individual user or project.

Once a system is customized and automated for the end user, the usage of the system can significantly increase. For example, because we had made the off-line telemetry query and analysis process so easy, a much greater number of operators than originally estimated began to use the system extensively. This created serious network loading and disk storage problems.

Automation must be focused on changing the way we fly spacecraft, not just automating the old way of doing business. The greatest cost reductions can be realized if more attention is paid to the operators and the tasks they perform in order to eliminate tedious, labor-intensive processes and to assist in improving the reliability of critical tasks.

The users also want training geared to their work in the operations environment. The trainers need to know how the users might actually use the system in operations. In addition to providing standardized configurations on training and on project operations workstations, the CAT team developed specialized follow-on training classes focused on the project-specific configuration and use of MGDS capabilities.

CONCLUSION

JPL's Multimission Ground Data System has provided a powerful, adaptable and extensible set of operational capabilities at an analyst's fingertips. With more emphasis on a Multimission Customer Adaptation Team providing integrated systems with customized configurations and interfaces, success has been shown in improving system operability and reducing cost in operations for individual projects.

ACKNOWLEDGMENTS

This work was done at the Jet Propulsion Laboratory, California Institute of Technology, under a contract from the National Aeronautics and Space Administration. We would like to acknowledge the work of the technical staff in the OEL and the JPL Mission Operations Teams for their enthusiasm and support.

111174 N95-17589

354275 P. 8

CONFIGURABLE TECHNOLOGY DEVELOPMENT FOR REUSABLE CONTROL AND MONITOR GROUND SYSTEMS

David R. Uhrlaub
McDonnell Douglas Space and Defense Systems
Dept. F166, PO Box 21233
Kennedy Space Center, Florida 32815
Internet: dru@grumpy.ksc.nasa.gov

ABSTRACT

The control monitor unit (CMU) uses configurable software technology for real-time mission command and control, telemetry processing, simulation, data acquisition, data archiving, and ground operations automation. The base technology is currently planned for the following control and monitor systems: portable Space Station checkout systems; ecological life support system; Space Station logistics carrier system; and the ground system of the Delta Clipper (SX-2) in the Single-Stage Rocket Technology program.

The CMU makes extensive use of commercial technology to increase capability and reduce development and life-cycle costs. The concepts and technology are being developed by McDonnell Douglas Space and Defense Systems for the Real-Time Systems Laboratory at NASA's Kennedy Space Center under the Payload Ground Operations Contract. A second function of the Real-Time Systems Laboratory is development and utilization of advanced software development practices.

INTRODUCTION

The control monitor unit (CMU) automates a wide variety of ground operations at moderate cost utilizing standard software components and appropriate hardware. Users can further automate and customize the CMU through programming languages such as C, UNIX shell scripts, defining measurement triggered logic, defining derived measurements, and creating custom graphical displays. The system can be further customized and extended by using the CMU kernel programming interface (KPI) and C.

CMU technology can operate on UNIX-based notebook computers, desktop computers, single-board VME computers, symmetric multiprocessor systems, and distributed systems using real-time shared memory networks, ethernet, or FDDI-based networks. Fault tolerant configurations are possible with minor software enhancements. Configuring CMU system-level software is similar to configuring operating systems by editing text files. Multiple configurations are defined for a multipurpose system or a single configuration for a special purpose system.

CMU software technology is being developed on Digital Equipment's Alpha AXP computers running OSF/1 that conforms to IEEE POSIX standard and real-time application programming interfaces 1003.1 and 1003.4. The OSF/1 user interface supports the X/Motif standard. Application-specific displays are created with SL-GMS, an X windows-based graphical display editor that provides dynamic real-time graphics driven by measurement values. Mission and test data definitions are stored in an Oracle database that supports real-time additions and modifications of measurement definitions.

All data is archived and may be retrieved and analyzed in real-time using DADiSP, a graphical spreadsheet for scientific data analysis. All operation and user guide information is

PRECEDING PAGE BLANK NOT FILMED

1176

maintained in an integrated on-line documentation system with graphics and hypertext facilities provided by the Interleaf Worldview environment.

Performance testing of CMU shows that configurations supporting 10,000 to several million measurements per second (mps) are practical. A two processor DEC Alpha AXP 2100/500 has benchmarked at 170,000 mps. Data acquisition interfaces planned include MIL-STD-1553B, PCM telemetry, IEEE-488, analog, discrete, and serial I/O.

SYSTEM CONFIGURATIONS

A CMU system may be configured in a variety of ways. Two main types of configurations are off-line and real-time. The off-line configurations are ' standard office-based computers that can define a database, simulate data acquisition, retrieve, display, and analyze data and print it. Simulated data acquisition substitutes for actual hardware data acquisition to provide an environment for developing custom software without utilizing actual end-item hardware. An off-line configuration is a DEC Alpha AXP notebook or desktop workstation, as shown in Figure 1.

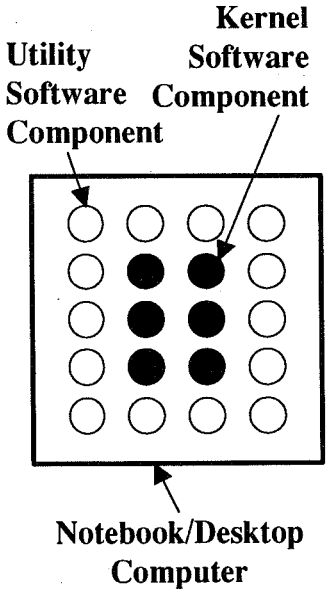


Figure 1. Off-line Configuration

Real-time configurations support data acquisition, end-item commands, and data processing. These configurations may be as small as a single-board computer or portable or desktop computers supporting only a few telemetry or MIL-STD-1553B interfaces. Larger configurations with symmetric multiprocessors and large archive storage devices are configured for handling significant amounts of data for extended periods of time from multiple data acquisition interfaces. Commercial equipment is used to configure fault-tolerant systems. An embedded system using single-board computer technology is another possible configuration, as shown in Figure 2. For large systems where data input and output must be physically distributed, the CMU is configured with ethernet, FDDI, or shared memory networks, as shown in Figure 3. Other configurations currently being developed include portable and mobile weather-proof systems for field use. System configuration is accomplished by modifying one or more ASCII files. Changes to the hardware configuration does not require corresponding software changes.

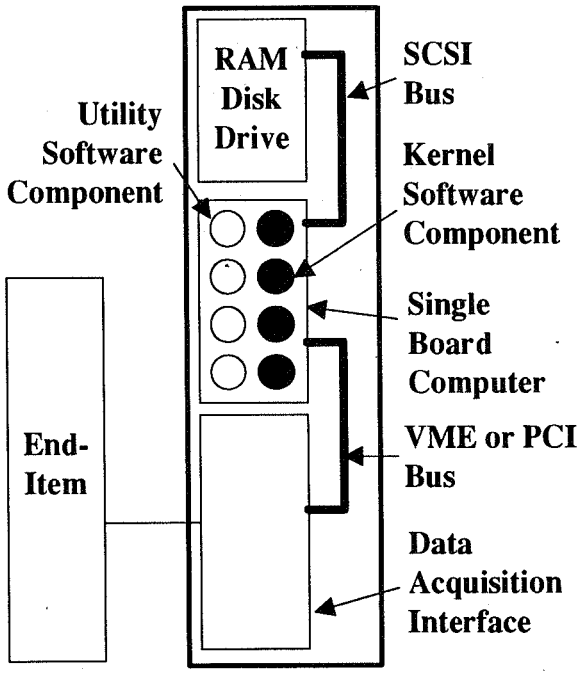


Figure 2. Embedded Configuration

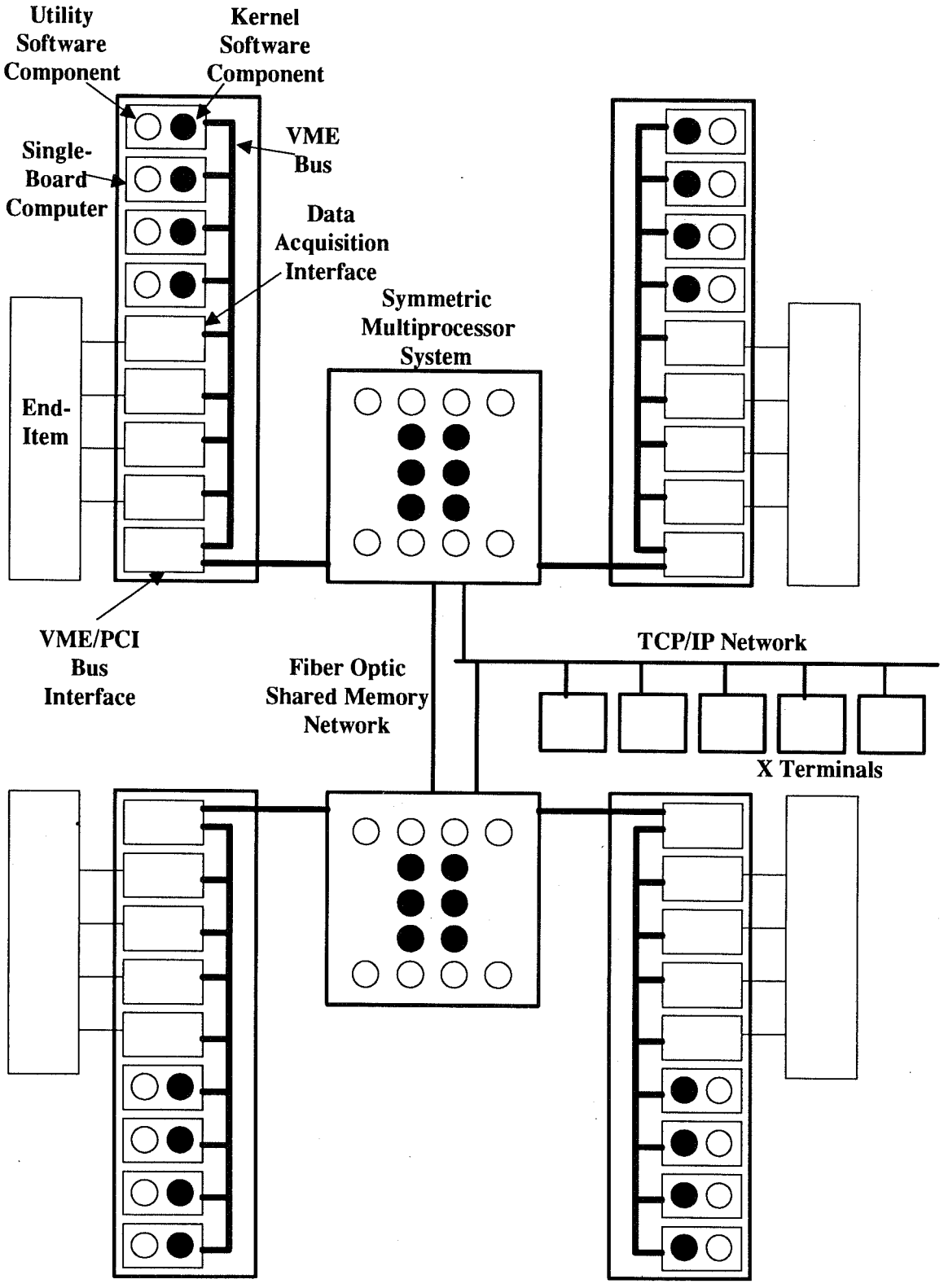


Figure 3. Large Distributed Configuration

SOFTWARE CONFIGURATION

The CMU software architecture has four main elements: the kernel, utilities, kernel programming interface (KPI), and channels as shown in Figure 4. The kernel contains most of the common system-independent functions, while utility software components are more system-specific. The kernel is composed of several UNIX processes. Utilities are generally a single UNIX process. Hardware utilities are unique hardware data acquisition interfaces; user interface utilities provide common and custom graphical displays; and data processing utilities interface to external systems and custom processing functions. The KPI is a high-level interface for developing utilities that communicate with the kernel. All communication between the kernel and utilities

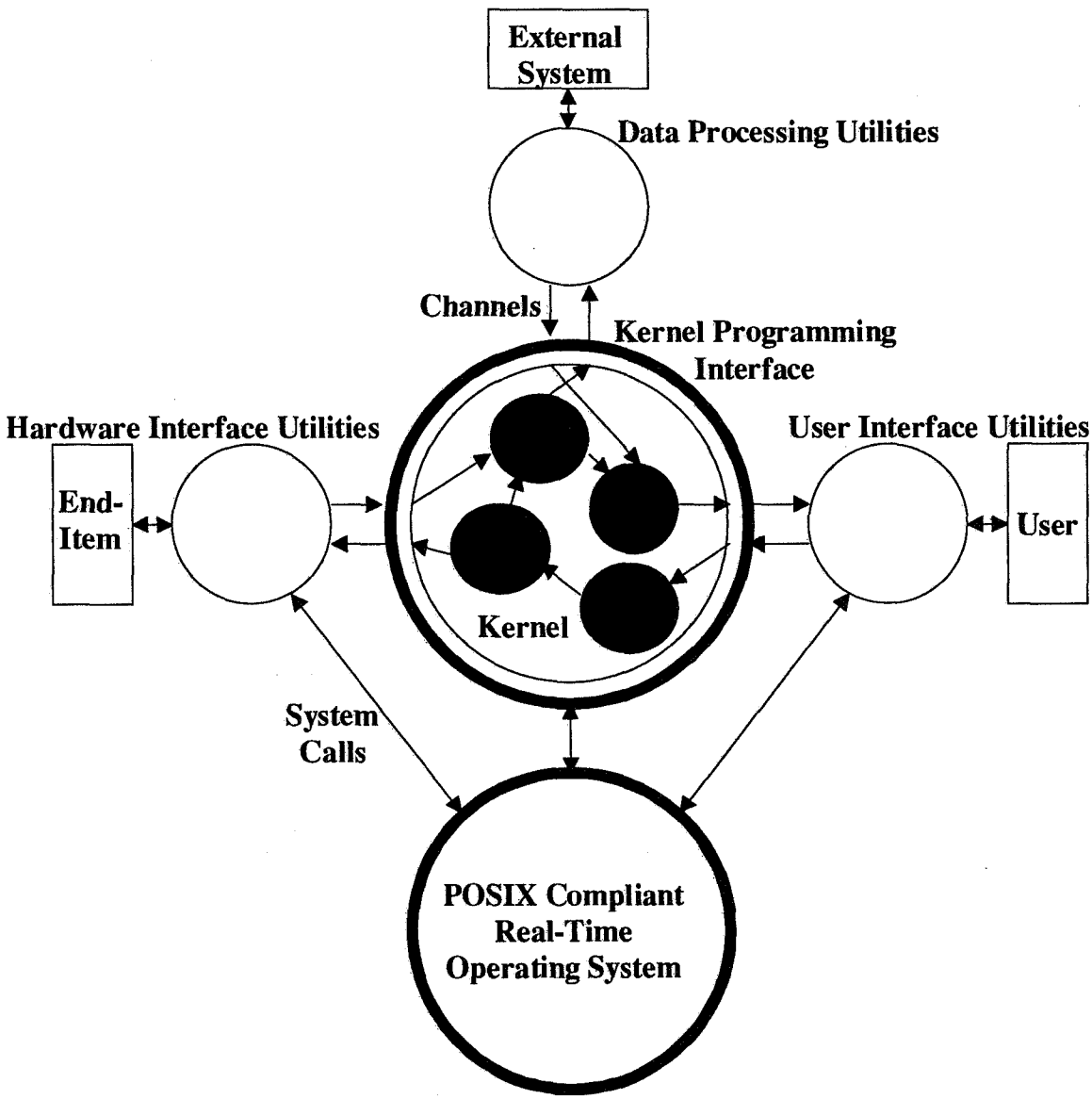


Figure 4. Software Architecture

occur through channels. The four CMU architectural elements combine to provide software functional, architectural, and performance configuration capability.

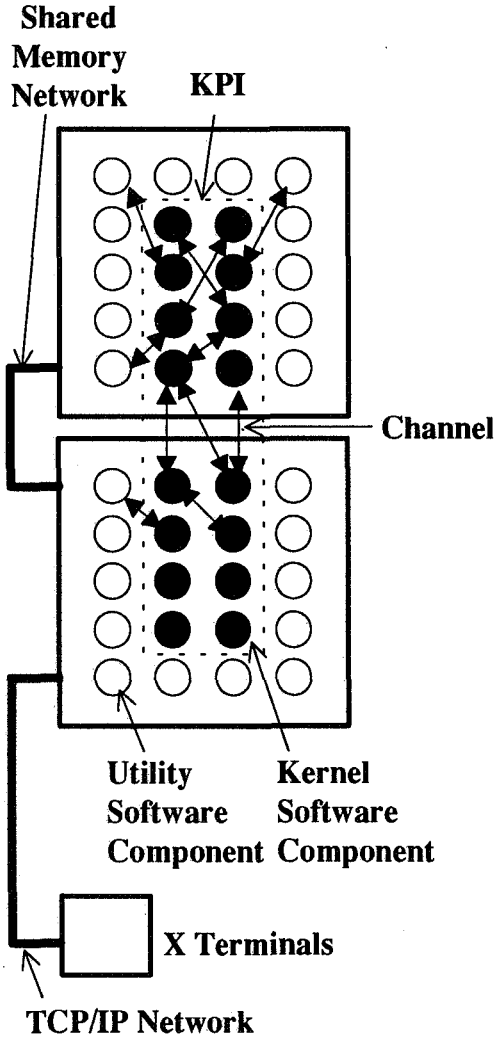


Figure 5. Distributed Kernel

FUNCTIONAL CONFIGURATION

Functional configuration incorporates only the required functionality, conserving system resources and reducing complexity. A system is composed of only the kernel and utility components needed to accomplish a specific task. A system that requires frequent changes in use and measurement definitions is configured to include a commercial relational database system such as Oracle, whereas an embedded system used in static environments where changes in measurements are rare omits the database system and uses a simple ASCII table for storing measurement definitions. An off-line configuration for data analysis such as a notebook computer could omit the commercial database and eliminate the memory and storage requirements. A system used only for monitor and display would not require the CMU archival, retrieval, command, and control components.

ARCHITECTURAL CONFIGURATION

Channels are the primary mechanism for supporting a wide range of hardware and software architectures. Channels provide three key benefits: (1) hardware configurations do not affect the software, (2) a single portable software application programming interface is used for all channel types, and (3) data transfers have very little operating system or application overhead.

Each channel is a connection between two or more software components. A configuration file is used during system startup to configure channels for a specific hardware implementation. Current channel implementations completed or planned are: shared virtual memory, VME bus physical memory, and shared memory networks/reflective memory. Channel support for TCP/IP networks is a simple extension. Kernel components communicate among themselves via channels. Figure 5 illustrates kernel components distributed across two separate computers.

Channels follow the UNIX open/close/read/write/ioctl model. Functions included are: CClose(), CCntl(), CInit(), COpen(), CRead(), CWrapup(), and CWrite(). Benchmarks of a channel configured for shared memory exceeded 2 million mps transfer rates, which would normally consume 120 MB/second, or about 75% of a 160 MB/second system bus bandwidth. Channels have been ported from a 32-bit UNIX System V-based operating system to DEC's 64-bit OSF/1 with minor modifications.

PERFORMANCE CONFIGURATION

A primary design philosophy of CMU software is support for single processor and multiprocessor computers. Multiple computers are combined for higher performance levels. Kernel and utility software components are configured to provide the required performance by taking advantage of additional CPUs. Figure 6 shows actual measured data processing rates for four different CPU configurations and projections for two additional configurations. These configurations use commercial symmetric multiprocessor (SMP) computers. Performance increases of 70% and 100% have been benchmarked for two different CMU software configurations. These increases occur when utilizing a second CPU on a DEC Alpha AXP 2100/500 system. Two 400 MHz 2100/500 computers with four CPUs each connected by shared memory networks alone would support almost 1.4 million mps of data processing. Higher rate systems are configured by using additional SMP and single-board computers linked by shared memory networks and I/O buses.

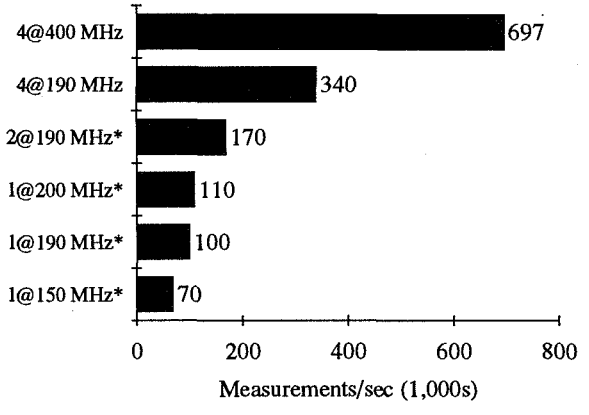


Figure 6. Data Processing Rates
*** Actual Measured Rates**

Archival and retrieval rates are scaled to match system requirements by combinations of magnetic, optical, and tape devices. RAID arrays of storage devices are used to achieve the required storage capacity and data rates. All data are time stamped to the nearest microsecond. Nominal automated control delays are on the order of a millisecond. Specialized configurations using multiple single-board computers in addition to the host computer can support automated control delays in the tens and hundreds of microseconds.

REUSABILITY

The CMU promotes reuse in several ways. Portable software based on POSIX application programming interfaces supports code reuse. The kernel/utility/channel software architecture provides the ability to configure widely varying systems from a single set of software. Reconfigurability allows reuse of CMU software at the component level.

In addition to software reuse, CMU is defining reuse techniques for all products created during development. This would allow custom high-performance real-time systems to be developed quickly and reliably at low cost. Reusability has been extended to modular testing software developed and maintained with the production software for all levels of testing. Methods for reuse of requirements, requirements traceability, and on-line user documentation are also being developed and implemented. Analysis and design models from Cadre Teamwork/Ensemble CASE tools are structured to promote maximum reusability across systems. Hardware design and documentation follow this same reuse philosophy.

SOFTWARE DEVELOPMENT

An evolutionary life-cycle is being used to rapidly improve products and processes in support of McDonnell Douglas' and NASA's TQM initiatives. Process-based methods such as the Software Engineering Institute's (SEI) Capability Maturity Model have been used over the past two years for software process improvement. With development cycles being reduced to four months between requirements and final verification testing, the opportunity for

process analysis and improvement is much greater than that of a conventional two or three-year development cycle of a large real-time system. As with financial investments, the compounding interest effect is already producing significant benefits in cost and quality in CMU software development. Early evaluation by users has already resulted in several improvements over the original requirements. These improvements can be incorporated with little or no impact on cost and schedule since they were identified early in development.

Development techniques incorporated include formal inspections of requirements, system designs, detail designs, code, test plans, test code, and user documentation. Other practices implemented are process and product metrics, nearly 100% path coverage during software unit/component testing, 100% automated testing from unit through system verification, and extensive use of CASE tools. The first two alpha releases of CMU software, A0.1 and A0.3, have had a delivered defect density of 0.06 defects per 1,000 source lines of code (KSLOC). This is compared to industry results [SEI, 1993] in Figure 7.

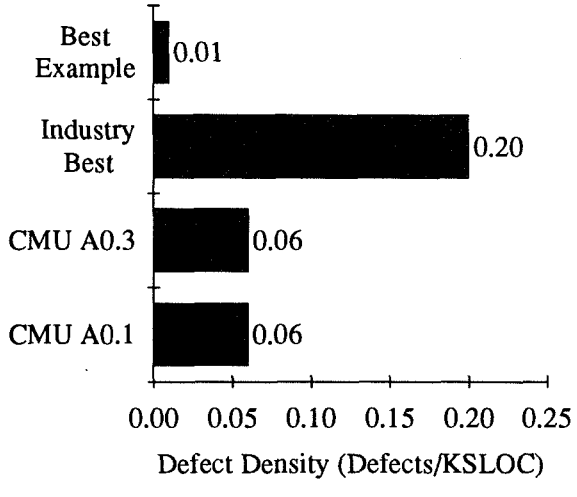


Figure 7. Delivered Defect Density

Only light operational usage via demonstrations and evaluation by the developers and users has been experienced since this is pre-delivery software. The current software is operational two years before its first delivery date, so significant random use and testing naturally occurs during development. This provides further opportunity for defect removal before final delivery to the customer.

Testing was found to remove 3% of all defects detected while inspections removed 97%. On a per defect basis, inspections are 40 times more effective than testing although the testing effort is significantly higher than the inspection effort. Inspections and testing combined for a total defect removal efficiency of 99.87%.

Figure 8 shows the detected and remaining defect densities during each phase of software release A0.3. At the conclusion of code inspections, the remaining defect density was 1.53. The remaining defects were removed during component and integration testing, resulting in a delivered defect density of 0.06. The total potential defect density was 49.9. The detected defect severity during inspection phases averaged 4, or trivial. Average defect severity during testing phases was 3, or minor. Only 2% of all defects detected were of severity 1 or 2, critical or major defects.

Net project software development productivity improved 60% compared to initial releases of earlier projects and is much greater than published for similar projects [Jones, 1991]. Software size is estimated using feature points and bottom-up detailed estimates. Software product quantity average within 10% of the estimates. Software development schedule compliance has improved in the first two releases from 14% to 6% without using overtime. Release A0.3 slipped one week during a six month development schedule.

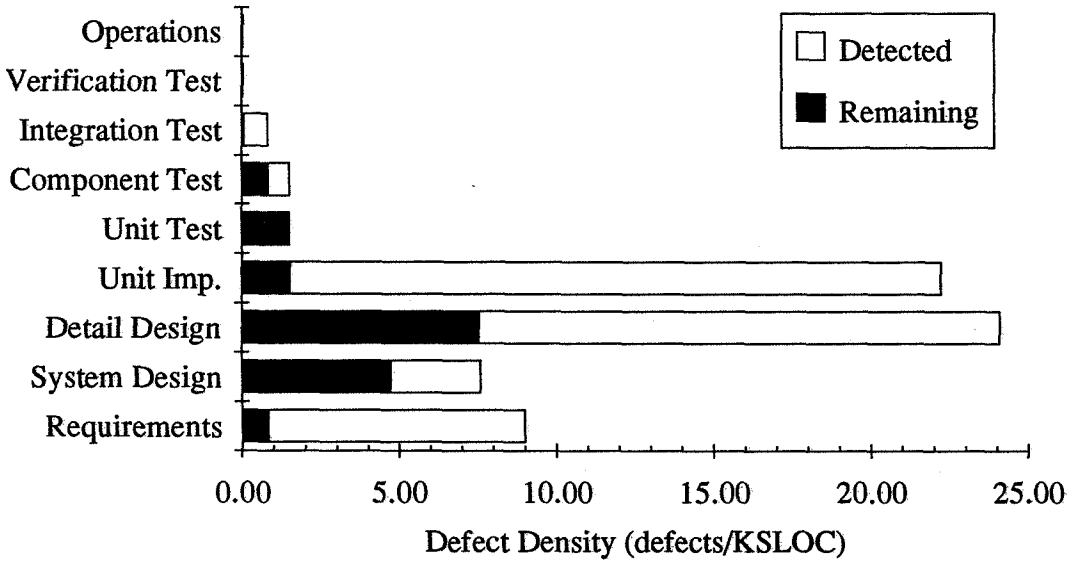


Figure 8. Development Defect Densities

CONCLUSIONS

Configurable software and hardware technology for demanding ground control and monitor systems has been demonstrated. The technology is reusable across small and large systems. Evolutionary development combined with continuous process improvement is an effective approach for developing real-time systems within budget and schedule constraints. Comprehensive inspection and testing contribute to world-class quality levels for software development.

Acknowledgments

The author would like to thank the CMU team members for their invaluable contributions: Jeannine Baker, Rick Bard, Sam Coniglio, Jim Gaines, Ray Ho, Ho Pham, Lawrence Robinson, Bill Snoddy, and Maria Thomas.

REFERENCES

Jones, Capers (1991). Applied Software Measurement. New York, NY: McGraw Hill, Inc.

Gilb,Thomas (1988). Principles of Software Engineering Management. New York, NY: Addison-Wesley.

Software Engineering Institute (SEI) - Carnegie Mellon University (1993). 1993 Software Engineering Symposium Conference Notes. Pittsburgh, PA: Carnegie Mellon University.

3. Standards

* *Presented in Poster Session*

111175 N95-17590

35y282 p.9

A New Communication Protocol Family for a Distributed Spacecraft Control System

Andrea Baldi, ESA/ESOC/FCSD

Marco Pace, Vitrociset Space Division

Abstract

In this paper we describe the concepts behind and architecture of a communication protocol family, which was designed to fulfil the communication requirements of ESOC's new distributed spacecraft control system SCOS II.

A distributed spacecraft control system needs a data delivery subsystem to be used for telemetry (TLM) distribution, telecommand (TLC) dispatch and inter-application communication, characterised by the following properties: *reliability,* so that any operational workstation is guaranteed to receive the data it needs to accomplish its role; *efficiency,* so that the telemetry distribution, even for missions with high telemetry rates, does not cause a degradation of the overall control system performance; *scalability,* so that the network is not the bottleneck both in terms of bandwidth and reconfiguration; *flexibility,* so that it can be efficiently used in many different situations.

The new protocol family which satisfies the above requirements is built on top of widely used communication protocols (UDP and TCP), provides reliable point-to-point and broadcast communication (UDP+) and is implemented in C++.

Reliability is achieved using a retransmission mechanism based on a sequence numbering scheme. Such a scheme allows to have cost-effective performances compared to the traditional protocols, because retransmission is only triggered by applications which explicitly need reliability. This flexibility enables applications with different profiles to take advantage of the available protocols, so that the best rate between speed and reliability can be achieved case by case.

Andrea Baldi (abaldi@esoc.bitnet) works within the Flight Control Systems Department at the European Space Operations Centre (ESOC), Robert Bosch Strasse 5, D-64293 Darmstadt, Germany. Marco Pace (mpace@esoc.bitnet) works for Vitrociset Space Division, Via Salaria 1027, I-00138 Rome, Italy. The work described in this article was carried out at ESOC under a contract with the European Space Agency.

Introduction and Context

SCOS II is a generic mission control system, providing a collection of buildings blocks upon which a custom control system can be implemented with moderate effort (ref. [1]). Basic services are provided by the Distributed Access Service layer (DAS) responsible for distribution, local caching, and retrieval of mission information (e.g. TLM and TLC) over the network. An Application layer (APP) provides basic building blocks for implementing mission applications.

A SCOS II system is distributed and is composed by several Unix workstations connected on a local area network. Each workstation or node has a role with associated communication requirements determined by the mission configuration. The role of a node and consequently its communication requirements are determined by the applications running on it. The following classification is useful to understand the different roles a node might play:

- *server*: a node that provides services, usually data to be consumed by clients.

PRECEDING PAGE BLANK NOT FILMED

PAGE 1186 INTENTIONALLY BLANK

- *replica*: a node that provides services, like a server, but is only available when the primary server is down.
- *client*: a node that makes use of the services provided by servers or replicas, usually data to be consumed.

Servers, replicas and clients can be classified as *Reliable*; in case of a server or replica *Reliable* means that the node *supports* reliable delivery of data; in case of a client it means that it *requires* reliable delivery of data.

A workstation may play more than one role at the same time (e.g. server and client, replica and client), therefore the communication requirements may change over time. Communication and information distribution is achieved using the services provided by the Inter Process Communication layer (IPC) which is part of the DAS.

The IPC services are used by the DAS when communication is required, but also by the APP layer directly as shown in Figure-1.

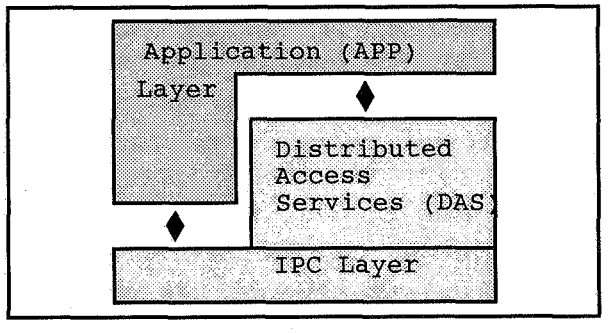


Figure-1 SCOSII Software Layering

The IPC layer has an important role because it supports the bulk of the information exchange among the different system components.

A typical SCOS II configuration (See Figure-2) will be composed by a number of servers, clients and replica nodes. The number of nodes may change dynamically according to the mission phase and configuration, to contingency conditions, and to the number of interactive users connected to the system.

TLM data is received at a central node and distributed to all the nodes by means of the IPC. TLC are dispatched using the IPC to a central node for uplink.

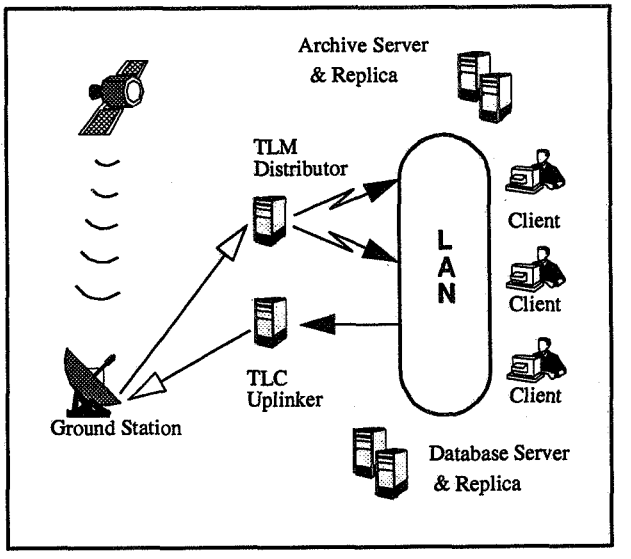


Figure-2 A Typical SCOS II Configuration

In a such context, where applications have different communication requirements, classical protocols like UDP and TCP are not able to cope efficiently with all the possible situations. The IPC tries to fill in the gap existing between UDP, a fast but not reliable protocol, and TCP, a reliable but not efficient protocol, defining the UDP+ protocol.

The protocol family available to SCOS II users extends the IP family and provides:

1. a reliable broadcast service (UDP+), with performance not too far from UDP.
2. an integrated environment where applications with different communication requirements can coexist without imposing overhead to each other.
3. the possibility to select the protocol that best fits the application's communication requirements.
4. compatibility with the already existing IP protocols.
5. support for fast local communication (FIFO).

Requirements

As introduced before, the IPC layer has to cope with many different situations and it is

clear that no unique protocol can be designed to fulfil all the application requirements simultaneously. A protocol family in fact best satisfies multiple and sometimes conflicting needs. The following considerations describe the trade-offs made in order to satisfy as many requirements as possible.

Communication Schemes

Allowing different communication schemes to coexist in an integrated environment is fundamental for achieving flexibility so that applications can use different approaches to data distribution such as *point-to-point* and *broadcast.*

The IPC layer supports all these schemes providing a single unified abstraction called *Channel* available for any supported protocol. A *Channel* can be seen as an endpoint for communication which an application can use to send or receive data.

Protocol Scalability

Scalability is another key requirement and the IPC layer fully scales with respect to the communication data volume by means of the broadcast communication schema. Moreover it tries to avoid situations where the unreliability of the used IP services, which triggers packet retransmission requests, might cause a network congestion.

The retransmission algorithm already tries to optimize the policy of lost packets retransmission using the most appropriate communication scheme; broadcast is used for instance in the case it is detected that a lost packet is requested by several applications. The algorithm is tunable and it is driven by application *Hints* and information piggy-backed into retransmission requests.

Hints are used to instruct a server about the application reliability requirements. They can be used to avoid or force retransmission of data case by case as well as determine the number of attempts the IPC layer carries out before giving up the retransmission.

Protocol Reliability and Speed

The reliability of the protocol together with the speed necessary to cope with a high TLM delivery rate is a primary issue for SCOS II applications. Reliability and speed are tightly related and an effort to meet both the requirements is made.

Within the IP family, TCP is a fully reliable protocol where the speed is inversely proportional to the network load, while UDP is a fast one with a reliability inversely proportional to the network load.

UDP+ stays in between, is highly tunable and tries to fill in the gap existing between TCP and UDP (See Figure-3).

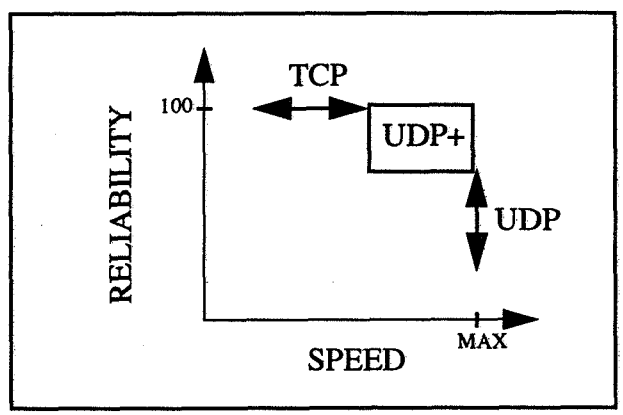


Figure-3 Speed vs Reliability Diagram

Transparent Reliable Data Delivery

Giving the user the responsibility of retransmitting or receiving data lost due to a network problem is not acceptable for client applications. The recovery of lost data is managed automatically by the IPC, without forcing the application to use any recovery policy.

Asynchronous Communication

When applications are data driven an asynchronous communication mechanism is very useful.

The IPC layer provides the concept of a *Notify Channel*: a channel marked as *Notify* does not require the attention of the application; the application only needs to define the handler to be called on data arrivals.

The IPC automatically gives control to the

application handler when data is received on the Channel.

Client - Server Model

The client-server model is a fundamental assumption of the SCOS II system, and the IPC layer supports this paradigm. Servers and clients can synchronize using IPC services according to any convention whose definition is left to the application.

Data Type and Size

It is not possible to foresee in advance the size of data to be carried by the IPC and even the type of data.

The IPC uses a fragmentation algorithm to split a user block of data in many small fragments and to rebuild it at destination upon reception of the complete set of fragments. If some fragments are lost due to a network problems, the IPC is able to rebuild the original block requesting only the missing fragments.

Data Compression and Encryption

The IPC layer provides hooks for data compression and encryption to support external data distribution. Once the application has provided the algorithm, the responsibility to do compression-decompression and encryption-decryption is left to the IPC layer, before any send and receive operation.

Dynamic Routing

The applications operating in a SCOS II system will be *clients*, *servers and replicas*, communicating using the different available schemes. Therefore, a traditional static service location mechanism is not flexible enough to deal with dynamic relocation of services. SCOS II uses a network routing system which allows such a management scheme, and the IPC layer defines virtual services to access such facilities when required.

Applications have the freedom to resolve logical service names using the routing system or to use directly the physical IP address when they initialise a Channel.

Extensibility and Portability

The IPC layer is designed to be easily extensible should the need for the implementation of other protocols arise in the future. The extensibility is made easy by its Object Oriented approach, which allows the specialisation of any of its classes. This is not limited to IP-based classes, but any other protocol can be easily integrated into the IPC hierarchy.

Portability issues are addressed basing the IPC software implementation on consolidated standards like TCP/UDP on the protocol side and POSIX, Unix System V Release 4 for the system interface.

Conceptual Protocol Layering

The conceptual protocol layering is one of the first issues addressed during the analysis phase. The Unix environment offers the IP family (ref. [4] and ref. [5]) as a baseline upon which broadcast protocols can be implemented. In this scenario two alternatives are possible: either to implement the *reliable broadcast* directly on top of IP or to use both UDP and IP, redefining some of their services. This latter is considered a good compromise between implementation cost and duration. Figure-4 shows the conceptual protocol layering, and the role of the IPC in giving the application writers an homogeneous interface.

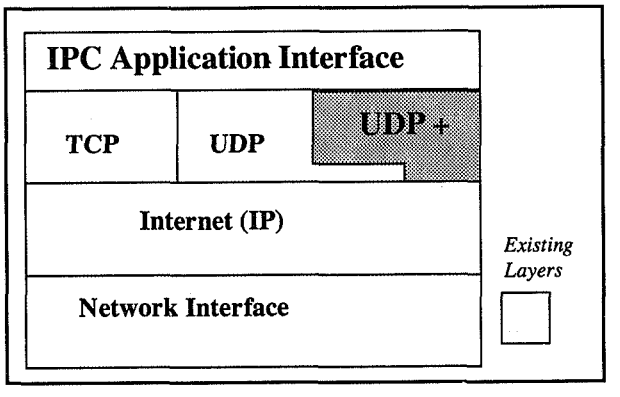


Figure-4 Conceptual Protocol Layering

Retransmission versus Acknowledge

Many papers address the problem of defin-

ing and implementing a reliable broadcast protocol, and the possible solutions are based either on acknowledge techniques or retransmission mechanisms (ref. [3]).

Acknowledge based protocols require an extra packet to be sent over the network to acknowledge the reception of any packet (or range of packets). As the acknowledge is received by the originator the packet is considered delivered. TCP/IP uses these techniques to achieve the full reliability of the protocol.

Protocols based on retransmission (ref. [2]) assign a unique sequential number to any transmitted packet so that it can be requested again when a gap is detected by the client side of the protocol.

When broadcast needs to be supported an acknowledge based protocol has high overhead, because the network traffic due to the acknowledge packets grows proportionally with the number of clients.

Moreover, clients need to register themselves with the broadcast protocol to make the acknowledge algorithm aware of the number of messages it should get back, before assuming that a packet is successfully delivered.

Both the approaches have advantages and disadvantages:

- *speed:* a retransmission based protocol is on average faster, because it does not require any acknowledge overhead.
- *reliability*: a retransmission based protocol might not be fully reliable depending on the algorithm used for the generation of the sequence number.
- *scalability*: in an environment where the number of clients changes dynamically and cannot be foreseen completely in advance performance must be stable over the time. Scalability with an acknowledge based protocol is more penalised.
- *support for broadcast*: it will be the basic interface for applications which don't know in advance how many clients are on the network at a certain time. Broadcast can in practise be supported by both protocol types, (ref. [3]) but in an acknowledge based protocol the design and implementation are more complex. The reason is that a sender should know at any time how many clients are currently receiving its packets, and handle the packet acknowledge consistently.

Since it was required to provide a tunable range of values, both for speed and reliability, we selected a retransmission based mechanism, which is much more flexible, allowing fine tuning of both the parameters. Had we selected an acknowledge based mechanism, we would have had a fixed value for the reliability with the speed bound to the network load.

Moreover the retransmission approach offers a solution which fully supports the broadcast, making the implementation simple enough to be layered on top of existing protocols with a consequent saving in development costs and duration.

Protocol Behaviour

The description of the protocol operations which follows assumes for simplicity the case of one server and one client, but it may be generalised to the case of multiple servers and clients. The server will be responsible for sending data, the client will be responsible for receiving data.

The server keeps sent packets in an internal *history buffer* so that it can satisfy a retransmission request from its client. It assigns sequence numbers so that a client can identify any problems due to the delivery.

The client part of the protocol uses an internal *client buffer* to store packets which cannot be yet delivered to the application layer, when a delivery problem is detected.

Under normal operation the server broadcasts a packet and the protocol stores the packet in the history buffer. When the server receives a retransmission request it accesses its history buffer, looks for the requested packets and transmits them again. Once a while the history buffer is purged to remove packets which will not be required any more, taking care to avoid the potential risk of removing still requestable packets.

To keep this risk at a minimum a *piggy-backed* acknowledgement of the already received packets is used in the retransmission requests. It is important to note that for broadcast communication the *piggy-backed* information provides just an indication and there is no guarantee that a packet will not be requested in the future.

Under normal operation, when a packet is received the client protocol verifies that its sequence number is correct and then it delivers it to the application. The following anomalous situations are recognised and handled by the client side of the protocol:

- *loss of packets*: when a gap in the sequence is detected the retransmission of the missing packets is requested. In the meantime out of order packets can be received; they are discarded or stored in the client buffer according to their sequence number.
- *duplication of packets*: duplicated packet are always discarded.
- *delay in packet delivery*: if a delayed packet is received before the retransmitted one, it is returned immediately to the application, otherwise it is discarded.

Protocol Family Design

This section provides the description of the architecture of the IPC layer. The architecture described is a simplified one showing the main classes relevant to the problem domain. Some implementation classes have been omitted for the sake of simplicity and clarity.

The description uses a Rumbaugh Object Diagram (ref. [6]) where classes have been grouped into 3 *subjects (Data Handling, Transport Mechanism* and *Statistics*) according to the responsibilities they fulfil in the problem domain, as shown in Figure-5.

Data Handling Subject

Data handling groups together the classes dealing with the SCOS II transfer unit (STU). They implement respectively the *header* and the *data part* of an STU, the fragmentation and reassembly of STUs and the storage of STUs for the client and server side of the reliable protocol.

The *StuHeader* class specifies the information needed by the IPC layer to perform the transport of the packets on the network, the sequence number used by the UDP+ protocol,

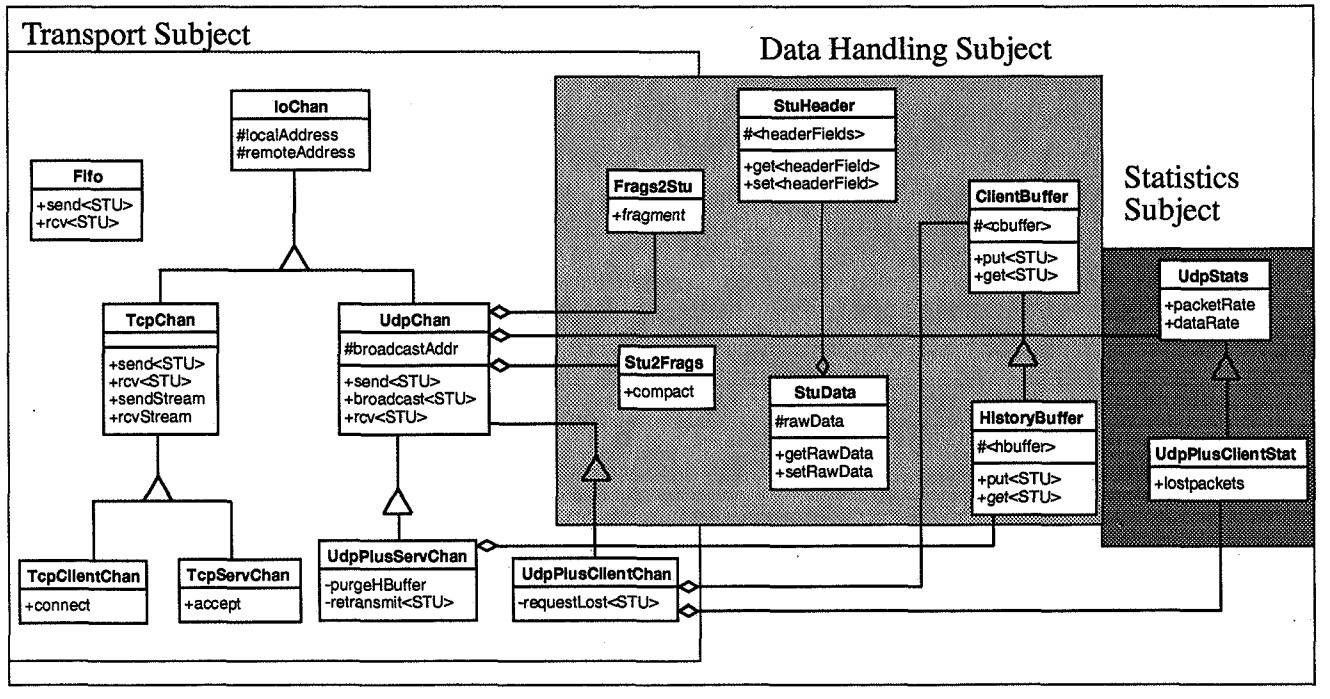


Figure-5 IPC Object Oriented Model

the fragmentation information and client hints. Services for getting and setting the value of the header fields are used by the IPC internally, but can also be used by applications to manipulate fields containing application-defined values. Those fields allow the STU to be tagged as belonging to a specified category.

The *StuData* class contains the application data, of whatever type. Such data are appended to the header and sent over the network when a send operation is performed.

The fragmentation layer (*Frags2Stu* and *Stu2Frags*) takes care of the splitting of large packets into smaller ones and of their reassembly at the destination. In this way the IPC layer is able to support the transmission of packets of virtually any size.

The *HistoryBuffer* and *ClientBuffer* classes implement data structures used by the UDP+ transport classes for the storage and retrieval of STUs. In particular, *ClientBuffer* is used to temporary store packets which cannot be delivered to the application because the retransmission of a lost packet is in progress; the *HistoryBuffer* is used to store transmitted packets which might be requested when a packet is lost.

Transport Mechanism Subject

Transport mechanism groups together the classes which deal with data delivery for the IPC layer protocol family. With the exception of the *Fifo* class introduced for fast local communication all the classes support remote communication.

The *Channel* class defines attributes and services which are common to all the supported communication schemes in the IP domain. *Channel* specializes into two branches respectively responsible for a TCP-based and an UDP-based communication. New IP based protocols can be derived from the *Channel* class specialising it at the most suitable level of the hierarchy.

Inheriting from the *TcpChan* class, two specific classes are defined to model the client and server side of a connection based communication. Both of them provide services for sending and receiving STUs and byte streams: the *TcpClient* adds connection establishment capability and the *TcpServer* adds connection acceptance capability.

Also inheriting from *Channel*, the class *UdpChan* provides services for receiving and sending STUs, both with point-to-point and broadcast connectionless transmission.

Classes *UdpPlusServChan* and *UdpPlusClientChan*, reliable components of the IPC layer, are derived from the *UdpChan* class. In principle, they could have been grouped together in a single *UdpPlusChan* class because they provide the same interface as *UdpChan*. The reasons for such separation are:

- typically applications behave either as clients or as servers, not as both.
- the resulting implementation is less complex and easier to maintain.
- the resulting application overhead is reduced because applications will only include the minimal amount of data structures instantiated by the relevant classes, being such data structures different in the two cases.

Class *Fifo*, at last, supports fast local communication of STUs through a UNIX FIFO, maintaining a similar interface to the one provided by the remote communication classes.

Statistics Subject

Statistics groups the classes responsible for gathering information on volume of data sent and received on any UDP based channel. They have been introduced to tune and debug the internals of the IPC layer, since they provide a complete snapshot of the behaviour of the protocol including all the information on lost and retransmitted STUs.

Moreover, the Statistics classes have been used to implement a performance monitoring tool which reports on data and packet rate. Statistics are also available to an application for any possible usage it envisages.

Building Applications

The IPC layer offers application writers a flexible solution for the exchange of data. Applications can in fact exchange STUs using the protocol that best fits their communication requirements. Moreover applications using one of the UDP based protocols can select the reliability level as they like. It is important to notice that applications can be configured to use any of the available UDP based protocols, and still be able to communicate with each other.

Data transmitted by an application using a reliable server channel (*UdpPlusServChan*) can be received by an application using a non reliable channel (*UdpChan*) with the only difference that lost packets will not be detected and consequently not requested.

Reliable clients (*UdpPlusClientChan*) can also receive data sent over a non reliable channel, but in this case the protocol does not perform any check on the sequence number, and just delivers the data it receives to the application.

The following list shows some of the SCOS II applications or system components together with their communication requirements (see also Figure-2):

- *TLM Receiver and Broadcaster:* it receives the telemetry from the ground station and broadcasts it to the system. It is a reliable server and satisfies retransmission requests coming from reliable clients.
- *History File Archiver (HFA)*: it archives the received telemetry and retrieves it on application demand. As a consequence that all the telemetry coming from the ground station need to be archived, the HFA is a reliable client. At the same time, it satisfies retrieval requests on the network, so it is a reliable server.
- *TLM Cache*: it receives real time telemetry and makes it locally available to the applications. It can be configured either as a reliable or a non reliable client according to the role the node has in the system.

Conclusion

The use of the IPC layer for more than one year in the SCOSII system has shown that the initial objectives have been achieved:

- the retransmission approach together with the almost full reliability of the network hardware make the degree of reliability high enough to guarantee that any node receives the data it needs to accomplish its role.
- UDP+ efficiency compares favourably with UDP and definitely well with TCP. The overhead introduced by the retransmission mechanism is a fraction of the benefits obtained, especially when considering reliable broadcast. The results collected using the IPC statistics are summarized in Figure-6 where the data rates achieved are shown. Such figures may vary, however, depending on the dynamic tuning the applications perform on the IPC using hints.

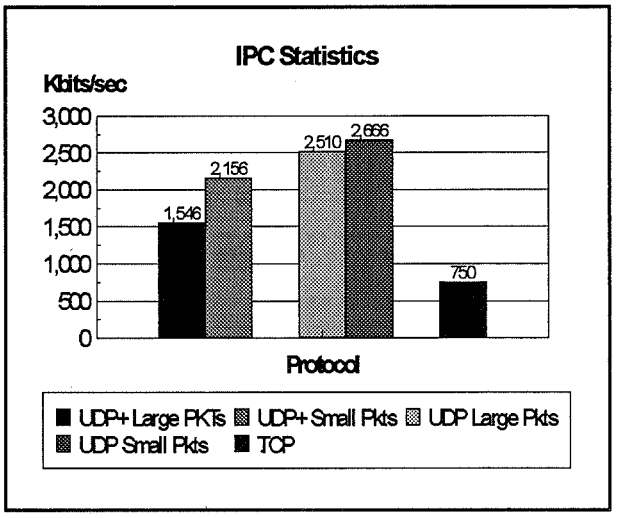


Figure-6 Protocols Statistics for a Typical Mission Configuration Consisting of 20 Nodes.

- the protocol scalability guarantees that adding new client workstations does not require any reconfiguration and does not impose unacceptable network overhead.
- the protocol family has shown to be flexible enough to satisfy different communication requirements for a wide range of applications that need to exchange the same data

using different protocols. This is achieved by the IPC layer through the introduction of a common exchange data unit (the STU) together with a continuous range of performances both for point-to-point and broadcast communication. It is important to note that SCOS II applications can communicate even with already existing software not supporting a STU based data exchange.

The IPC layer is now complete and stable in the interfaces, although its implementation evolves as a result of a continuous life cycle which includes analysis of statistics, tuning and test.

New generations of Unix already support multiprocessor hardware and the time to make the IPC layer fully reentrant is mature as multithreaded SCOS II applications are under development. To have a full coverage of commonly used protocols the IPC layer will be augmented in the future to support non IP based protocols, like Unix datagram, streams and X25.

References

[1] K.Keyte: SCOS II - *A distributed Architecture for Ground System Control* - International Symposium on Spacecraft Ground Control and Flight Dynamics SCD1 - Feb. 94 - Sao Jose dos Campos - Brazil.

[2] M.Frans Kaashoek et al. - *An Efficient Reliable Broadcast Protocol* - Operating System Review-Volune 23 Num. 4, October 1989.

[3] T.A. Joseph and K.B. Birman - *Reliable broadcast protocols* - Lecture Notes of Artic 88, Tromso, Norway, July 1988.

[4] Douglas E.Comer - *Internetworking with TCP/IP* - Vol.I - Prentice Hall.

[5] W.Richard Stevens - *UNIX Network Programming*- Prentice Hall - 1990.

[6] Rumbaugh et al. - *Object Oriented Modelling and Design* - Prentice Hall - 1991.

111176 N95-17591

354286
p. 8

STANDARDIZING THE INFORMATION ARCHITECTURE FOR SPACECRAFT OPERATIONS

C. R. Easton
McDonnell Douglas Aerospace
Space Station Division, MS 17-4
5301 Bolsa Avenue
Huntington Beach, CA 92647

ABSTRACT

This paper presents an information architecture developed for the Space Station Freedom as a model from which to derive an information architecture standard for advanced spacecraft. The information architecture provides a way of making information available across a program, and among programs, assuming that the information will be in a variety of local formats, structures and representations. It provides a format that can be expanded to define all of the physical and logical elements that make up a program, add definitions as required, and import definitions from prior programs to a new program. It allows a spacecraft and its control center to work in different representations and formats, with the potential for supporting existing spacecraft from new control centers. It supports a common view of data and control of all spacecraft, regardless of their own internal view of their data and control characteristics, and of their communications standards, protocols and formats. This information architecture is central to standardizing spacecraft operations, in that it provides a basis for information transfer and translation, such that diverse spacecraft can be monitored and controlled in a common way.

ACKNOWLEDGMENT

The Space Station Freedom Program (SSFP) funded the development of the information system standard, "Data and Object Standards" (DAOS), SSP 30552, which forms the basis of the standard recommended for spacecraft control. The SSFP adopted only the naming portion of the standard. The remainder was under consideration when SSFP transitioned to the International Space Station Alpha (ISSA) Program.

The concepts presented in this paper are not part of ISSA. Funding constraints and the newness of the technology led to a more conventional approach for initial operations. The concepts in this paper may be appropriate for inclusion in a program growth phase to simplify information management and reduce operating costs.

Several people contributed to the development of the SSFP DAOS standard. The author would like to especially acknowledge Virgil Enos (formerly with McDonnell Douglas Aerospace in Houston, Texas) and Lee Neitzel (with CTA in Houston, Texas) for their leadership in developing and prototyping the SSFP standard which forms the basis for this recommended spacecraft standard.

The concepts from DAOS have been incorporated into the Instrument Society of America, Fieldbus Application Layer Specification as a draft international standard.

INTRODUCTION

The operation and control of spacecraft requires data exchange between the spacecraft and its control center. Ground control centers must also communicate with payload specialists and technical support teams in off-site locations. With an ever increasing number of spacecraft, and with limited resources, multi-program control centers are likely to become common. It will be essential to standardize many aspects of the spacecraft and their control centers. An integrated information architecture standard will support this goal by providing a common view and understanding of data regardless of source.

The IEEE Standard Dictionary of Electrical and Electronic Terms defines data as, "any representation such as characters or analog quantities to which meaning may be assigned." An information architecture provides a formal mechanism for assigning meaning, as well as defining data in its various representations.

The purpose of an information architecture is to provide a standardized way of identifying, formatting, transporting and storing program data. Ideally, an information architecture would

PRECEDING PAGE BLANK NOT FILMED PAGE 1196 INTENTIONALLY BLANK

be in place at the beginning of a program, and would encompass all program data in a compatible format. But the reality of spacecraft programs is that such an approach does not happen. Designers develop data in various formats in many different repositories. A program approved information architecture is not available at the start if the program. Even if it were, it might not be compatible with all of the Computer Aided Design (CAD) and Computer Aided Software Engineering (CASE) tools to be used. In addition, all of the program team would have to be trained in the use of the architecture from the beginning of the program. Finally, there is a need for new, multi-program control centers to be backward compatible with existing spacecraft designs and data.

From the above, it becomes clear that an information architecture must take into account the practical realities of spacecraft programs. It must permit local representations of data in local, user defined data bases and spreadsheets. It must support data exchange among local, user defined representations. It must support data collection, integration, and validation throughout the design, development, test and evaluation (DDT&E) phases of the program. It must support the promotion of DDT&E data to the operations phase of the program, its integration with operations data and the eventual post-mission evaluation of the program. It must support the operation of multiple, dissimilar spacecraft from the same control center, including the retrofit for operation of existing spacecraft missions.

DESIRED FEATURES

A standardized information architecture would provide for a basic set features which standardize information formats, access to and exchange of data. Among these features are the following:

A common way of naming data - Data names control the access to, transport of and utilization of the data. No single naming standard will suffice for all of these purposes. The standard must define a common way of naming data which allows for various short form names for various purposes. Ideally, all of the short form names would be related to the standard by definitive rules and program specific data. All such names need to be defined in a program dictionary. The common way of naming data provides the user with a way of locating the data in the dictionary.

A common way of accessing data - Data will tend to reside in user defined repositories. Access may be locally controlled, and access procedures are also user defined. To make data readily available, the information architecture needs to include a directory indicating where data are located and how to access the data. Ideally, the architecture should support access in the requester's local representation. The dictionary/directory would then also support conversion of local syntax, format and storage between local representations.

A common way of transporting data between different local representations - Data accessible over a network may be imported into other data bases on demand or automatically imported via linkages. When the local representations differ, the import routines must be customized to make the necessary conversions. With many different local representations, conversion can become unmanageable. The information architecture should define a common transport representation which allows each local representation to map to and from a single, common representation for data exchange.

A common way of viewing information - The architecture must support user oriented views of information. These views are normally organized around the spacecraft design, subsystem function and mission operations. The same data may be important to all such views in differing contexts.

A common way of understanding information - Human users and computers need to understand the data. The understanding usually comes from defined relationships. A human operator sees a number displayed next to an icon labeled, "Pump 1 Inlet Temperature", and understands the data. Behind the display, the computer "understands" that a particular data item is the inlet temperature attribute of Pump 1. Thus, humans and computers have different needs for data and ways of understanding data. Both must be supported by the information architecture.

A common way of relating information - The same data may be used in a variety of contexts. For example, system architects are interested in device connectivity to assure failure tolerance. Hardware designers are interested in the same information for wiring harness design. Software developers need connectivity to relate I/O ports to commands and data. Test personnel need to verify connectivity and I/O function. The

information architecture must support these various relations of information to context.

TOP LEVEL REQUIREMENTS

The information architecture must meet a set of top level requirements in order to be able to support a wide variety of spacecraft applications. Some of these requirements are stated below as mission goals.

Be robust enough to describe complex spacecraft and spacecraft constellations - There is a trend toward designing simple spacecraft for limited missions and using multiple spacecraft for more complex missions. While this trend may make it seem that the information architecture need only deal with simple spacecraft, the architecture should not preclude more complex spacecraft which may be developed in the future.

Support a common view of all spacecraft and payloads - The goal is to provide an operator control interface such that all spacecraft can be viewed in the same way. Note that this does not mean that all spacecraft views are identical. Rather, it means that the logical approach to accessing and working with spacecraft capabilities and functions is the same for all spacecraft. Spacecraft will not be designed with the all of the same capabilities and functions. Those which are the same may be implemented differently. As a result, the information architecture will take on a portion of the responsibility for providing the common view of the spacecraft and its operation.

Be transaction oriented to support remote operation and access - Spacecraft control is not limited to working with data local to the control center. It involves message exchange with the spacecraft, with payload specialists at various locations, and consultation with spacecraft designers and other specialists. It may involve access to remote data bases. The transaction orientation separates the action of the operator (or software) to access data from the process of accessing the data.

Provide global definitions of information and relationships - There are two aspects to this requirement. First, a program will often develop differing definitions of data to serve the design, test and operations phases of the program. Second, data and definitions will usually vary from one program to another. The global definitions serve to integrate data throughout the phases of a program and to make common data definitions available to new programs. Thus, once "Control Moment Gyro", "Greenwich Mean Time" and "CCSDS Packet" have been defined, those definitions can integrate data across a program. The definitions are portable from one program to another.

Support a variety of local representations and formats - People develop local representations to meet local needs. Some of the data in local repositories need to be made available to outside access. Most of the data can be readily converted and transferred. But then the system has multiple copies of the same data, with the attendant configuration management problems. The information architecture needs to support the exchange of data among local representations. This will not solve the configuration management problems, but will make them more tractable.

Provide for definitions to be transferred with data - Many information exchanges will be made with the definitions of the information already known. As systems become more open, there will be an increasing need to transfer the definitions of the data with the data. This will be especially important in the sharing of payload and spacecraft data with outside investigators. One major limitation on the ability of investigators to access such data is that the definitions are not available and may be permanently lost. The information architecture should support standardized definitions and the ability to store and transfer the definitions with the data.

Be well grounded in proven standards - Grounding in existing standards is desirable for two reasons. First, it is far more efficient to use or modify an existing standard than it is to develop a new standard. Second, the development and consensus building that have gone into forming an existing standard will make it easier to form a consensus on an extension to a new application of the standard.

Have the potential to support growth and technology upgrade - There are three reasons for supporting growth and technology upgrade. First, individual programs experience growth and upgrade. Upgrade may come from on orbit refurbishment, or from new generations of the same satellite. Second, a control center may be tasked to host controls for an entirely new spacecraft or constellation. Third, both the types of satellite technologies used and the technologies

for hosting the information, itself, will change over time. Since the information architecture is to grow from one program to another, it must support the technology upgrade and growth.

OVERVIEW OF THE STANDARD

The Space Station Freedom Program developed an information architecture standard having the features and meeting the requirements noted above. Subsequently, the Instrument Society of America incorporated this standard into its Field Bus standard. It is being considered as a draft international standard for field buses. The same standard is also under review by the Spacecraft Control Working Group of the AIAA Space-Based Observing Systems Committee on Standards as its information architecture standard.

SSFP used the terminology, "Data and Object Standards" (DAOS) to describe the information standard. It includes standards for an integrated Data Dictionary/Directory, Object Definition, Data Modeling, and Message/Data Structure Definition. This paper will continue to use the term "DAOS" to refer to the several individual standards which make up the information architecture standard.

Dictionary/Directory Standard

DAOS uses the term "Encyclopedia" to mean an integrated Dictionary and Directory. The encyclopedia provides for a single source for definitions, contextual references and access information.

The dictionary part of the encyclopedia is based on the Information Resource Dictionary System (IRDS), (ANSI X3.138 and FIPS PUB 156). It defines classes of objects (or families of spacecraft devices) as determined by the common characteristics of real devices.

The directory portion of the encyclopedia is based on the ISO/ IEC Directory Standard (ISO 9594). ISO/IEC provides rules for naming objects and protocols for querying remote directories and receiving replies. In the DAOS encyclopedia standard, the directory provides information used to locate object instances. This includes object names, descriptions, and object specific attributes (such as location).

The encyclopedia is fully integrated, in that all information is organized about objects or devices, their classes and their relationships. This way of organizing information intermixes the dictionary and directory within object descriptions.

The encyclopedia is comprised of four layers, as shown in Figure 1 and recommended by IRDS. The top layer defines the schema for the second layer, and is not to be altered by the users.

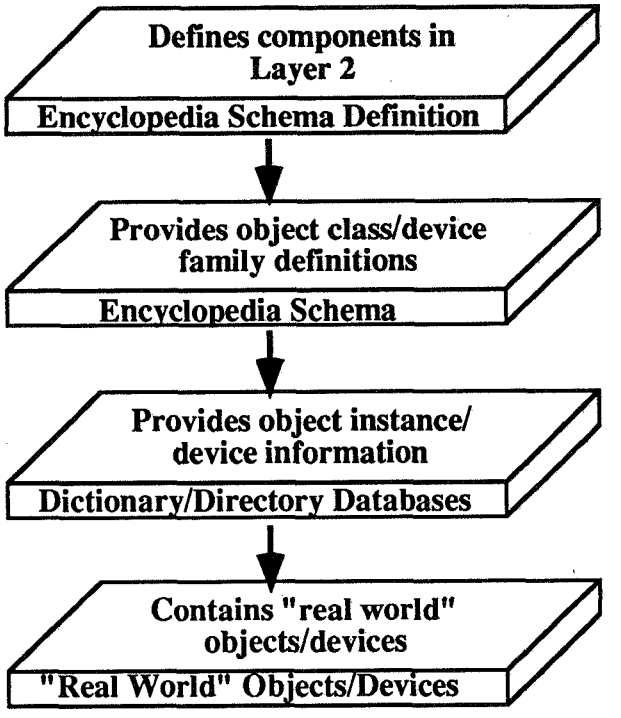


Figure 1 Encyclopedia layering

This top layer specifies such how objects are to be defined, how relationships are to be specified, what data modeling is assumed, etc. Thus, the top layer defines all of the tools to be used for defining object classes/device families.

The top layer can be expanded as needs for new definitions arise. Existing definitions will not normally be modified. If modifications are required, they will normally be included by creating new definitions. The definitions are not to be modified or extended by the users.

The second layer provides the object class/device family definitions. While there are differences between "object class" and "device family", there are no differences which are important to this paper. An encyclopedia may be developed using either or both.

The directory part of Layer 2 contains rules for naming data, syntax rules for storing and transporting data, and attribute or data types for common definitions of data.

Object classes will be discussed below. An object instance or device exhibits the characteristics specified for the object class or device family to which it belongs. The information about the object instances or devices is carried in layer three of the encyclopedia.

Layer four holds the actual objects or devices. As such, it is not directly a part of the encyclopedia, but is a part of the model for the encyclopedia. Layer 3 of the encyclopedia does contain the description of these real world devices.

From the above, it can be seen that each layer of the encyclopedia contains the information necessary to understand the successive layer.

Object/Device Model

The standard is Object Oriented, in that all information is categorized as either exchanged between objects or describing an object. Objects exchange actions, responses and other message types. They are defined by attributes, functions, events and behavior. The object class structure standardizes information definitions.

An object is anything that is accessible and of interest to a user. It may be a representation of a physical device, a software function, a message, or other. The generic model allows an object to be tailored to include just those portions necessary to describe it.

The object model for DAOS is shown in Figure 2. Beginning with the upper left hand side, an object communicates with other objects via messages. (Note that devices are not necessarily constrained to communicate by messages.)

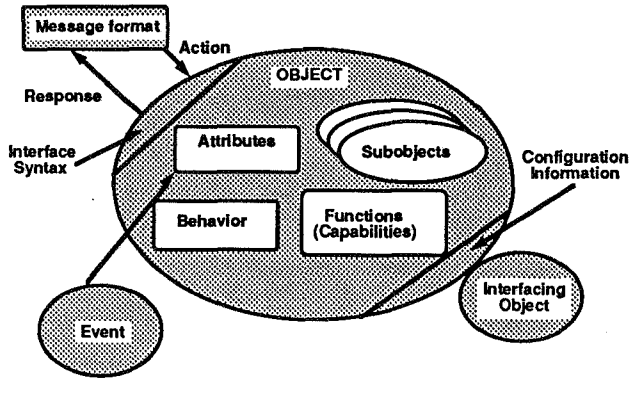


Figure 2 Generic object model

An action is a specific message type which acts as a command. Commands may result in responses such as the ability of the object to perform the indicated action. Other messages may be defined to provide more general information.

Messages require an interface syntax to define the structure and content of the message.

Attributes are data about an object. For a pump, the attributes might include its pressure, speed and temperature. Attributes might also specify the working fluid, capacity, manufacturer and serial number. In general, attributes may be variable data about a device, such as its present state, status, use, location, etc. They may also be invariant data about the device design or construction, etc. Some of the invariant data is the same for all devices of a class, and is carried as object class data.

Objects will usually perform one or more functions. The functions are described as though the "real world" object were performing them.

Objects may exhibit one or more behaviors. Behaviors may describe the way an object performs its functions, such as a telescope slewing in such a way as to avoid pointing at the earth or sun. They may describe the response to detected failures. They may describe the characteristics of action processing and the conditions for action responses.

Objects may contain sub-objects. A sub-object is an object which is wholly contained within or dedicated to another object. A coolant loop contains a circulation pump. The pump contains a bearing. The bearing has a temperature sensor. Each is an object, and has its own class, function attributes, etc.

Objects may also interface with other objects. For example, software is configured with an operating system. It may be configured with certain application software. The same drive motor may be configured with or without a brake.

An object may detect its own events, such as failures and off-nominal conditions. But it is more common for an object to be monitored by another object to prevent a possibly failed device from providing incorrect data about itself, resulting in an inappropriate failure response.

Each object must have an object class which defines the template for the object. The object class identifies each attribute, describes the functions and behaviors, defines the syntax and

format of messages, and identifies sub-object and interfacing object classes.

Messages, attributes, actions, functions and behaviors all have class definitions or types. The type definitions allow for complex data types to contain other data types. Thus a quaternion is defined to contain a three component vector and a scalar, each of which may defined by units, valid ranges, precision, etc.

Data Model

The standard uses entity-relationship data modeling. An entity is anything someone wants to know something about. An entity may represent an object or a spacecraft device.

Attributes describe an entity. In this context, attributes include everything which describes an entity in isolation from other entities. The features of an object, other than its relationships, are attributes in this modeling.

Relationships describe information about an object as it is associated with other objects. Relationships include information exchanged, or messages, as well as many other types of relationships. Some of the useful relationships are shown in the figures that follow.

Figure 3 shows a "contains" relationship. In the figure, an assembly can contain one or more subassemblies. Hardware trees, indentured parts lists, bills of materials and logistics data bases will use this relationship.

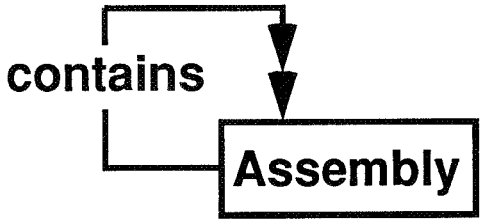


Figure 3 "Contains" relationship

Figure 4 shows the "connects to" relationship. This relationship is used to describe such things as wiring harnesses and assembly sequences.

Figure 5 shows the "communicates with" relationship. This relationship can be used to associate instrumentation with pin-outs on control devices. It can also show logical processor hierarchies and logical interfaces among software.

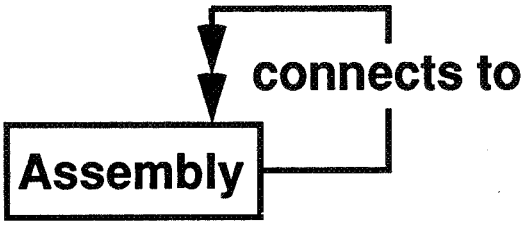


Figure 4 "Connects to" relationship

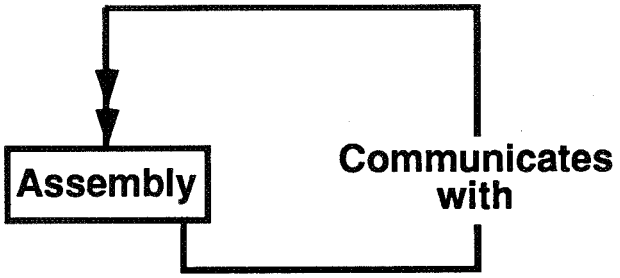


Figure 5 "Communicates with" relationship

The similar "provides value for" relationship connects the source of a data value with the process that uses the data value. A temperature sensor may provide the value for the bearing temperature attribute of a pump device. A square root library routine may provide an input to a computation.

Figure 6 shows the logical decomposition of a system. Requirements will also follow a logical decomposition, and may be allocated to the decomposition products of a system. In the figure, the functional decomposition is carried out to a point that allows functions to be allocated to physical objects such as assemblies, components, devices, software objects, etc.

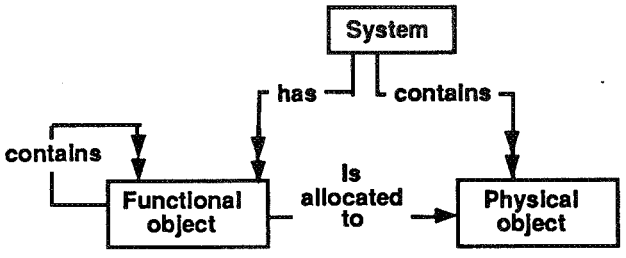


Figure 6 Decomposition relationships

The above examples show just a few of the relationships defined for spacecraft. The encyclopedia allows additional relationships to be defined as needed. The relationships may be defined on line by the users, as can the object classes, data types, and object instances.

Procedures for configuration management have been defined for a single program, but have yet to be defined for multiple programs.

Data Naming

The standard uses attribute based naming. A name is comprised of a verb (if the name is that of an action), an administrative name expression to identify the "owner", and a technical name expression to identify what is named.

No single construct of names appears to meet all of a program's needs. Descriptive name forms with logical, hierarchical structure are preferred for browsing through a dictionary or directory. The logical structure allows a user to locate items in the encyclopedia without prior knowledge of the actual name.

Descriptive names, by their nature, are long and syntactically precise. Software developers and data bases will not usually devote memory to support full descriptive names. Various aliases are not only required, but become the "official" names for a program by virtue of their usage.

The alias names should be constructed to meet two conditions. First, they need to be able to be related to a descriptive name form. The descriptive name form does not need to be actually stored in an encyclopedia if it is derivable from an alias name, the encyclopedia information and a rules set. The encyclopedia can include logic to permit user browsing without having the descriptive name form actually present.

The second condition for an alias name form is that it should be meaningful to the system users. An object instance name should include all of the parts of the attribute name, but may encode these parts and allow portions to be understood from the context. If the name is not meaningful to the users, it will not be used.

The exact construction of names must also take into account the fact that some names are inherited from one program to another. This is true of object class names, and the associated attribute, action, function, and behavior names. These names must not contain an administrative name expression that limits them to a single program. Program specific administrative name expressions are proper for object instances, but may be understood from the context of the object.

The SSFP construct for names needs a significant amount of work to be adapted to the more general usage of spacecraft control across multiple programs.

Data Format, Syntax and Semantics

The standard provides a means for defining the format and syntax of messages, data stores and data structures. Figure 7 illustrates the means for defining messages, using entity-relationship modeling. A message "contains" one or more fields. Each field contains a single data item. The data item is defined by its data type, as was previously described in the data model.

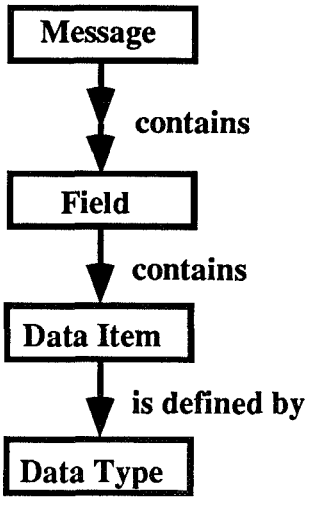


Figure 7 Message definition

Similarly, a data base contains files, which contain records, which contain fields, each of which contains a data item which is defined by a data type.

Data structures are data items which contain sub-elements which are meaningful data items. A 16 bit integer might be constructed such that individual bits represent the state of individual switches in a power control box. Each bit is a defined data item, and the integer may also be defined as a field in a message, or in processing to determine whether the measured switch state matches the currently commanded switch state. In this case, the integer would be defined as a data item with it data type definition, and the type definition would include that each bit is a state variable. In general, data structures are defined by their data types. The data type definitions include parsing rules for the structure, similar to those of a message. The data types for the

individual data items in the structure are also defined by the data structure data type.

The information standard provides data type definitions for all of the program data. The data type definitions are standardized across programs. These type definitions provide the semantics (meaning) of the data. Data types may be added as needed.

DATA TRANSFER

Because there is a global set of data type definitions, it becomes possible to automate data transfers among different local representations. Each local representation needs to map its data to a common transfer syntax and format. Automatic code generators can then be used to for export and import conversions. The export conversion puts local data into the transfer syntax and format. The import conversion transforms the data from the transfer syntax and format to the destination local syntax and format. Thus, each local representation need map only to the common transfer representation to make data globally available.

CONCLUSIONS

Use of an information architecture standard will help reduce the cost of developing and operating spacecraft by providing a common view of all information. This will allow reuse of displays and controls and facilitate adapting control centers to the control of multiple spacecraft.

The proposed information architecture allows different spacecraft with different views of their data to interface with a control center using either a common view of the data for all spacecraft, or separate views specialized to each spacecraft.

The proposed information architecture standard also supports exchange of data between different local representations. It does this by defining a mapping between each individual local representation and a common data transfer representation. Only the common data transfer representation needs to conform to the standard.

REFERENCES

(April 2, 1991). *Flight Software Data and Object Standards*. (Report SSP 30552). Space Station Freedom Program Office

American National Standard for Information Systems - Information Resource Dictionary System (IRDS). (ANSI X3.138). New York, NY: American National Standards Institute, Inc,

Information Resource Dictionary System (IRDS). (FIPS PUB 156). Gaithersburg, MD: National Computer Systems Laboratory, National Institute of Standards and Technology

Information Processing Systems - Open System Interconnection - The Directory. (ISO/IEC 9594, Parts 1-8, (CCITT X.500 0 X,521)). Geneva, Switzerland: International Organization for Standardization

(April 30, 1992). *Fieldbus Application Layer Specification*. Instrument Society of America Draft International Standard

111177 N95-17592

354288 P-8

A STANDARD SATELLITE CONTROL REFERENCE MODEL

Constance Golden[1]

Abstract - This paper describes a Satellite Control Reference Model that provides the basis for an approach to identify where standards would be beneficial in supporting space operations functions. The background and context for the development of the model and the approach are described. A process for using this reference model to trace top level interoperability directives to specific sets of engineering interface standards that must be implemented to meet these directives is discussed. Issues in developing a "universal" reference model are also identified.

INTRODUCTION

The need for a standard approach to identify where standards would be beneficial in supporting space operations functions has been expressed by many people in the field.

- To show a link between the selected standard and the desired benefits of applying the standard.
- To broaden understanding and acceptance of recommended standards.
- To permit benefits of standardization to be spread across several networks.

This "standard approach to selecting standards", based on a functional satellite control reference model, should not be "benefit dependent"; that is, it should permit identification of standards needed to support any specific benefit such as interoperability, cost reduction, etc. Ideally this standard approach would apply to all space operations networks and would allow each standard to be easily tied to its supporting operational function and therefore the benefits could be evaluated.

The Air Force has funded the development of such a standard approach that is based on an extension of the approach in Vol. 7 of the DoD Technical Architecture for Information Management (TAFIM) developed by the Defense Information Systems Agency (DISA). The approach is being applied to the Air Force Satellite Control Network (AFSCN). Representatives from other Government agencies and the commercial space operations community have expressed interest in extending the initial approach by developing a more universal reference model as its basis. Benefits from standardization in one satellite control network can then be evaluated for use in another network using the same functional and interface definitions.

BASIS OF REFERENCE MODEL

Satellite Control Systems can be defined as "A configuration of communications and data processing subsystems that collectively provide the capability to control satellites". Implicit in this definition is the fact that these systems are used in all phases of satellite control, including prelaunch, launch and early orbit checkout, on-orbit operations, and mission completion. Missions include weather forecasting, missile warning, navigation, and communications. Mission execution and mission data processing systems are not included in this definition, although the capability to perform these mission functions can reside on the same subsystems as the ones used for satellite control. Process control systems are an

[1] Loral Space & Range Systems, 1260 Crossman Ave. S80, Sunnyvale, CA. 94089-1198.
The ideas presented in this paper were originally developed under contract with the Air Force Space and Missile Center/CWI.

important subset of satellite control systems because their real-time characteristics drive many of the system performance requirements. Based on this definition of a satellite control system, it would be logical to use already accepted frameworks to tie standards to satellite control functions. In this paper the word "standard" refers to an engineering or product standard (physical/electrical interfaces, formats, protocols, etc.), not an operations standard (procedural or administrative).

Information Systems Reference Model.

The Defense Information Systems Agency/Center for Information Management has derived a TAFIM from the NIST Application Portability Profile and the IEEE P1003.0 OSE models (begun in 1986). This architecture defines a target common conceptual framework or reference model for an information system infrastructure and the specific applications that the information system must support. It also subsumes the widely accepted Open Systems Interface (OSI) reference model within the network services and communications area. This architecture, and associated model, is not a specific system design. Rather, it establishes a common vocabulary and defines a set of services and interfaces common to information systems. DISA's Information Technology Standards Guidance (ITSG) and Adopted Information Technology Standards (AITS) documents describe and support this architecture. The associated AITS identifies standards and guidelines in terms of the architecture services and interfaces. The architecture serves to facilitate the development of plans that will lead to interoperability between mission area applications, portability across mission areas and cost reductions through the use of common services.

Satellite Control Reference Model.

Operations that are unique to satellite control need to be addressed in the Mission Area Applications Section of the TAFIM. Therefore specific satellite control services, such as Timing (those aspects unique to satellite control), Tracking and Data Relay, Telemetry Processing, Command, Resource Control, Contact Execution, and Management were added to those in the generic information systems model. Network Services were also broadened in scope to include earth/space and space/space services. By moving the major service areas into an OSI-like reference structure, it is possible to establish a hierarchical "standard" framework for understanding the relationships between satellite control functions. Figure 1 illustrates this framework or Satellite Control Reference Model (SCRM). The hierarchy is based on levels of functional abstraction, from management services, down to control services, further down to basic computer services and finally down to network and point-to-point communications (layers 1 to 6). All of these unique satellite control services would operate at OSI Application Layer 7. Functions in layers 7b and above in the hierarchy would relate to the Mission Area Applications area in the TAFIM. Functions in layers 7a and below relate to information management systems. Note that all services are not used at each location and that these services are not dependent on location nor are they necessarily automated.

STANDARDS IDENTIFICATION APPROACH

Since the unique satellite control services are not part of core information management systems, they may require standards unique to the satellite control domain that are not covered in the AITS. An approach, based on the SCRM, is needed to accomplish this standards identification. Of special interest is identification of those standards that are most appropriate to reap the benefits of interoperability.

The approach is based on describing the functional flows between each of the satellite control service areas in enough detail so areas where standards would be beneficial can be easily identified and existing standards evaluated to see if modifications/replacements are necessary to achieve the benefit desired. To

this end, a baseline set of functional diagrams for every satellite control service area has been developed. These simple functional diagrams show input, output, and the basic functions provided by each service area. In the future we will generalize the functional descriptions related to levels 7b and above, remove operational procedures inherent in the functional descriptions and move as many currently "unique" satellite control functions into the information management category (black background) as possible.

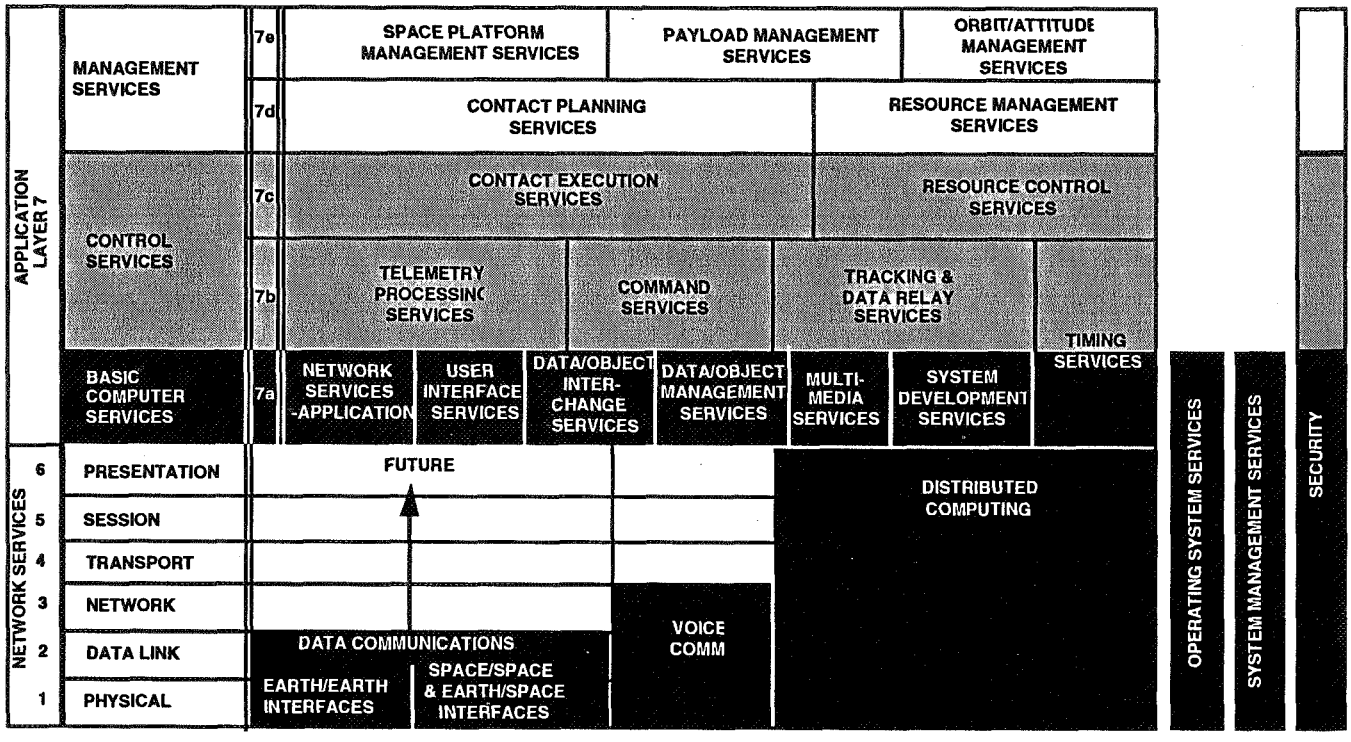


Figure 1 STANDARD FRAMEWORK FOR SATELLITE CONTROL
-Defines functions and interfaces for all services provided-

Overview of Approach.

The specific approach, which facilitates identification of the relevant standards to apply to development and implementation of the satellite control functions, is outlined in the following steps:

Step 1: Describe the desired benefit of standardization for the relevant program. For each major satellite control service area in the SCRM, review the baseline functional diagrams to ensure they match the functional flow for the relevant program. If they need to be modified, do so.

Step 2: Based on the functional diagrams, the desired benefit, and expertise about how various services are provided, identify areas where use of standards would be beneficial, (are needed), and list these areas so they are tied directly to a relevant function and service area.

Step 3: For each of the beneficial standardization areas identified in Step 2, identify what standards are currently being used and which emerging standards, if any, might be applied to that area to achieve the desired benefit. Coordinate with other satellite control organizations for review and feedback, and to ensure commonality among interested groups. List these standards under the appropriate "standard needed" heading on the form used in Step 2.

Step 4: From the compiled information, identify what relationships exist among the standards. Where multiple standards are used for the same satellite control functions, investigate the feasibility of joint adoption of a future common standard and devise an evolutionary path to it.

For functions where standards are needed but none exist or are emerging, describe how such a standard might be developed for the benefit of all networks.

Example of Application of Approach to Identifying Standards.

Figure 2 illustrates a functional diagram for the Contact Planning Services function. The primary inputs, subfunctions, and outputs are shown, along with a short description of how the services are to be accomplished to provide a context for understanding where standards might be appropriate. To increase interoperability, the input-output external interfaces are of primary interest.

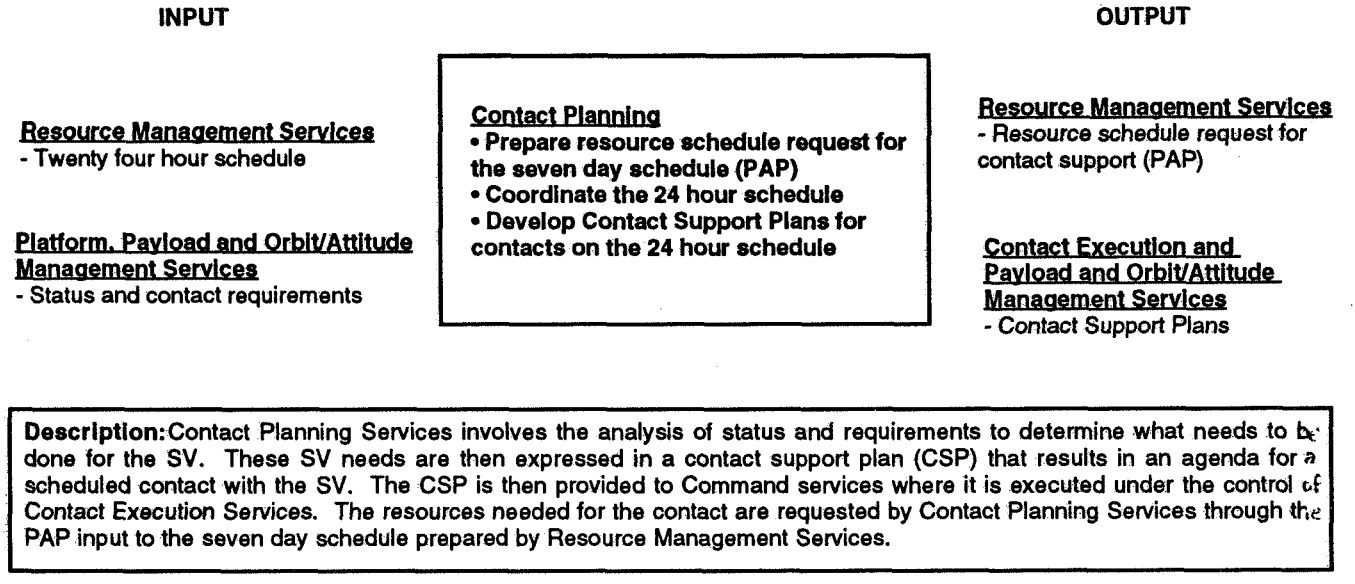


Figure 2 CONTACT PLANNING SERVICES FUNCTIONAL DIAGRAM

Figure 3 portrays a form used to assess where standards would be beneficial for the Contact Planning Services function. The form provides space for indicating what service(s) are interfaced with, the interfacing function, and the context (input, output, HCI) and type (Protocol/Format, Electrical/Mechanical/Physical) of that interface. In addition, there is space for indicating the areas where standards are needed and a column for indicating the current standard status, as defined in the lower part of the Figure. This assessment approach can then be used for each function within the Satellite Control Service areas to assure a level of consistency and completeness in the eventual results.

Once areas of needed standards are identified, the status of any applicable standards can be more readily assessed. The assessment occurs for three time periods: currently, near term and long term. It is effective to record this assessment on the same form shown in Figure 3. For the Contact Planning function, it was noted that there is no standard format for requesting use of network resources by an external user. Standardization on an interface format would facilitate interoperability in scheduling and allocation of the network assets.

SATELLITE CONTROL TECHNICAL REFERENCE MODEL
ASSESSMENT OF STANDARDS NEEDED AND AVAILABILITY

SERVICE AREA	FUNCTION	CONTEXT:	STD STATUS*	WHAT TYPE OF STANDARD WOULD BE BENEFICIAL AND WHY STD 1993 | STD 1994-95 | STD > 1995
CONTACT	PLANNING	INPUT	NOW Limited	• **Formats for expressing SV service/control requirements** AZ/EL acqusition format is std. | None known - should be developed to reduce operator training time/cost and increase I/O
			NOW	• **24 Hour Schedule** RCC format dev. by ASTRO
		USER I/F	VOID	• **HCI/Method for generating CSPs**
			VOID	• **HCI/Method for planning contingency responses** In OOH/TO
		OUTPUT	NOW	• **Format for Contact Support Plans (CSPs)** Existing Std. Format (should be automated where not already done)
			VOID	• **Format for resource request for contact support** Each Program has own format
			NOW	• **Station Configuration** Existing part of CSP

* NOW: Standard is reasonably mature with products that are available today or expected to be available within 6 months.

FUTURE: Standard is emerging and may be subject to change but is generally headed toward stability.

GAP: Standard is available as temporary gap-filler. It is recommended for use only if the organization is willing to take a moderate investment risk because the final standard for the area may or may not be compatable with the gap-filler.

VOID: No standards in the area and no known emerging one. The absence of a standard here may translate into significant risk for long-term planning or investment.

UNSTABLE: Standards are emerging and rapidly evolving.

N/A: No standard is needed in this area. This code will be reserved for areas where at first glance it would appear that a standard might be useful, but further analysis shows that the disadvantages of standardization in this area outweigh any potential advantages.

Figure 3 EXAMPLE ASSESSMENT OF STANDARDS NEEDED AND AVAILABILITY

FUNCTIONAL INTEROPERABILITY APPLICATION

There have been several published definitions for "interoperability" including those in JCS Pub 1-02 and MIL-STD-973. According to the JCS Pub 1-02, interoperability is "The ability of systems, units or forces to provide services to and accept services from other systems, units or forces, and to use the services so exchanged to enable them to operate effectively together". While this definition provides overall guidance, more specific information is needed to tie high level (ORD and CON OPS) interoperability requirements to specific engineering and operational consequences/benefits. One approach is to have overall requirement documents address "how much interoperability" is needed between specified programs or domains. That is, to specify the "degree" of interoperability needed.

Degrees of Interoperability

Figure 4 portrays the breakdown of the "Services" to be exchanged, to achieve general interoperability, into more specific functions as the domain of application becomes narrower.

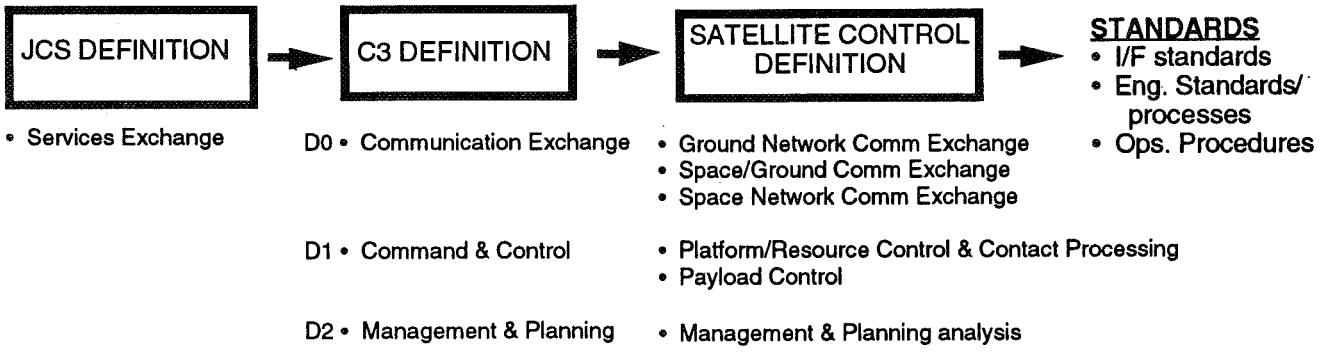


Figure 4 SPECIFIC DEFINITIONS FOR INTEROPERABILITY

Moving from the General Domain to the C3 Domain, "Services" can be broken into Communication Exchange, Command & Control and Management and Planning Services. Moving further into the Satellite Control domain, Communication Exchange can be broken into 3 subsets, (Ground Network, Space/Ground and Space Network), because of the differences in their application environment. Each of these can be specified as a "degree of interoperability" in the satellite control operational environment. Command and Control Services can be broken down into Platform/Resource Control & Contact Processing and Payload Control for the Satellite Control Domain. Each of these can be specified as a degree of interoperability. The Platform/Resource Control & Contact Processing Degree of Interoperability was purposely constrained to routine processing functions and resolution of Level 1 and some Level 2 anomalies because these can be most readily automated and there is high likelihood that many programs will find it beneficial to be interoperable to this degree. The benefits of implementing this degree of interoperability are high, but are dependent on basic Ground Network Communication Exchange being available. Management and Planning Analysis services in the Satellite Control Domain include resolving Level 3 anomalies and require operators to be cross trained on mission and payload information. The benefits of this degree of interoperability are dependent on the "lower" degrees of interoperability being implemented first. Four of these degrees of Satellite Control interoperability are pictured in Figure 5.

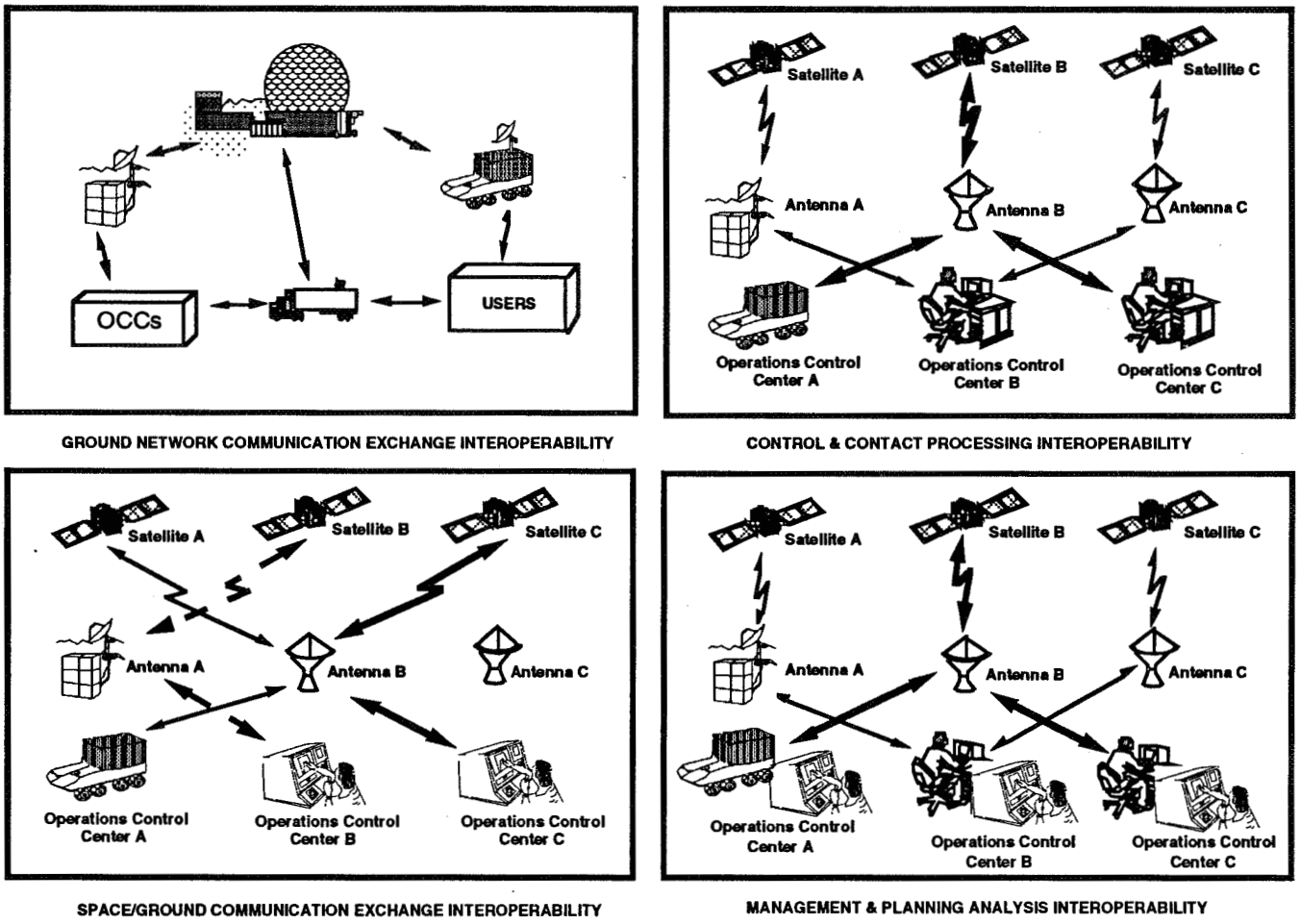


Figure 5 SATELLITE CONTROL DEGREES OF INTEROPERABILITY

Mapping Degrees of Interoperability to Set of Standard Interfaces.

As the degree of interoperability increases from D1 to D3, so too does the emphasis on higher levels of functional abstraction represented in the SCRM. As shown in Figure 6, Ground Network Communication Exchange Interoperability (D1-a) is accommodated almost entirely within the lower six OSI layers, Platform/Resource Control and Contact Processing Interoperability (D2-a) is accommodated almost entirely within OSI layers 7b and 7c, while Management and Planning Analysis Interoperability (D3) is accommodated almost entirely within OSI layers 7d and 7e. Using this correspondence the SCRM can be used to determine the set of interfaces that need to be standardized to support the various degrees of interoperability. Determination of which specific set of standards to select for standardizing these interfaces can then be performed for the environment of interest. The mapping from definition of degree of interoperability to a specific set of standards to be applied is then complete.

CONCLUSION

The standard framework and approach described above is still in the process of being developed. It has the advantage of being based on the already established OSI and TAFIM reference architectures. However, the question of whether the functional interfaces can be defined in enough detail and

generically enough to be able to produce a baseline model that supports all satellite control networks has still to be answered.

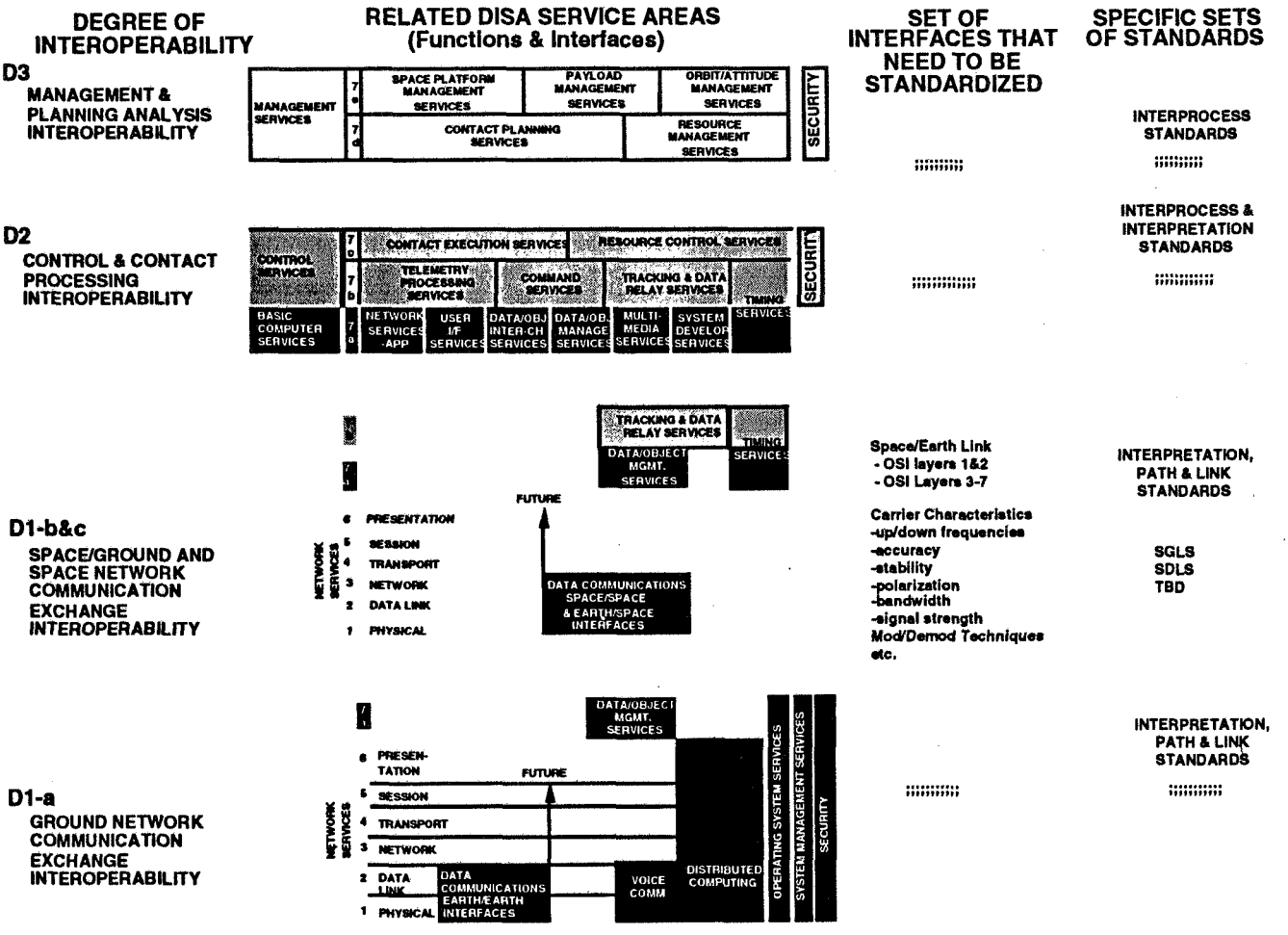


Figure 6 MAPPING DEGREE OF INTEROPERABILITY TO SET OF STANDARD INTERFACES

In the six months that this model has been applied to various situations, it has become apparent that some of the originally identified satellite control unique functions may be able to be defined as generic information systems functions in the future. On the other hand, some of the functions that were initially allocated to information systems are really process control functions and may have to use different standards than those selected for general information systems to meet the real-time response requirements needed. There are several related efforts ongoing and in each a satellite control reference model with standard terms and functional flows has proven to facilitate the analysis.

REFERENCES

1. Defense Information Systems Agency/Center for Architecture, Department of Defense Technical Architecture Framework for Information Management; Technical Reference Model and Standards Profile Summary, (Final Draft), Version 2.0, 1 Nov. 1993

2. Air Force Space & Missile Systems Center, CUE-TR-93-179, Standards for Interoperability, 30 Nov. 93

111178 N95-17593

354290
p. 8

STANDARD PROTOCOL STACK FOR MISSION CONTROL

Adrian J. Hooke

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91109

ABSTRACT

It is proposed to create a fully "open" architectural specification for standardized space mission command and control. By being open, i.e., independent of any particular implementation, diversity and competition will be encouraged among future commercial suppliers of space equipment and systems. Customers of the new standard capability are expected to include:

- o The civil space community (e.g., NASA, NOAA, international Agencies).
- o The military space community (e.g., Air Force, Navy, intelligence).
- o The emerging commercial space community (e.g., mobile satellite service providers).

INTRODUCTION

In response to declining space budgets, the U.S. civil and military space communities both have a critical need to significantly reduce the cost of operating spacecraft, while simultaneously accommodating requirements for increased mission flexibility and capability. The emerging commercial space community has a similar need for low-cost "off the shelf" command and control systems that reduce the need for capital and operating investment.

Standardization has emerged as a key weapon in the conflict between new demands for space mission complexity and increasingly limited space mission budgets. The command and control of space mission systems is an area that is ripe for standardization. For lack of standards or guidance, space mission command and control is (by and large) re-invented for each mission; this drives up cost because a constant cycle of system redesign results in customized, non-automated operations that are highly labor intensive.

There is a pressing need to develop and emplace new standard user services that allow many different types of spacecraft, and their supporting ground networks, to appear basically harmonious from the perspective of ground controllers. With such capabilities, the spiral of constant redesign can be broken, automation may be deployed, and operations and maintenance budgets can be contained.

The new services should:

- o exploit rapid ongoing improvements in onboard data processing, storage and autonomy capabilities by encouraging the spacecraft designers to present simpler, more consistent and more mission-independent interfaces to ground operators;

C-7

- o import off-the-shelf technologies by integrating a wide range of emerging commercial data processing and data communications capabilities into cohesive systems that support high performance space mission command and control;

- o enable the mission-independent operation of spacecraft and their supporting ground networks by small teams of multidisciplinary personnel whose productivity is leveraged by the the widespread deployment of automation;

- o be backwards-compatible with existing space systems so that a smooth transition from the present to the future may be observed.

Many off-the-shelf capabilities currently exist; the primary challenge is to import these diverse technologies and to system engineer them into an integrated solution which satisfies the unique requirements of space mission operations.

It is therefore proposed to develop and functionally specify a Space Project Mission Operations Control Architecture - "SUPERMOCA" - which will provide the open systems framework around which the integration and demonstration of multi-vendor implementations of the new approach may occur.

TECHNICAL CONTEXT

To control a remote spacecraft, the user formulates command directives, transmits them, monitors their execution, and takes corrective action in case of anomalous behavior. The spacecraft executes the command directives using various levels of onboard autonomy. The control center and the spacecraft exchange information via a space communications system that includes both ground and space/ground networks.

Users in the control center also perform a similar set of actions to configure, monitor and control the remote ground data acquisition stations which are supporting the spacecraft. To facilitate automation and to reduce human staffing needs, the SUPERMOCA should promote a unified approach towards the command and control of the spacecraft and its supporting ground systems.

In the terminology of Open Systems Interconnection (OSI), the SUPERMOCA resides within the Application layer and draws upon underlying lower layer space communications services.

Figure-1 shows the SUPERMOCA operating over a space data network containing:

- o Standardized space/ground data channels, as defined for the civil mission community by the Radio Frequency and Modulation standards defined by the Consultative Committee for Space Data Systems (CCSDS).

- o Standardized space/ground networks and data links, as defined by the CCSDS Recommendations for Packet Telemetry, Telecommand and Advanced Orbiting Systems.

- o Standardized upper layer protocols, operating efficiently in a "skinny stack" configuration that is currently being defined by the joint NASA/DoD "Space Communications Protocol Standards" (SCPS) development program. The SCPS stack provides fully secure and reliable file and message transfer services in support of the SUPERMOCA layer.

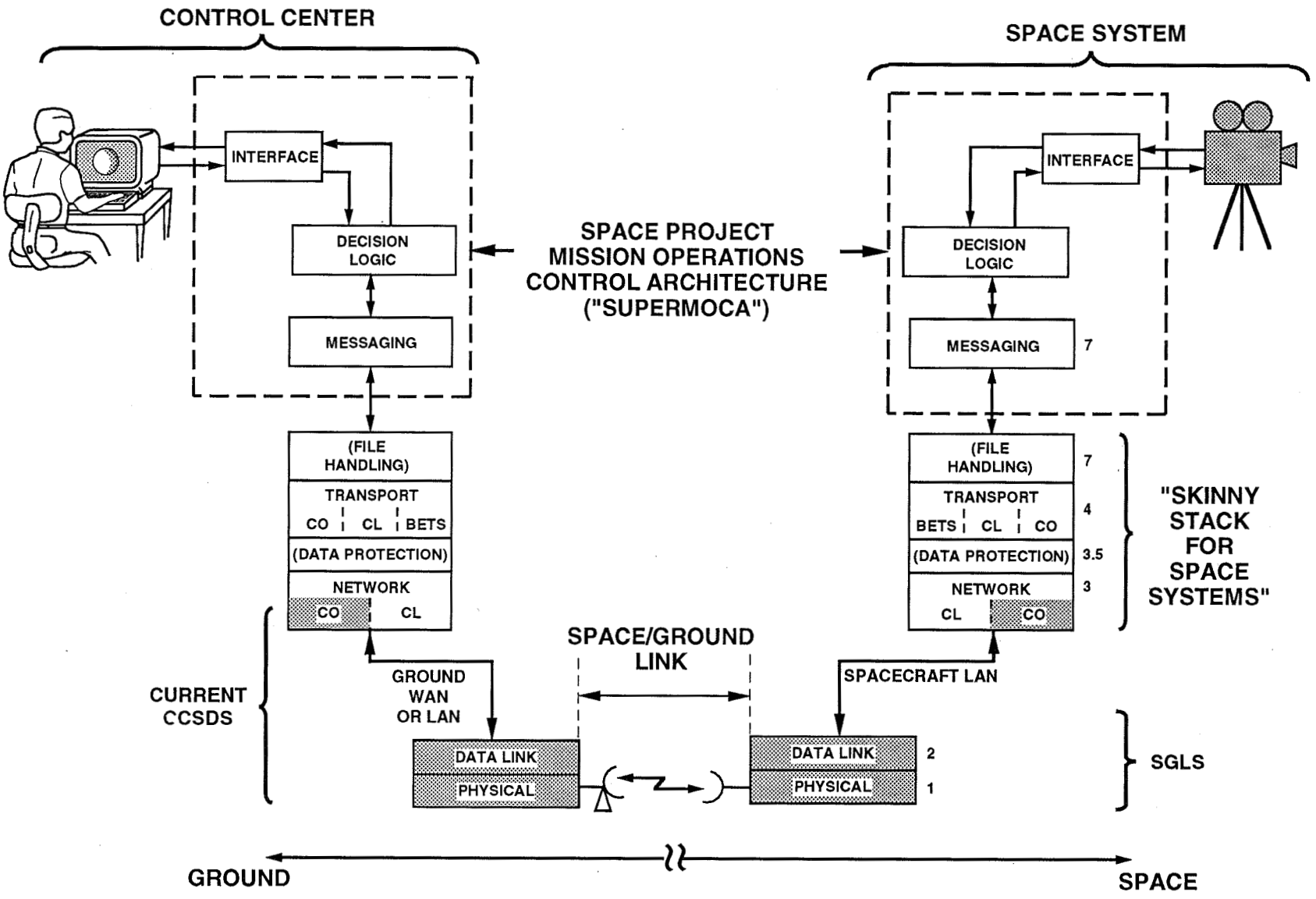


Figure 1. Context of Space Mission Control and Command

ELEMENTS OF THE SUPERMOCA

The SUPERMOCA provides an "upwards" mission control service interface to the mission planning systems which are used to construct the broad profile of desired mission activities. "Downwards" it draws upon a space communications service provided by a stack of underlying standard protocols. Figure 2 shows these service relationships, and postulates a possible internal organization of the layer.

The potential to achieve "backwards compatibility" with existing spacecraft is fundamental to the SUPERMOCA concept: this may be accomplished by locating all of the new SUPERMOCA architectural elements in the control center, and interfacing with the existing communications services that are possibly unique to that spacecraft. By retrofitting existing spacecraft into the SUPERMOCA, a smooth and rapid transition to the future is facilitated.

As currently envisaged, the SUPERMOCA contains five elements. Three of these elements (the Control Interface, the Decision Support Logic and the Space Messaging System) form the heart of the actual process control system. The remaining two elements (the Data Architecture and the System Management Architecture) supply the framework within which the other elements operate. Because they have great significance throughout entire mission lifecycle, the Data Architecture and the System Management Architecture also frame the Mission Planning System.

- Control Interface

  The Control Interface provides a human-oriented mechanism whereby a flight controller can specify and monitor the desired sequence of operations to be conducted in a remote system. It also provides the translation between high-level human directives and actual atomic-level commanded actions at the remote end.

- Decision Support Logic

  The Decision Support Logic provides the capability whereby rules for command execution may be programmed into a distributed inference engine, which may be located wholly on the ground, wholly in space, or partitioned in varying degrees between the two. Commands may only be issued to end effectors in space when they conform to the flight rules that are programmed into the engine. Responses from end effectors will be compared against rule-based expectations, and the Decision Support Logic may take further preprogrammed command actions based on the observed performance.

- Space Messaging System

  The Space Messaging System translates the machine-readable command calls from the user's Control Interface into standard-syntax messages which invoke the desired actions and responses in the remote space system. At the receiving end, generic device manipulations are translated back into concrete, atomic-level actions via the Control Interface.

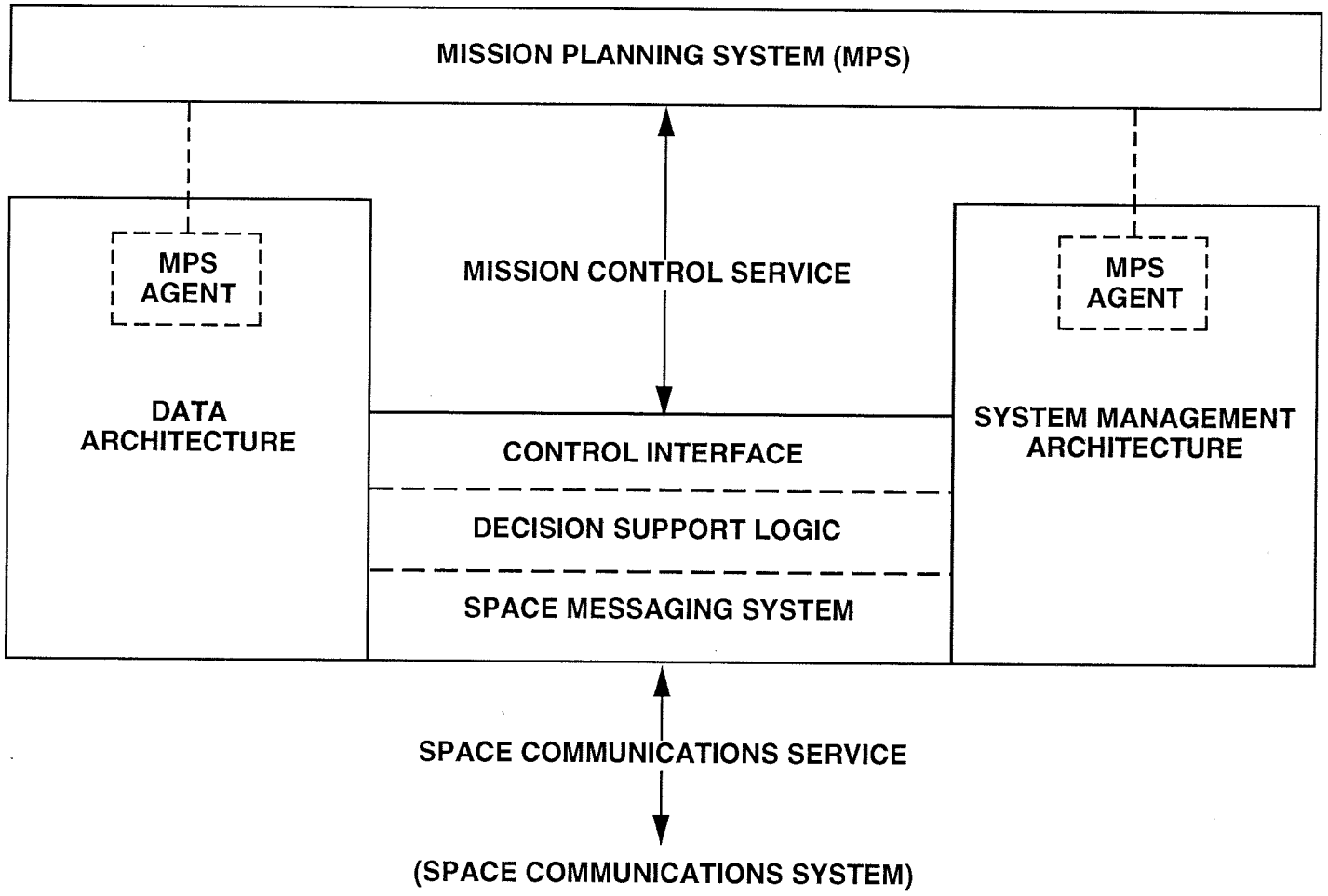


Figure 2. Elements of the Control Architecture

- Data Architecture

  The Data Architecture provides the mechanism whereby the precise characteristics of a concrete spacecraft system can be captured and described in abstract terms. It allows specific spacecraft devices to be described in standardized ways and for this information to be compiled into data dictionaries and encyclopedias. These data descriptions can be gathered starting at the earliest point in the project design lifecyle, thus supporting the progressive and seamless refinement, extension and translation of information from conceptual mission planning, through operations, and into post mission evaluation.

- System Management Architecture

  Space mission process control fundamentally boils down to a problem of meeting mission success and safety-related criteria. The SUPERMOCA accomplishes this through the allocation and control of shared onboard resources, and by managing the relationships which describe how individual systems interact with the operating environment. To achieve this, "operations envelopes" are assigned to individual users, granting them certain "environmental rights" to conduct their operations and consume an allocated share of system resources, and certain "environmental privileges" to perturb the overall system environment. Providing users stay within their assigned envelopes, they are free to operate without detailed supervision. Potentially dangerous activities are precluded via a combination of software controls on command execution, plus hardware inhibits and interlocks which preclude unsafe or undesirable operations from occurring unless the system is prepared for them.

DEVICE MODEL OF OPERATIONS

The SUPERMOCA is conceptually founded in terms of a powerful "device model" of space mission command and control, which is illustrated in Figure 3. Within this model, all of the functions of the space mission are allocated to devices. A device may be physical hardware, a software module which serves as a control interface for hardware, a pure software function, or a combination of these. Each device has a function or functions which it performs: a pump circulates its working fluid; a motor rotates a solar panel; a software module calculates the pointing vector to the sun to guide the solar panel drive motor.

Devices exist at many levels; normally, low-level devices will be aggregated into higher level devices, such that the operator can issue high level commands to the higher level devices, which will themselves orchestrate the function of the low-level devices to accomplish a complex function. A complete spacecraft (and, for that matter, its supporting ground system) is thus composed of many concrete low-level "space devices" which are assembled into complex subsystems that are integrated into an operating mission system.

A space device has a standardized input/output interface through which the external world can know about it, or can control its behavior. This interface can be accessed by sending commands and receiving data or status messages. Attributes describe the device: they include information about the current operation of the device (such as temperature, mode, state, etc.) and descriptors of the device itself (such as serial number, date of manufacture, capacity, operating limits, etc.). Attributes can also include information about the intended use of the device, such as its redlined operational limits.

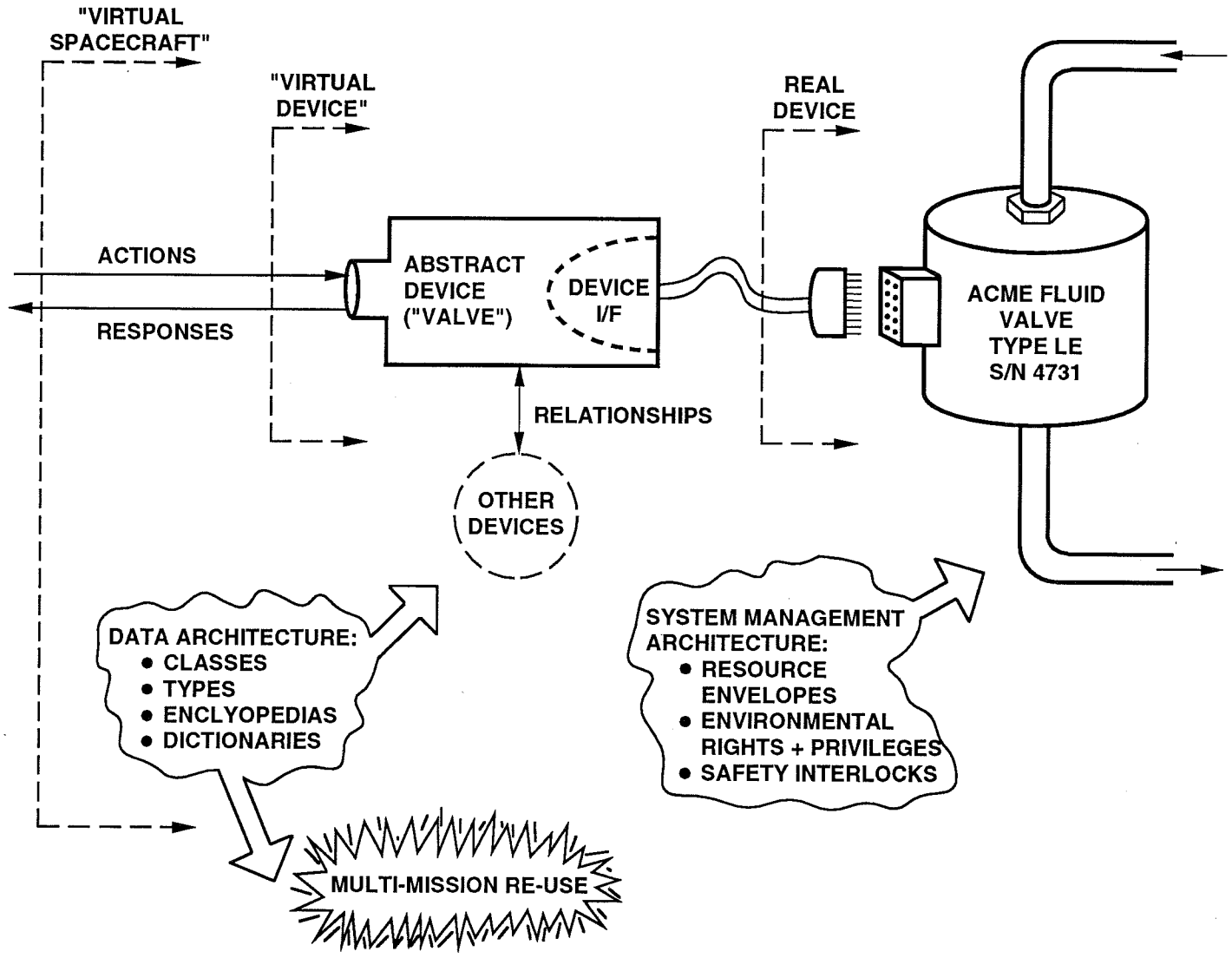


Figure 3. Device Model of Space Mission Control

A device may exhibit one or more behaviors: an oven heats at a rate of 50 degrees per minute; software sends a particular response to an invalid command; an instrument will slew from one pointing direction to another without pointing at the sun. A device may issue messages indicating that specific events have occurred: a parameter may be out of limits, a function may have failed, or a hazardous condition might be noted.

Relationships describe the context for a device. A device may be a part of a higher level assembly, connected to a particular data bus, communicating with another device over the data bus, powered by a a specific power supply, outputting a signal which becomes an attribute of another device, and configured with certain software to perform its functions.

Device types are abstractions which provide a single definition for a family of related "virtual devices" (e.g., all valves, or all pumps, or all pointing actuators, or all voltage regulators, or all transponders share common features; which means that within a family, the same device interface exists for all of them). Therefore the general interface for a device type may be stored in dictionaries and encyclopedias that can be re-used and inherited across multiple space missions.

By masking the uniqueness of a particular space system from its human operator, while providing the tools to progressively capture and exploit knowledge across multiple systems, the device model for space operations will enable the widespread and progressive standardization of the way in which human beings interact with complex, concrete systems in simple, abstracted ways. In particular, adoption of the device model will inject the discipline of standardized system description throughout all phases of space project design: this provides a powerful mechanism for creating a "design to operate" philosophy early in the project lifecycle. From the embryonic stages of mission planning, through operation and post mission evaluation, a seamless flow of data capture is created.

CONCLUSION

It is suggested that a completely standardized mechanism for space mission control is within our reach. By importing and marrying many diverse off-the-shelf technologies, powerful new capabilities may be emplaced that contribute significantly to reducing the cost of operating space systems. Since the needed capabilities will be functionally defined in the form of an "open" specification, the SUPERMOCA will encourage a diverse set of compatible implementations to be placed on-the-shelf by the private sector, for shared use across the entire space mission community.

ACKNOWLEDGMENTS

The work described in this paper was carried out at the Jet Propulsion Laboratory/California Institute of Technology under a contract with the National Aeronautics and Space Administration. The SUPERMOCA concept was developed by many talented individuals affiliated with the AIAA Spacecraft Control Working Group: special thanks go to Bob Easton of McDonnell Douglas; Randy Heuser and Gael Squibb of JPL; Chuck Fuechsel of NASA Headquarters; Connie Golden of LORAL; Brian Buckley of Interface and Control Systems; Captain Vern Veglia of the United States Space Command; and Randy Davis, Elaine Hansen and Sam Siewert of the University of Colorado.

N95-17594

111179

354291

The Space Communications Protocol Standards Program

Alan Jeffries
Science Applications International Corporation
8301 Greensboro Drive
Mclean, VA 22102

Adrian J. Hooke
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109-8099

ABSTRACT

In the fall of 1992 NASA and the Department of Defense chartered a technical team to explore the possibility of developing a common set of space data communications standards for potential dual-use across the U.S. national space mission support infrastructure. The team focused on the data communications needs of those activities associated with on-line control of civil and military spacecraft. A two-pronged approach was adopted: a top-down survey of representative civil and military space data communications requirements was conducted; and a bottom-up analysis of available standard data communications protocols was performed.

A striking intersection of civil and military space mission requirements emerged, and an equally striking consensus on the approach towards joint civil and military space protocol development was reached. The team concluded that wide segments of the U.S. civil and military space communities have common needs for:

- An efficient file transfer protocol
- Various flavors of underlying data transport service
- An optional data protection mechanism to assure end-to-end security of message exchange
- An efficient internetworking protocol

These recommendations led to initiating a program to develop a suite of protocols based on these findings. This paper describes the current status of this program.

INTRODUCTION

The U.S. civil and military space programs are in a state of rapid and turbulent change. Both share the overarching need to more rapidly integrate and deploy space assets, while satisfying expanding mission requirements in an era of extreme cost constraints. Standardization and system interoperability are widely agreed to be the cornerstones towards achieving these goals. Recognizing that both communities share the same industrial contractor base the joint development of common standards and approaches may be expected to reap large benefits in terms of this nation's overall effectiveness in space.

For many years, space agencies have focused on solving the Physical (Layer 1) and Data Link (Layer 2) problems of data transfer through special purpose (noisy, bandwidth constrained, very long time delay) space channels that connect ground users with robotic or piloted space vehicles.

As space missions become more highly networked, requirements are emerging to provide capabilities at the Network layer (Layer 3) and above. Organizations such as the Consultative Committee for Space Data Systems (CCSDS) have begun to address these new needs. At Layer 3, CCSDS currently provides an "Internet

Service" which allows the option to run the full stack of commercially-supported ISO/OSI services (Transport, Session, Presentation and Application) between space and ground. The CCSDS Internet service is the same as the ISO 8473 Connectionless Network Protocol (CNLP). The Internet Service is paired with a special purpose "Path service" (using a low-overhead CCSDS Packet) which functions as a connection oriented network protocol.

There is mounting opinion that neither of these upper layer options can meet all future mission requirements. Concerns have been voiced about the communications overhead and onboard processing resource implications of operating the full ISO/OSI protocol stack in space, particularly with respect to the large amount of on-line protocol associated with the ISO CNLP, the time delay sensitivity of the early implementations of the ISO Class 4 Transport protocol, and the comprehensive but "heavyweight" nature of the ISO Layer 6/7 protocols. The CCSDS Path Service is very limited in terms of addressing capability, and presently has no support of needed upper layer functions such as flow control, end-to-end ARQ, and file transfer.

The strong need to provide a more robust and efficient set of end-to-end communications protocols prompted the initiation of the joint NASA/DoD Space Communications Protocol Standards Technical Working Group (SCPS-TWG) to develop more flexible upper layer protocol options for space missions.

The SCPS-TWG effort determined that many space missions share a common need for an efficient and reliable data delivery service to transfer individual messages, or files of messages, from their source end system to their destination end system error-free and with their sequence preserved. Many missions will also require that these transfers be secure.

Such a service should be standardized, easy to use and able to support a wide range of mission configurations. The protocols which implement the service must conserve onboard resources such as memory, processing power and (especially) communications capacity. They must be capable of supporting rapid, reliable on-line data exchange during brief contact sessions through unique space data channels with a wide range of significant propagation delays.

The development of these standard communication data protocols are being performed by the SCPS-TWG in three phases: Exploratory Analysis (FY93), Standards Development (FY94/95), and Validation (FY95/96). The Exploratory Analysis Phase has been completed and a report is available outlining the analysis efforts and their conclusions.

The remainder of this paper discusses the scope of the SCPS-TWG program, a overview of the Exploratory Phase activities and results, a review of efforts to date on the Development Phase activities and a summary.

SCPS-TWG SCOPE

The primary focus of the SCPS-TWG is to examine standardization of the data communications systems which support on-line spacecraft control. It therefore embraces the end-to-end aspects of the control center processes which are associated with commanding and monitoring the spacecraft and its payload, and returning mission results via a flow of telemetry, during periods when the end systems in space and on the ground are connected and are exchanging data.

Dialog between control centers and remote spacecraft requires frequent (and often two-way) interchange of digital command and response messages through space data links. Such interchange must routinely cope with a data transmission environment that has unique characteristics that are not

encountered in commercial data networking.

The current efforts embrace data communications functions at the Network, Transport and File Transfer (3,4,7) layers. Services of existing (and usually different) underlying civil and military physical and data links layers were assumed.

EXPLORATORY PHASE

The SCPS-TWG Exploratory Phase was performed during the period of October 1992 through December 1993 and culminated in a report on activities and recommendations for initiating the development program. The purpose of this phase was to assess whether there was a common need for data communications protocols between the civil and military space communities, and if there was, what were the common functional requirements. Additionally the Exploratory Phase was to assess the feasibility of adapting existing data communications protocols to meet these common requirements and to define a recommended program of development.

The SCPS-TWG employed an approach of simultaneous top-down and bottom-up analysis. The top-down activity involved a surveying representative, civil and military missions to gather a broad set of functional and performance requirements, and to pinpoint technical constraints intrinsic to space based communications which must be factored into the development of standards. The captured requirements and constraints were allocated to specific protocol layers and fed into the bottom-up activity.

The bottom-up analysis activity involved evaluating the capability of existing off-the-shelf data communications protocols to perform needed space mission functions at each of the layers. As a matter of policy, ISO protocols were the first choice for evaluation to maintain as much conformance as possible with GOSIP and to ensure a high degree of interoperability with ground-based systems. The selection of commercially supported protocols allows the space community to leverage the years of effort that went into engineering their development, and to avoid expensive and duplicative re-invention of capabilities.

The missions surveyed during the Exploratory Phase were as follows:

DoD
- BMD/Brilliant Eyes
- Global Positioning System
- Defense Met Sat Program

NASA
- Space Station
- Earth Observing System (EOS)
- Solar Anomalous and Magneto-sphere Particle Explorer (SAMPEX)
- Tropical Rainfall Measurement Mission (TRMM)
- X-ray Timing Explorer (XTE)
- Advanced Composition Explorer (ACE)
- Discovery Series

The mission survey documented specific functional services by protocol layer (based on the ISO/OSI layered model) which were common between the civil and military projects. These services form the functional data communication requirements that are being supported in the SCPS-TWG protocol stack development, and are listed below in Table 1.

Table 1: Functional Requirements for SCPS Protocols by ISO/OSI Layer

File Transfer (10)	*Transport (9)*
Operations on entire files	Full reliability
Operations on file records	Best effort reliability
Two party file transfer	Minimal reliability
Three party file transfer	Multicasting
User initiated interrupt, resumption and abort	Operation over spacelink bandwidth, outages and delays
Automatic progress monitoring	Segmentation
Automatic interrupt detection	Precedence handling
Automatic resumption after interrupt	Graceful closing of connections
Integrity over operations on entire files	Response to congestion and corruption
Integrity over operations on file records	
	Networking (6)
Data Protection - Optional (5)	Support for multicasting
Access control	Support for multiple routing options
Source authentication	Packet lifetime support with auto discard
Command authentication	Reporting of congestion and corruption
Integrity	Support for precedence handling
Confidentiality	Differentiation between real and exercise data

The bottom-up review of candidate off-the-shelf protocols evaluated potential protocols by asking the following questions:

- Is the functionality provided by the protocol necessary for space use, and if not can the protocol be easily trimmed down?
- Does the selected functionality provide complete support for space use (i.e., is the protocol sufficient in its off-the-shelf state or are additional capabilities required)?
- Does the selected functionality operate efficiently and within the constraints of the space environment, or are modifications needed?
- Can the selected space functionality be achieved with minor change (i.e., is the protocol still commercially supportable after modification) or does it have to be discarded?

If the initial ISO protocol was not able to meet the space application needs, then other commercially available and broadly implemented protocols were assessed (such as those used within the Internet community). Only once all reasonable, off-the-shelf options were discarded, was a solution unique to space use considered. Results of this review are as follows:

File Transfer - OSI FTAM was determined to be too large and couldn't be slimmed down through tailoring. The Development Phase activity is doing a detailed analysis comparing the Internet FTP and Space Station File Transfer protocols to determine which will form the basis for the SCPS-TWG file transfer protocol

Transport - Initially the OSI TP4 protocol was selected, but it too is larger than its Internet counterpart and is expensive to procure. Subsequently a combination of the Internet UDP and TCP protocols are being used to develop the SCPS-TWG transport protocol.

Data Security - the SP3 protocol, based on the OSI NLSP protocol, is being adopted as one option to use with existing or soon to be completed systems. A skinny version of SP3

(called SP3-prime) is being developed to reduce the bit overhead associated with SP3 as an option for future missions.

Network - No existing protocol provided the functionality required with the minimal bit overhead required to optimize use of the spacelink resources. Therefore a custom protocol with elements derived from OSI 8473 and IP is being developed for space applications.

The final suite of protocol services is depicted in Figure 1. As illustrated the SCPS protocol suite can be run over the existing CCSDS protocols used by NASA or the DoD SGLS protocols which achieves the expected interoperability.

DEVELOPMENT PHASE

The SCPS-TWG Development Phase was officially begun in January, 1994 and is planned to run for 33 - 36 months. The first 18 months of this phase are focused on developing protocol specifications for broad community review (equivalent to CCSDS redbooks). The remaining 15-18 months involve two or three rounds of distribution and comment by the US space community culminating in final protocol specifications ready for NASA and DoD adoption.

During the first 18 months the development teams for each of the protocol layers are working in conjunction with a systems engineering group to develop, analyze, and validate the protocol specifications.

The basic approach to this phase is illustrated in Figure 2. Each development team will employ a three pronged development effort consisting of protocol specification development, prototype development, and simulation analysis. The purpose of this approach is to ensure that the specifications developed during this phase have been properly assessed under the broad range of mission architectures represented by DoD and NASA missions.

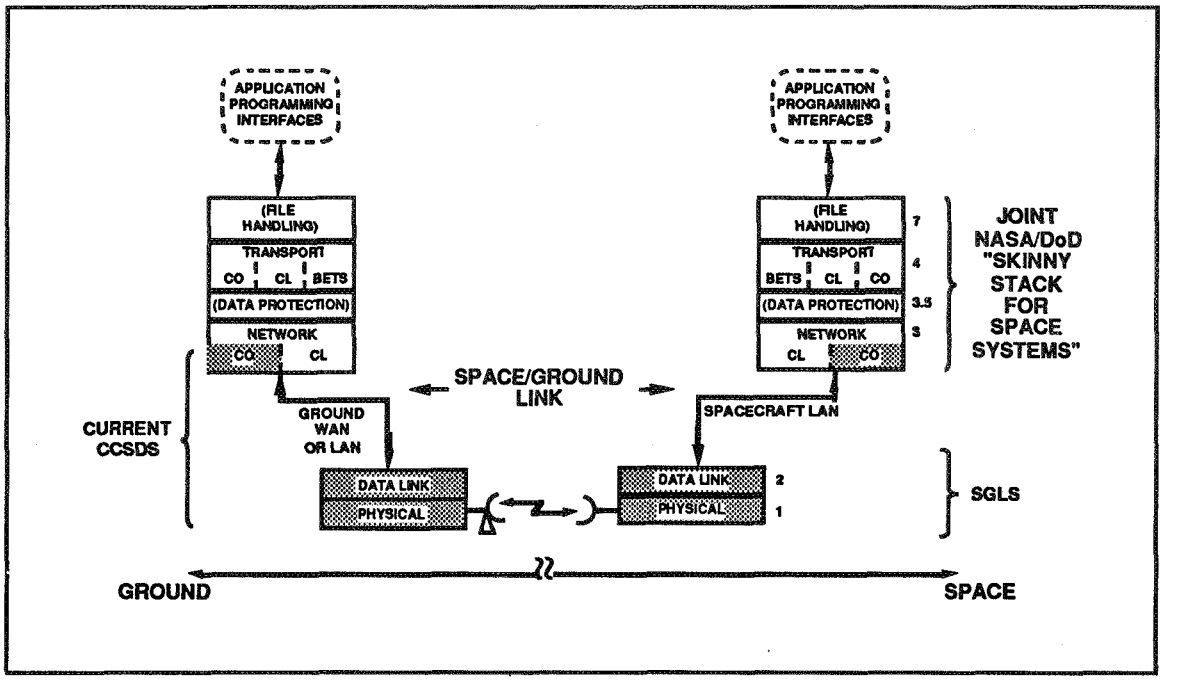


Figure 1: SCPS-TWG Exploratory Phase Recommended Suite of Protocol Services

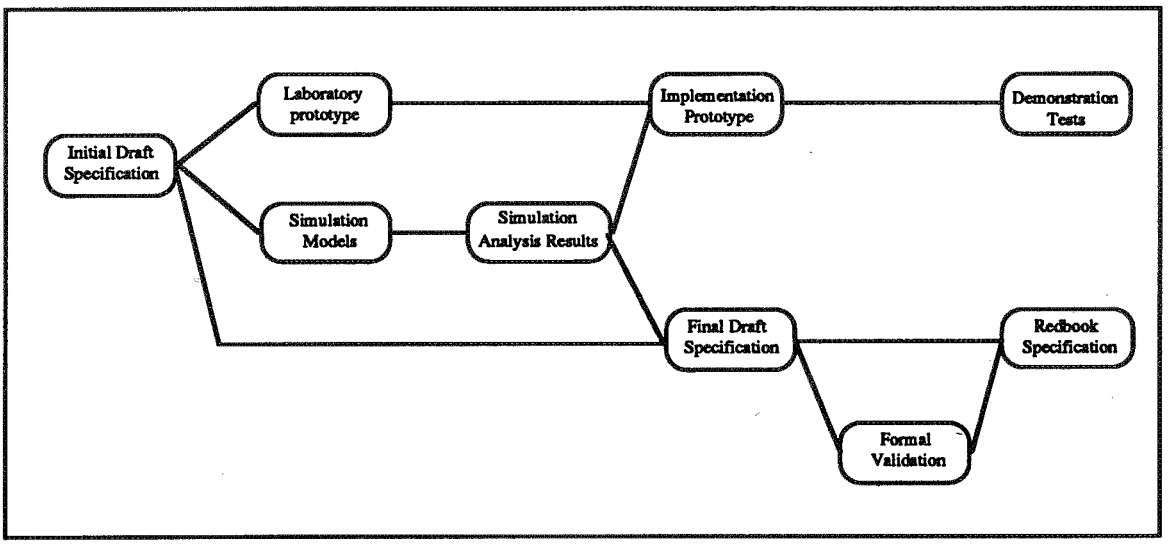


Figure 2: SCPS-TWG Protocol Development Approach

Most of the analysis will be performed via simulation using the MIT NETSIM modeling tool. Each of the protocols will be modeled using NETSIM and then assessed under at least five mission architectures and various scenarios:

- Single earth orbiting satellite communicating through relay satellites
- Single earth orbiting satellite communicating through ground stations
- Single deep space satellite communicating through ground stations
- Multiple deep space satellites communicating through relay satellites
- Multiple earth orbiting satellites communicating through ground stations

Operational prototypes of the protocols will initially be used to benchmark the simulation models to ensure the models accurately represent actual implementations. Once analysis and design of the protocols is complete, the prototypes will be modified to represent the recommended protocols defined in the final specifications. they will then be used in a series of proof-of-concept demonstration tests. These demonstrations are planned to include the use of flight equivalent testbeds such as the GSFC AOS Testbed, "bent-pipe" testing using DoD and NASA on-orbit platforms, and the hosting of the protocols on a spacecraft which has completed its mission phase and is available for the evaluation of new technology concepts.

The detailed schedule of activities which lead to the first set of protocol specifications available for broad community review in September of 1995 is presented in Figure 3. Note that at this time not only will draft specifications be available, but some level of functional prototypes and a sophisticated simulation capability will have also been developed.

In order to ensure community involvement in the protocol development efforts, the SCPS-TWG holds quarterly "Users Forums" identified in the schedule as SCPS-TWG -XX meetings. These meetings are designed to provide community insight into the protocol development and analysis activities. Participation from government, industry and commercial space ventures is welcomed.

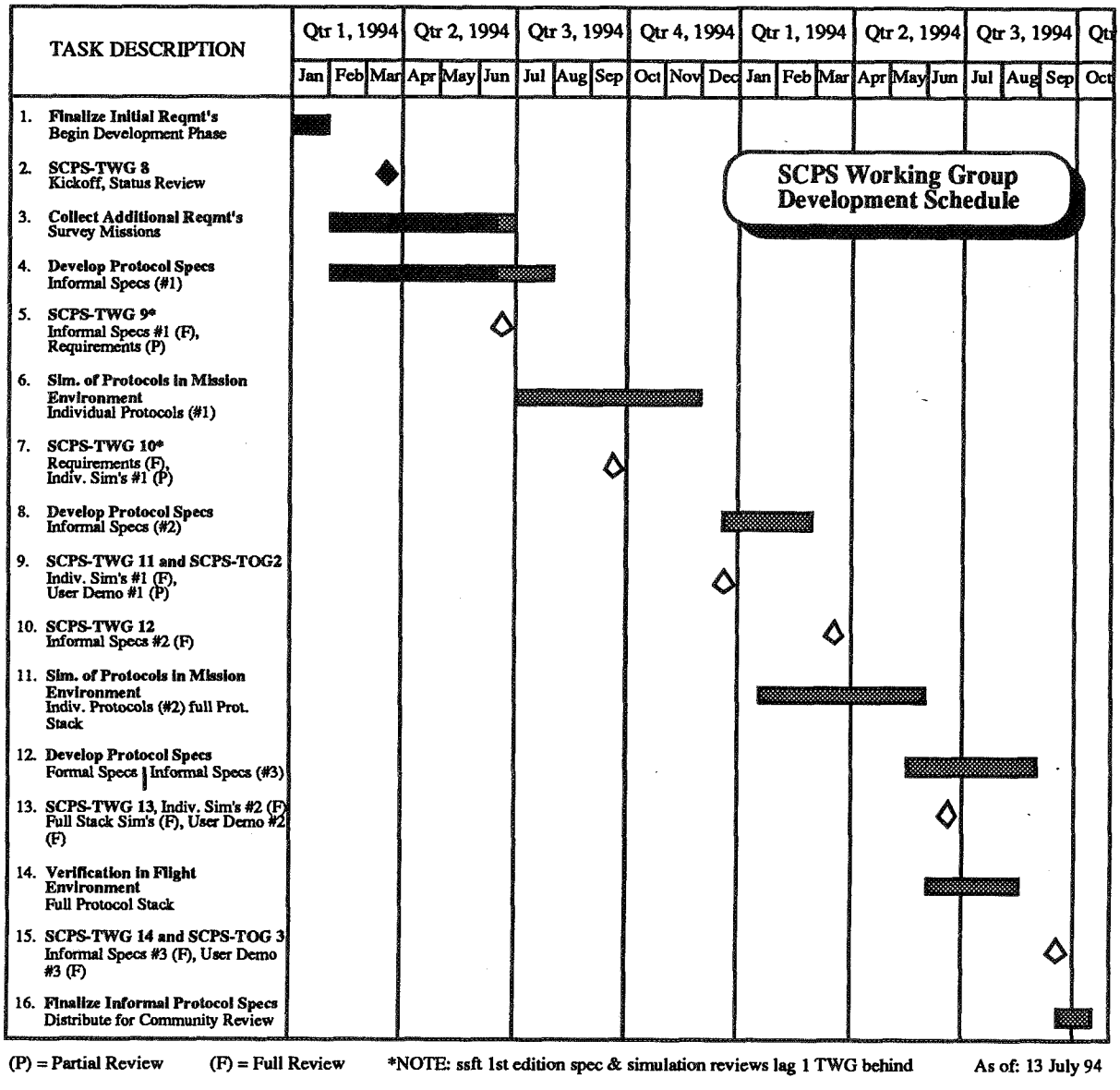


Figure 3: SCPS Working Group Schedule

To date initial draft specifications have been completed and circulated for review for the file transfer, transport and network layer protocols. These drafts are the first in a series of three SCPS-TWG internal drafts which will be developed prior to full community review in September of 1995. Work is proceeding on developing initial prototypes for these three layers with a prototype file transfer protocol based on FTP already completed. Additionally, efforts to develop models of the three current protocols are also in development as is the simulation environment which will model the various mission architectures and data communications scenarios.

SUMMARY

The SCPS-TWG efforts has been highly successful to date in identifying common data communications requirements across U.S. civil and military space missions and defining a program that consists primarily of adapting existing communication protocols to meet the rigors and unique characteristics of space communications.

As one indicator of its success the Defense Information Systems Agency (DISA) has designated the SCPS-TWG activity as its lead effort for developing "thin stack" data communications protocols applicable to a wide range of applications, including airborne, shipboard and in-field communications.

Another indicator is the recent interest in SCPS efforts shown by the commercial satellite venture called Teledesic, which plans to deploy a constellation of 840+ satellites to create a full data communications system equivalent to ground based systems on-orbit.

At the last SCPS-TWG Users Forum, held in June, 1994, DoD representatives working on an existing experimental communications satellite initiated discussions on how to perform "bent-pipe" testing of the SCPS protocol.

Recently, even representatives of the British and French national space complexes have begun discussions on how to become participating members of the SCPS-TWG User Forum. Additionally, the SCPS-TWG team has been coordinating its activities with CCSDS members to facilitate the acceptance of the final protocol by that international body which has shown great interest also.

The importance of this work in NASA can be illustrated by current efforts on the GSFC Mission Operations Control Architecture (MOCA) initiative which has stated that in order to achieve standardized and autonomous operations of GSFC spacecraft communications services of the type now being developed by the SCPS-TWG are paramount. Interest from the NASA missions surveyed in having these protocols was universal.

Continued success of this program is dependent on continued interaction and review by the space community at large. These inputs can have their most positive influence during the current initial 18 month activities of the SCPS Development Phase while preliminary design and analysis are being performed. Critical insights and lessons learned need to be provided by government and industry representatives who have years of space mission experience to share.

111180 N95-17595

354293
p. 6

THE ESA STANDARD FOR TELEMETRY & TELECOMMAND PACKET UTILISATION P.U.S.

J.-F. Kaufeler ESA/ESOC, D-64293 Darmstadt, Germany

ABSTRACT

ESA has developed standards for packet telemetry (Ref.2) and telecommand (Ref.3), which are derived from the recommendations of the Inter-Agency Consultative Committee for Space Data Systems (CCSDS). These standards are now mandatory for future ESA programmes as well as for many programmes currently under development. However, whilst these packet standards address the end-to-end transfer of telemetry and telecommand data between applications on the ground and ***Application Processes*** on-board, they leave open the internal structure or content of the packets.

This paper presents the ESA ***Packet Utilisation Standard (PUS)*** (Ref.1) which addresses this very subject and, as such, serves to extend and complement the ESA packet standards. The goal of the PUS is to be applicable to future ESA missions in all application areas (Telecommunications, Science, Earth Resources, microgravity etc.). The production of the PUS falls under the responsibility of the ESA Committee for Operations and EGSE Standards (COES).

Keywords: Packet Utilisation, Packet Structure, COES.

1. INTRODUCTION

In the past, the monitoring and control of satellites was largely achieved at the "hardware" level. Telemetry parameters consisted of digitised read-outs of analogue channels and status information sampled from registers or relays. These parameters were sampled according to a regular pattern and appeared at fixed positions in a telemetry format.

Similarly, control was performed using fixed-length telecommand frames which contained basic instructions for loading on-board registers or for enabling/disabling switches.

Moreover, the associated space-ground communications techniques guaranteed neither a reliable nor a complete transmission of telemetry and telecommand data.

Through the 1980s, there was a progressive increase in the use of on-board software to implement functions which should logically be performed on-board the satellite rather than on the ground e.g. control loops with short response times, data compression prior to downlink etc. However, this software had to be remotely monitored and controlled using the traditional hardware-oriented techniques.

This imposed significant constraints on the on-board software implementation, limiting its flexibility and consequently hampering the trend towards more on-board intelligence and autonomy.

In order to overcome these problems, the CCSDS recommended the use of telemetry and telecommand packets (Refs. 4 & 5) which provide a high quality space-ground communication technique enabling a flexible exchange of data between an on-board ***Application Process*** and a ground system. An Application Process is a logical on-board entity capable of generating telemetry packets and receiving telecommand packets for the purposes of monitoring and control. It is uniquely identified by an Application ID, which is used to establish an end-to-end connection between the Application Process and the Ground. Many different mappings can be envisaged between Application Processes and on-board hardware. At one extreme, each platform subsystem or payload (or part of thereof) could contain its own Application Process. In a more modest design, a single Application Process, say within the OBDH, could serve many, or even all the on-board subsystems and payloads.

The door was now open to implement a "message-type" interface between ground and space-based applications and thus to move towards the realisation of "process control" techniques.

In 1987 ESA set up the Committee for Operation and EGSE Standards (COES). The primary objective of this group was to define those functions which are common between a satellite checkout system (EGSE) and a satellite control system. Even though these systems are used for different objectives and in different project phases, the logical interface to the satellite is identical and many of the functions are similar. Therefore, a ***common system*** could be used for the pre-launch checkout and post-launch mission operations both within a given project and also across different projects (see Fig.1).

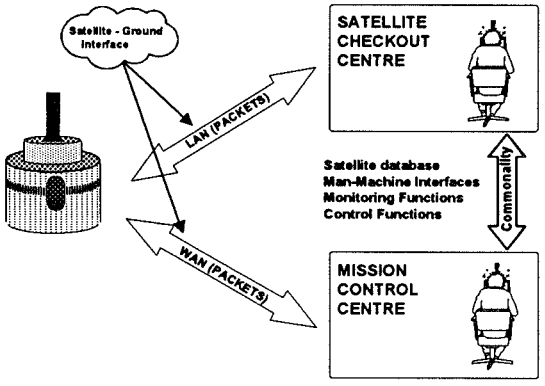


Fig.1 Check-out / Operations Commonality

COES decided to define such a common system for missions using the newly defined ESA Telemetry and Telecommand packets. However, the flexibility introduced by the use of packets leads to the possibility of implementing a given control function in many different ways. It soon became clear to COES that its task was only feasible if a clear satellite-ground interface existed, based on the use of packets.

Consequently, the first task of the COES was to produce a standard which defined precisely ***how*** telemetry and telecommand packets should be used.

2. SCOPE OF THE PUS

The term "Utilisation" is used in the title of the standard, since the intention is that the PUS should address all aspects relating to the use of packets i.e. the circumstances under which they are generated and the rules for their exchange, as well as their structure, format and content.

The PUS can therefore be seen as an interface document defining the relationship between space and ground.

The PUS contains the following elements:

- ⇨ operational requirements relating to satellite monitoring and control functions and to testability;
- ⇨ standards for the secondary data header of telemetry and telecommand packets;
- ⇨ the definition of a set of PUS Services which respond to the operational requirements. A Service specification includes the corresponding on-board Service model and a full definition of all the Service Data Units (SDUs) supported by the Service i.e. the telemetry and telecommand packets;
- ⇨ standards for the data structures and parameter encoding types allowable within packets.

The Operational Requirements cover all aspects of Nominal and Contingency Operations for the full spectrum of mission types and classes. They include generic requirements for:

- ⇨ the different classes of telemetry data to be transmitted to the ground and the circumstances under which the data shall be generated;
- ⇨ the provision of different levels of telecommand access to the satellite to ensure the maximum degree of controllability;
- ⇨ telecommand verification;
- ⇨ the control of on-board software;
- ⇨ the loading and dumping of on-board

memories.

In addition, requirements are identified for a number of "advanced" on-board functionalities, which may only be required for particular classes of mission:

- ⇨ on-board scheduling of commands for later automatic release;
- ⇨ on-board parameter monitoring;
- ⇨ on-board storage and retrieval of data;
- ⇨ transfer of large data units (e.g. files) between space and ground and vice-versa.

The requirements for Contingency operations cover the setting up of a "diagnostic" mode, wherein the ground can oversample selected telemetry parameters for ground evaluation purposes. Also, it should be possible to by-pass on-board functions by ground command and to operate a function in an off-line mode in order to isolate hardware faults.

The Packet Data Field Header (PDFH) is left undefined within the ESA packet standards. However, the PUS identifies a fixed structure for this header for both telemetry telecommand packets, which is shown in Figure 2 below

Version Number	Checksum Type	Ack	Service Type	Service Sub-type
3 bits	1 bit	4 bits	8 bits	8 bits

Telecommand Packet Data Header

Version Number	Checksum Type	Spare	Service Type	Service Sub-type	Time
3 bits	1 bit	4 bits	8 bits	8 bits	Variable

Telemetry Packet Data Header ← Mission Optional →

Fig. 2 : Packet Data Field Headers

The PDFH for telemetry and telecommand packets is identical, with the exception that a telemetry packet may (optionally) contain a time field for datation purposes.

The version number allows for future versions of the data field header and possibly of other aspects defined by the PUS. For example, a new version could be defined for packets containing multiple Service Data Units, as proposed by NASA/JPL for deep-space missions.

The two most important fields in the PDFH identify the ***Service Type*** and the ***Service Subtype*** to which the packet relates. The specification of the "standard" Services provided by the PUS constitutes the bulk of the standard and these Services are covered in more detail in the next section.

In principle, 256 Services and, for each Service, 256 Service Subtypes can be defined. The range from 0 to 127 is reserved for the PUS, in both cases, whilst the range from 128 to 255 is denoted as "mission-specific". The PUS thus has considerable growth capability for the later introduction of new Services or new Service Subtypes within an existing Service.

3. PUS SERVICES

At present, 17 PUS Services have been defined and these are listed in Table 1 below.

Type	Service Name
1	Telecommand Verification
2	Device Command Distribution
3	Housekeeping & Diagnostic Data Reporting
4	Statistical Data Reporting
5	Event Reporting
6	Memory Management
7	Task Management
8	Function Management
9	Time Management
10	Time Packet
11	On-Board Scheduling
12	On-board Monitoring
13	Large Data Transfer
14	Packet Transmission Control
15	On-Board Storage and Retrieval
16	On-Board Traffic Management
17	Test

Telecommand Verification Service

Whilst none of the PUS Services is mandatory, it is expected that all Application Processes would implement this particular Service. Depending on the operational requirements and the on-board capabilities, commands can be verified at all stages: acceptance, start of execution, intermediate stages of execution and completion of execution. The selection of verification stages and whether positive as well as negative acknowledgement packets shall be generated can be done at the level of each individual command which is uplinked.

Device Command Distribution Service

There are 3 sub-services for the distribution of hardware-level commands:

- ⇨ distribution at Telecommand Segment level; these commands require no software for their execution and would be used e.g. for unblocking or resetting the on-board Packet Assembly Controller (PAC);
- ⇨ distribution by the CPDU (Command Pulse Distribution Unit) within the decoder. These are high priority on/off commands which are distributed directly (hardwired) to on-board devices;
- ⇨ distribution by other Application Processes to devices, for example over an internal bus. Such commands may be used for normal operations or in a contingency situation e.g. where the normal higher-level control of the device is not to be, or cannot be, used.

Housekeeping and Diagnostic Data Reporting Service

The ***housekeeping*** sub-service covers the reporting of engineering data to the ground for monitoring and evaluation purposes. In order to adapt to changing operational conditions, the capability exists to define new housekeeping packets (or to re-define the contents of existing packets). Also, instead of systematically transmitting the housekeeping data to the ground, an optional "event-driven" mode is available. Event-driven means that the housekeeping packet is only generated if the value of a parameter within it varies by more than a prescribed threshold.

The ***diagnostic*** sub-service is used to support ground-based troubleshooting, where high sampling rates may be required for selected parameters

Statistical Data Reporting Service

In addition to the direct reporting of engineering data to the ground, summary ***statistical*** data may also be provided, consisting of the reporting of maximum, minimum and mean values of specified parameters over a time interval.

Event Reporting Service

This Service covers reports of varying severity from "normal" reports (e.g. progress of operations) to the reporting of serious on-board anomalies. This provides the mechanism for on-board functions to report to the ground autonomous actions they have taken or events they have detected.

Memory Management Service

This covers all aspects of loading and dumping of on-board memory blocks, as well as performing checksums on specified memory areas on ground request.

Task Management Service

This Service allows the ground to exercise control (e.g. start, stop, suspend etc.) over on-board software tasks managed by an Application Process. For many missions, this level of control may only be exercised in contingencies.

Function Management Service

This Service provides the "normal" mechanism for control of the functions executed by an Application Process (e.g. activate, deactivate, pass parameters etc.)

Time Management Service

This service permits control over the on-board

generation rate of the Time Packet. In the future, this may be extended to cover the use of GPS.

Time Packet Service

This service is constituted solely of the Time Packet which is defined at the higher level of the ESA Packet Telemetry Standard (Ref.2).

On-Board Scheduling Service

For many missions, it will be necessary to load telecommands from the ground in advance of execution, for release on-board at a later time. For example, LEO missions, where operations must be conducted whilst outside of the limited ground passes.

This Service provides the capability for loading, deleting, reporting and controlling the release-status of telecommands in an On-board Schedule. Telecommands may also be time-shifted, without the necessity of deleting and re-loading them with new times.

A telecommand may also be "interlocked" to another telecommand, released earlier in time from the Schedule. That is to say, the release of the telecommand will be dependent on the success (or, alternatively, the failure) of the earlier command.

On-Board Monitoring Service

This Service provides some of the basic telemetry monitoring functions which are normally implemented on the ground i.e. mode-dependent limit, trend and fixed-status checking. Out-of-limit conditions are automatically reported to the ground.

Large Data Transfer Service

For many mission, it is anticipated that the largest desirable packet size may be much bigger than the maximum allowed by the ESA standards. This Service provides for the reliable transfer of a large Service Data Unit of any Type (e.g. a file. a large memory load block or a large report) by means of a sequence of smaller packets. The Service may be invoked either for the uplink or the downlink of a large Service Data Unit.

Packet Transmission Control Service

This Service permits the enabling and disabling of the transmission of packets (of specified Type/Sub-type) from an Application Process.

On-Board Storage and Retrieval Service

This Service allows for the selective storage of packets for downlink at a later time under ground control.

In principle, a number of independent stores may exist, which may be used for different operational purposes. For example, for missions with intermittent ground coverage. packets of high operational significance (e.g. anomaly packets) could be stored in a dedicated packet store so that they may be retrieved first during the next period of coverage.

A "lost packet recovery" capability may also be achieved by systematically storing all event-driven packets on-board.

On-Board Traffic Management Service

This Service provides the capability to monitor the on-board packet bus (e.g. its load, the number of re-transmissions etc.) and to exercise ground control over on-board traffic and/or routing parameters or problems.

Test Service

This Service provides the capability to activate test functions on-board and to report the results of such tests in the telemetry. A standard Link Test ("Are you alive?") Sub-service is provided.

4. MISSION-TAILORING

An important aspect for the wider acceptance of the PUS is that it should be easily to tailor it to the specific requirements of a given mission.

This consideration has been at the forefront whilst developing the standard and is achieved by the following measures:

⇨ a mission may choose to implement only that sub-set of the PUS Services (and/or Sub-services) which it deems

appropriate to its requirements;

- ⇨ the structures defined for the Service Data Units (the telecommand and telemetry packets) identify "mission-optional" fields. These correspond to the "optional" capabilities within a Service (the so-called ***Capability Sets***). If a capability set is not implemented for a particular Service, then the corresponding mission-optional fields may be omitted;

- ⇨ for the data type of each field of the Service Data Units, the PUS only specifies the ***encoding type*** (e.g. real or integer) with the ***encoding length*** being specified at mission-level;

Thus, a mission may remain fully compliant with the PUS whilst incurring no detrimental impact on its packet overhead as a consequence.

5. VALIDATION

Prior to approval of the PUS, and before implementing supporting infrastructures, it was necessary to ensure the correctness, practicability and operational usefulness of the standard. This was achieved by means of a prototyping exercise completed in 1992, which both validated the standard and, at the same time, provided some indicators for possible implementation techniques.

The packet communication techniques were not addressed in this prototype since these have already been independently demonstrated. Instead, the prototype concentrated on the end-to-end application-level aspects, emulating the on-board behaviour in response to the Ground control system.

This prototype (called PUSV) runs on one or two SPARC workstations and at the same time allows modelling of different on-board Application architectures. A reference satellite model (called PUSSAT) was implemented for validation and demonstration purposes.

6. FUTURE PERSPECTIVE

Following an exhaustive review at Agency level during the course of 1993, the PUS in its present version was approved by the ESA Inspector General and thus is now an Agency standard.

The PUS is expected to evolve in the future, in an incremental manner, as new monitoring and control Services become sufficiently mature to be generalised and thus standardised.

ESOC is currently undertaking a major mission control Infrastructure development, the so-called SCOS-II, which is a distributed system based on SUN workstations. SCOS-II will provide full application-level support to missions conforming with the PUS.

COES is also specifying the functional requirements for a generic system to be used for checkout and operation across different projects.

7. REFERENCES

1. Packet Utilisation Standard (PUS), ESA PSS-07-101 Issue 1, May 1994.

2. Packet Telemetry Standard, ESA PSS-04-106, Issue 1, January 1988.

3. Packet Telecommand Standard, ESA PSS-04-107, Issue 2, April 1992.

4. (CCSDS) Packet Telemetry, 102.0-B.2, Blue Book, January 1987.

5. (CCSDS) Telecommand: Part 3, Data Routing Service, 203.0-B.1, Blue Book, January 1987.

6. (ESA) EGSE & Mission Control System (EMCS) Functional Requirements Specification, ESA PSS-07-401, Issue 1, Draft 8 November 1992.

111181 N95-17596

354297 p. 8

PACKET UTILISATION DEFINITIONS FOR THE ESA XMM MISSION

Mr. H. R. Nye
European Space Operations Centre,
Robert-Bosch-Str. 5, 64293 Darmstadt, Germany

ABSTRACT

XMM, ESA's X-Ray Multi-Mirror satellite, due for launch at the end of 1999 will be the first ESA scientific spacecraft to implement the ESA packet telecommand and telemetry standards and will be the first ESOC-controlled science mission to take advantage of the new flight control system infrastructure development (based on object-oriented design and distributed-system architecture) due for deployment in 1995.

The implementation of the packet standards is well defined at packet transport level. However, the standard relevant to the application level (the ESA Packet Utilisation Standard) covers a wide range of on-board "services" applicable in varying degrees to the needs of XMM. In defining which parts of the ESA PUS to implement, the XMM project first considered the mission objectives and the derived operations concept and went on to identify a minimum set of packet definitions compatible with these aspects.

This paper sets the scene as above and then describes the services needed for XMM and the telecommand and telemetry packet types necessary to support each service.

INTRODUCTION

The introduction of packet TM and TC standards (Refs 1 and 2) has lead to a high degree of transparency in the operational interfaces between satellite on-board systems and the related ground systems, offering designers the potential for liberal definition of the data to be transported within TM and TC packets. The complexity of the on-board and ground systems can be greatly influenced by the

- the ***type of interaction*** (or service) and
- the ***structure and content of the packets*** used in this interaction

Only by careful definition of the packet structures and content can it be ensured that the satellite is provided with the information it needs (within command packets) for its operations functions and that the ground is provided with the information it needs (within telemetry packets) for execution of its operational tasks. This becomes even more significant now that satellite systems are increasingly implemented using on-board software.

In preparing for the XMM satellite development programme, it was

necessary to define the on-board services that will be needed to allow the XMM Flight Control System to undertake all mission operations. The services needed are driven by the mission objectives and the associated concept for conduct of the operations needed to satisfy these objectives.

The ESA Packet Utilisation Standard (ESA PUS) (Ref 3) was the reference standard for this the application-level interface and defines a wide range of services considered necessary for all future (unmanned) missions. The process of selecting services from the PUS and tailoring the packet related packet structures to suit the particular needs of any particular mission is referred to as "missionisation".

XMM OPERATIONS CONCEPT

The X-ray Multi-Mirror satellite (XMM) is an observatory in the soft X-ray region of the electromagnetic spectrum and is due for launch on Ariane 5 late in 1999. By virtue of the large collecting area of its telescope and the highly eccentric orbit, XMM will be able to perform long observations (upto 16 hours above 40,000 Km) of X-ray sources with an unprecedented sensitivity.

The satellite and its X-ray instruments will be controlled in real time from the European Space Operations Centre in Darmstadt, Germany, and employing a single ground station, will benefit from upto 22 hours of telemetry and telecommand contact every day. All science and housekeeping data will be transmitted in real time to the control centre for immediate processing (no bulk storage on board). In view of the on-line nature of satellite operations and the nearly continuous visibility from the ground and the desire to minimise on-board complexity, it was appropriate to identify straightforward almost "classical" ways for ground ***on-line control*** of the satellite while making use of the advantages offered by packets. The concept for ***safety management*** during planned (and unplanned) non-contact periods was defined to involve the use of delayed execution (time tagged) commands, a low degree of on-board monitoring and provision of a history of on-board events. Further, it was necessary to provide for ***operations maintenance*** in the form of telemetry management and definition and interaction with on board software.

XMM FLIGHT CONTROL SYSTEMS

A further constraint on definition of the ground/satellite interactions and hence the TM/TC services needed, relates to the Flight Control Systems infrastructure foreseen for XMM. Flight Control Systems for past missions (not utilising packets) involved handling of the individual characteristics of the TM/TC schemes by mission-specific software modules interacting with kernel systems offering basic functions only. These additional modules were needed to convert the peculiarities of the satellite data structures into a form processable within the kernel systems

and understandable to the Flight Operations Teams.

Recent advances in ground system technology (for example in the use of distributed workstation-based control systems and object-oriented techniques) now allow the development of multi-mission control systems offering a palette of services to potential users (missions). By defining data structure standards across the board, commonality between missions can be increased leading to a corresponding reduction in the need for mission-specific elements. This is the ultimate goal of the ESA PUS, a document now entering the approval stage.

The PUS defines the various operational requirements for on board functions and services and goes on to describe the TM/TC packet types and structures needed to support these services. The PUS then defines the format and content of the (variable) "Packet Data Field" being the user-defined part of the packet and including the "Source Data" for telemetry (Figure 1) and the "Application Data" for commands (Figure 3). The PUS further prescribes how the "Data Field Header", within the Packet Data Field is to be used (Figures 2 and 4) : two fixed fields in this header are reserved for identification of the ***Packet Type*** and ***Packet Subtype***. In this way, every packet in the ground or on-board systems is clearly identifiable in terms of its function and the processing needed.

XMM however, with its classical operations concept did not need to take advantage of the wide range of services available within the PUS : using the PUS as a starting point, the XMM project selected those services and related data structures of use in supporting the operational requirements (as documented in Ref 4) for all foreseen XMM mission scenarios .

XMM SERVICES

The services defined for XMM mission operations can be considered to fall into three major categories as follows (as documented in Ref 5):

1) ON-LINE CONTROL

Periodic Housekeeping Telemetry (TM Type 1) is required to permit the ground to derive and monitor the status, health and performance of the satellite systems and instruments.

Device Commands (TC Type 2) are required to configure the on board hardware using two subtypes :

- Pulse commands (Subtype 1)
- Register load commands (Subtype 2).

Telecommand Verification Service (TM Type 3) is required to allow the ground to positively verify all uplinked commands. Dedicated packets are required for each uplinked command indicating

- Successful Acceptance (Subtype 1)

- Unsuccessful Acceptance (Subtype 2)
- Successful Execution (Subtype 3)
- Unsuccessful Execution (Subtype 4)

Non-Periodic Telemetry (TM Type 4) is required to convey information related to non-periodic events (not contained in the periodic telemetry) to the ground. The service must provide for

- Event Reports (Subtype 1) for events of operational significance
- Exception Reports (Subtype 2) for notification of non-fatal errors
- Major Anomaly Reports (Subtype 3) for notification of major on-board anomalies

Task Management Service (TC Type 5) is required to control and interact with on-board software tasks. The service must provide for

- Start task (Subtype 1)
- Stop task (Subtype 2)
- Load task functional parameters (Subtype 3)
- Mode Transition (Subtype 4)

Science Telemetry (TM Type 15) is required to transport data from the XMM science instruments to the ground.

2) SAFETY MANAGEMENT

Time Tag Commands (TC and TM Type 7) are required to effect operations requiring well-defined execution times or which need to be executed in periods of non coverage or to ensure that the satellite is returned to its nominal state after any critical operation. The service must provide for

- Load a command into the time-tag buffer (Subtype 1)
- Report a summary of the contents of the time-tag buffer (Subtype 2)
- Report all commands in the time-tag buffer in detail (Subtype 3)
- Report a selected command in the time-tag buffer in detail (Subtype 4)
- Delete a selected command from the buffer (Subtype 5)
- Delete all commands in the time-tag buffer (Subtype 6)

On-Board Monitoring Service (TC and TM Type 8) is required to monitor a maximum of 30 parameters during periods when the ground does not have visibility of the spacecraft and to retain the results. The service must provide for

- Enable and refresh monitoring (Subtype 1)
- Disable monitoring (Subtype 2)
- Add to monitoring list (Subtype 3)
- Delete monitoring list (Subtype 4)
- Report the monitoring list contents (Subtype 5)
- Report the results of limit/status checks (Subtype 6)
- Report the minimum and maximum values over the period enabled (Subtype 7)

Non-Periodic Packet Storage Service (TC Type 11) is required to store all non-periodic packets (TC verifications reports, event reports, exception reports and major anomaly reports) in a cyclic buffer to permit the ground access to non-periodic packets generated at times when the ground has no contact with the satellite (planned and unplanned). The service must provide for

- Report stored packets (Subtype 1)
- Enable and refresh packet storage (Subtype 2)
- Disable packet storage (Subtype 3)

3) OPERATIONS MAINTENANCE

Memory Maintenance Service (TC and TM Type 6) is required to allow the ground to maintain the on-board software as needed to compensate for hardware failures, to resolve software non-compliance with design requirements, to account for new requirements or to enhance system performance. The service must provide for

- Load memory (Subtype 1)
- Dump memory (Subtype 2)
- Calculate Memory Checksum (Subtype 3)

Telemetry Management Service (TC and TM Type 9) is required to manage generation of telemetry packets by any particular application (on board subsystem or instrument). The service must provide for

- Report packet generation status (Subtype 1)
- Enable generation of all TM packets (Subtype 2)
- Disable generation of all TM packets (Subtype 3)
- Enable generation of specific TM packets (Subtype 4)
- Disable generation of specific TM packets (Subtype 5)

Telemetry Definition Service (TC and TM Type 10) is required to allow the ground to define new housekeeping packets (for the satellite systems only) if necessary for troubleshooting or anomaly rectification. The service must provide for

- Report new housekeeping packet definitions (Subtype 1)
- Define new housekeeping packet (Subtype 2)
- Delete new housekeeping packet definition (Subtype 3)

Test Commands (TC Type 13) are required to confirm that the on-board link to any application is alive.

DATA STRUCTURES

The definition of packet types is only completed when the data structures needed for each of the identified packets types and subtypes are expanded down to field level as is foreseen in the PUS. The purpose of each field, its length and its format must finally be agreed between satellite system and ground system developers. This final stage in the missionisation process for XMM has been initiated and is also documented in Ref 5.

CONCLUSION

This paper has summarised the way in which the XMM project has gone about selecting the on board services needed to fulfil the objectives of the mission and has outlined the data types defined in support of those services. One can draw three distinct conclusions from this process :

- Data required for the execution of satellite mission operations must comply with certain standards if it is to ensure that such operations are conducted in a safe and reliable manner.

- The structures defined for the transport of the data must follow established guidelines if the full advantages of the packet telemetry and telecommand standards are to be realised.

- Compliance with the derived packet structure requirements must be established across the whole satellite at system level if the benefits in common data structure definitions are to be felt in ground system development.

REFERENCES

1 ESA Packet Telemetry Standard (PSS-04-106, Issue 1, January 1988)

2 ESA Packet Telecommand Standard (PSS-04-107, Issue 2, April 1992)

3 ESA Packet Utilisation Standard (PSS-07-101, not yet approved)

4 XMM Operations Interface Requirements Document (XMM Ref : RS-PX-0028, Issue 2a, March 1994)

5 XMM Packet Structure Definition (XMM Ref : RS-PX-0032, Issue 2, 29 June 1994)

KEYWORDS

Packet Utilisation, Packet Structures, Missionisation, TM Packets, TC Packets, XMM

ACKNOWLEDGEMENTS

The author wishes to acknowledge the efforts made in the establishment of the XMM OIRD and the XMM PSD by members of the XMM Project at ESTEC and of the XMM team in ESOC.

SOURCE PACKET HEADER (48 bits)							PACKET DATA FIELD (VARIABLE)		
PACKET ID				PACKET SEQUENCE CONTROL		PACKET LENGTH	DATA FIELD HEADER	SOURCE DATA	PACKET ERROR CONTROL (Optional)
Version Number	Type	Data Field Header Flag	Application Process ID	Segment-ation Flags	Source Sequence Count				
3	1	1	11	2	14				
16				16		16	Variable	Variable	Variable

Figure 1. Telemetry Source Packet Fields (from Ref 1)

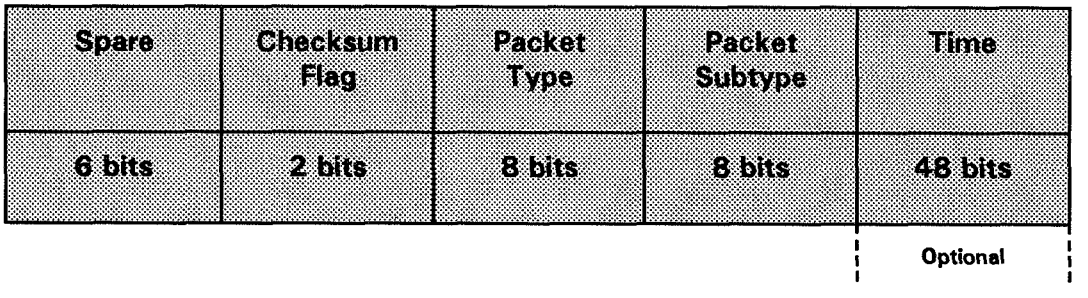


Figure 2. Telemetry Packet : Data Field Header (from Ref 5)

PACKET HEADER (48 bits)						PACKET DATA FIELD (VARIABLE)		
PACKET ID				PACKET SEQUENCE CONTROL		PACKET LENGTH	DATA FIELD HEADER	APPLIC-ATION DATA	PACKET ERROR CONTROL (Optional)
Version Number	Type	Data Field Header Flag	Application Process ID	Sequence Flags	Sequence Count				
3	1	1	11	2	14				
16				16		16	24	Variable	16

Figure 3. Telecommand Packet Fields (From Ref 2)

Spare	Checksum Type	Ack	Packet Type	Packet Sub-Type
2	2	4	8	8

Figure 4. Telecommand Packet : Data Field Header (from Ref 5)

111182 N95-17597

354299
P-9

Use of Data Description Languages in the Interchange of Data

Authors: M.Pignède, B.Real-Planells; European Space Operations Centre (ESOC),
Robert Bosch Strasse 5, 64293 Darmstadt, Germany; Tel: +49 6151 902216

S.R.Smith; Logica UK Ltd,
75 Hampstead Road, London NW1 2NT, England; Tel: +44 71 637 9111

ABSTRACT

The Consultative Committee for Space Data Systems (CCSDS) is developing Standards for the interchange of information between systems, including those operating under different environments. The objective is to perform the interchange automatically, i.e. in a computer interpretable manner. One aspect of the concept developed by CCSDS is the use of a separate data description to specify the data being transferred. Using the description, data can then be automatically parsed by the receiving computer. With a suitably expressive Data Description Language (DDL), data formats of arbitrary complexity can be handled.

The advantages of this approach are that (a) the description need only be written and distributed once to all users (b) new software does not need to be written for each new format, provided generic tools are available to support writing and interpretation of descriptions and the associated data instances. Consequently the effort of "hard coding" each new format is avoided and problems of integrating multiple implementations of a given format by different users are avoided. The approach is applicable in any context where computer parsable description of data could enhance efficiency (e.g. within a spacecraft control system, a data delivery system or an archive).

The CCSDS have identified several candidate DDLs: EAST (Extended Ada Subset), TSDN (Transfer Syntax Data Notation) and **MADEL** (**M**odified **A**SN.1 as a **D**ata **E**scription **L**anguage -- a DDL based on the Abstract Syntax Notation One - ASN.1 - specified in the ISO/IEC 8824).

This paper concentrates on ESA's development of MADEL. ESA have also developed a "proof of concept" prototype of the required support tools, implemented on a PC under MS-DOS, which has successfully demonstrated the feasibility of the approach, including the capability within an application of retrieving and displaying particular data elements, given its MADEL description (i.e. a data description written in MADEL).

This paper outlines the work done to date and assesses the applicability of this modified ASN.1 as a DDL. The feasibility of the approach is illustrated with several examples.

Keywords: **Heterogeneous Environments, Automated Interchange, Data Description, Modified ASN.1, Demonstrated Feasibility**

The Problems of Interchanging Data

The problems of data interchange are primarily those associated with providing the destination (potentially a different computer environment) with **all** the information it needs to be able to interpret the received data. At present, a typical data interchange system is dedicated to a particular flight mission or project. Data is acquired, processed, shared to some level with other members of the investigating team and eventually archived. Further, documentation of the data and its format may not be complete and up-to-date. This practice results in the need for a different interchange data system for each mission and makes the reuse of data and software at some future period difficult.

Overview of the CCSDS Approach

The CCSDS approach is to provide standardised techniques for the automated interpreting of data products in a heterogeneous computer environment. It puts no constraint on the format of the user data and can thus accommodate formats developed by other organisations or user communities. It offers a data labelling scheme

which permits associating a data instance and its (in principle separate) complete and unambiguous description. Further, it allows the development of generic software to support the retrieval, access, parsing and presentation of data to satisfy particular application and user needs. Three major stages are identified in the data interchange process and are explained in this paper.

The Data Interchange Requirements

To achieve the stated aims, the fundamental requirement that has to be fulfilled is the unambiguous description of data that has to be interchanged between users separated by both time and space. When received, data have to be understood fully. This means that the receiver has to not only be able to read the data from the transfer media and understand the basic physical elements of the data (e.g. an integer), but also the receiver must be able to understand the real world meaning of the data. For example, there is no point in the receiver knowing that the 10^{th} and 11^{th} bytes of the received data are an *integer of value 145*, if he also does not know that this conveys the *temperature of a spacecraft instrument in degrees Kelvin.*

Further, it is desirable that data products be automatically transferable without requiring any conversion of their contents. It is also desirable that transfers would be possible regardless of the source or destination computer types. In other words,

- the data must remain in its original form, i.e. does **not** need to be converted in order to be interpreted at the destination;
- the destination needs to know nothing about the source, regardless of how different the computer systems may be.

Thus, the approach presented in this paper is to use a data description processable on any computer which will allow the transmission of data in its native form, i.e. no data encoding will be required.

The Three Stages of the Data Interchange Model

In an attempt to establish a logical model of the whole data description process, an initial assumption was made, that is that the description of data can be cleanly split between the physical description (called the **syntax description**) and the meaning of those physical elements (called the **semantic description**). In fact, as the problem domain was studied further, it was realised that the description should be split into 3 parts, the same physical description as originally perceived, but the meaning component was really 2 separate components: the meaning of the physical elements being exchanged and furthermore the methods of combining the physical elements or relationships between these elements. Indeed, a generator of data products not only wishes to convey the physical data and its meaning, but also how they are intended to be interpreted taking into account their context. Figure 1 shows the model defined by the CCSDS to represent the following stages of data description:

Stage 1: Data is initially perceived as a group of bits accessible from a physical medium and is read in combination with its syntax description. This tells the user the physical layout of the data on the medium, which bits are grouped with each other and how they should be read by common computer hardware (e.g. as integers, reals, bit masks, etc). The syntax description must include all the physical bits, all the bits and groupings must be named in some way. At this stage it is possible to manipulate the basic values, for example converting to another physical representation, e.g. from VAX reals to IBM reals.

Stage 2: The next stage is when the physical data is read in conjunction with its basic semantic description: this only describes those parts of the data that the user is interested in fully understanding, i.e. the basic semantic

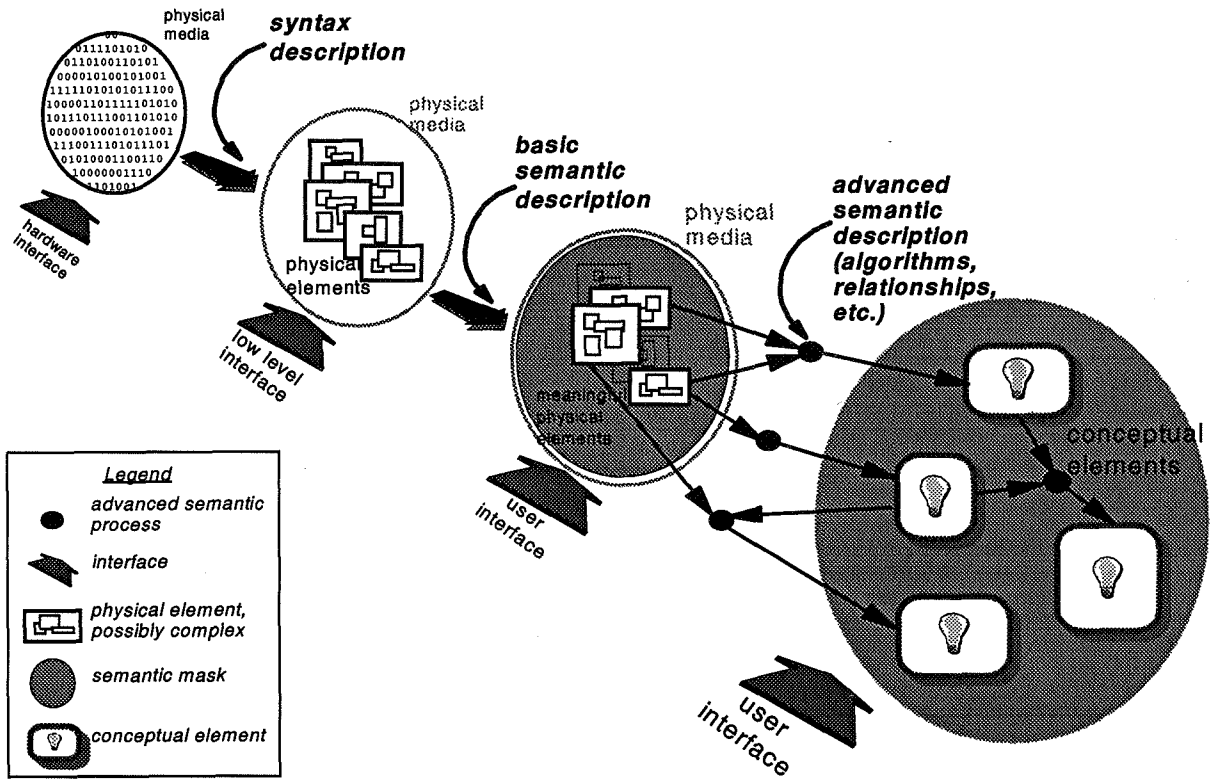


Figure 1: **Data Interchange Logical Model**

description could be perceived as a filter that covers the physical layout of the data and only allows those elements that are of interest to be seen. In this stage, the relevant information is typically: meaning, units, aliases, scaling factor. The elements which are not called out in the basic semantic description are still physically present, but not of interest to the user. Different users may use different semantic descriptions which will act as different masks over the same physical data and therefore perceive the data product in a different manner. Interfacing to the data at this level is the most common method within present day data processing systems.

Stage 3: The final stage in the understanding of the data is to add advanced semantics: these do not define the static meaning of each piece of data, but the way different pieces of data are related to each other. This may be specified in a natural language or as a mathematical algorithm or as a process defined in a programmable computer language. The elements thus created are called *conceptual elements*, as they never actually physically exist: they are pieces of information that are carried within an application domain. If they were ever written as a piece of physical data to a physical media, then this whole process would start from the beginning, i.e. with applying their syntax description. Figure 1 shows a ● at the point where the advanced semantics are applied, this is to indicate that some form of processing takes place. For example, the data may contain two integers, named *mass* and *volume* and for these named physical elements the advanced semantic description may also have a definition of *density*, as being a relationship between the two physical elements mass and volume.

Applicability of MADEL as a DDL

As the model shows, a mechanism must exist at each stage in order to link the data perceived so far to the syntax or semantic information being added. This paper focuses on stage 1: MADEL has been developed to perform the syntactic description of the physical data. Regarding stage 2, it is noted that CCSDS has developed a DDL

called PVL (Parameter Value Language) suited to handle the basic semantic description (see Reference-2). Regarding stage 3, processing is in practice very much application specific. Stages 2 and 3 are not covered further in this paper.

The reasons for selecting ASN.1 (see Reference-1) as a starting point of the MADEL development are that ASN.1 is an International Standard, is familiar to the CCSDS community and intuitive to understand, even by the non expert users. It must be noted at this point that ASN.1 is used as a separate DDL for describing data formats and not as it was originally designed that is, embedding within the data to be transferred auxiliary information ("ASN.1 encoding"). Hence, in the process of deriving MADEL, limitations were made as were some extensions added in order to fulfil the requirements as stated earlier. The most significant modifications to the ISO 8824 ASN.1 specification are detailed below:

Physical Representation of Base Types

Typical MADEL description statements are `SpacecraftID ::= OCTET STRING` or `Width ::= REAL`. Since the intention is to keep the physical data in its native form and to have the MADEL description in accordance with Reference-1, defaults must be adopted everywhere a complete description of the data is not supplied directly by the MADEL description itself. For example how many octets are in `SpacecraftID` and how many bits are used for the mantissa of `Width` and where are these bits located within the real number layout? These pieces of information must be provided in some manner, down to the bit level where necessary and are catered for by MADEL.

For example, real numbers are described using the MADEL `REAL` type whose defaults describe ANSI/IEEE 754 floating point numbers or using a generalised form of the `REAL` type: this type, to be used for any non ANSI/IEEE 754 real numbers, provides the capability to fully describe a real number physical layout in terms of mantissa, exponent, base, etc; thereby allowing to compute at the destination the real number value from:

$$ValueOfRealNumber = -1^{Sign} \times Mantissa \times Base^{Exponent-Bias}$$

Real Time Data Selection

The purpose of this modification to ASN.1 is to allow the description of situations where one possibility has to be selected among several alternatives at the time the physical data is being received. Such situations are typical in space related applications: for example the value of the first byte in the physical data could indicate the format of satellite tracking measurement samples -- Doppler/Ranging/Meteo --) or a layout word at the beginning of a housekeeping telemetry frame would indicate whether `commands acknowledgement` or `memory dumps` or `attitude parameters` are contained in that particular frame received on ground. Thus, the format of a piece of data typically is dependent upon a value of another piece of data (called the *discriminant*) and it should be possible to describe this aspect.

It has been felt that this ability is fundamental to existing space related data products and since at present ASN.1 does not have it, MADEL has been designed to support it: to this end, the `SELECT` type has been introduced. Thus with MADEL, the discriminant conveyed within the physical data can be processed in order to dictate the branch to be selected, without needing any encoding/decoding mechanism. Furthermore, the type of the discriminant itself must also be specified, for example, `INTEGER`, `REAL`, `IA5String`. The format of the `SELECT` statement in a MADEL description would be:

```
Packet ::= SELECT PacketType {          -- PacketType is the discriminant
                              "AX" : AuxiliaryPacket,
                              "HK" : HouseKeepingPacket,
```

```
                    "NS" : NormalSciencePacket,
                    "BS" : BurstSciencePacket
                  }
```

together with the following discriminant definition:

```
PacketType ::= IA5String(SIZE (2))
```

Implementation of the MADEL Interpreter

The task of interpreting a MADEL description together with an instance of the corresponding physical data is achieved (and demonstrated) by prototype software called the MADEL Interpreter. An overview of the MADEL Interpreter architecture is shown in Figure 2: this shows the main processes that are involved and the critical data structures.

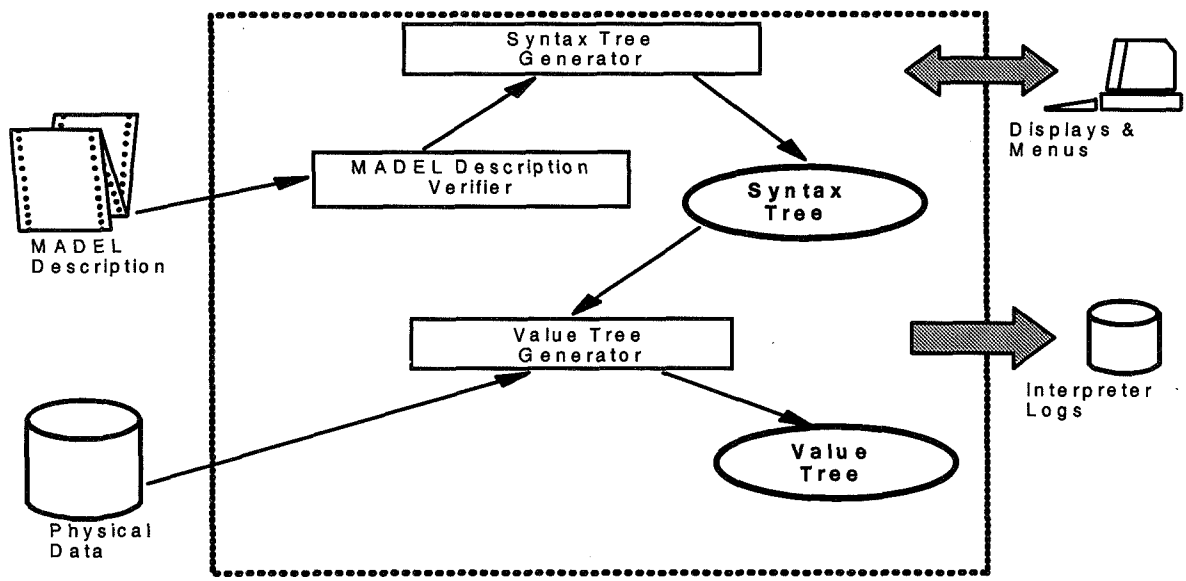


Figure 2: **MADEL Interpreter Software Architecture**

Running the MADEL Interpreter

• The user feeds into the MADEL Interpreter a MADEL description of his data and the physical data file. The MADEL description is verified for correctness by the MADEL Description Verifier module.

• The verified MADEL description is then parsed by the Syntax Tree Generator and a syntax tree of the data structure defined in the MADEL description is built internally. This internal tree represents the full syntax description that is defined including all possible choices or selections. Figure 3 shows a schematic of such a tree for a simple data structure: Element **A** is defined as a sequence of element **B** (an integer), followed by **C** (an octet) followed by **D** (a selection) which is dependent upon the value at reception time of the integer **B** (the discriminant). If **B**=1 then **D** will be **P** (an integer), if **B**=2 then **D** will be **Q** (a sequence) and if **B**=3 then **D** will be **R** (a real). Finally if **Q** is selected then this will be a sequence of **X** (an integer) followed by **Y** (a real).
• The Value Tree Generator then walks down the syntax tree reading the corresponding physical data driven by the syntax tree. It sees at the top of the syntax tree that the first element is a sequence, this corresponds to no actual physical data, so it creates the top node of the value tree with no data, just a flag indicating a sequence. It then goes on to read the elements of the sequence; firstly an integer, so bits are read from the physical data

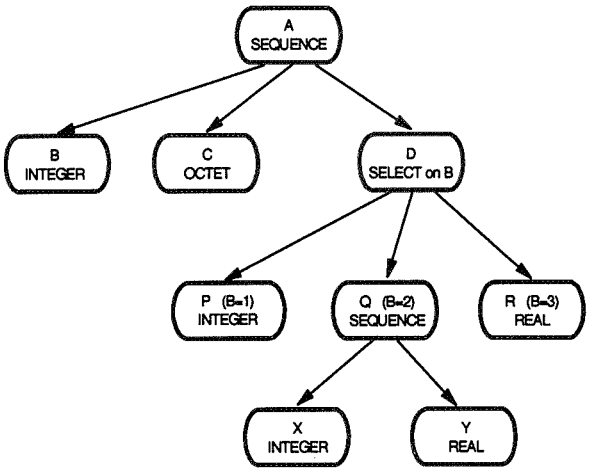


Figure 3: **Schematic of the Syntax Tree Generated**

and the Value Tree Generator adds a node to the value tree which it fills with the **local representation** of the actual value. Following this, the octet corresponding to **C** is read in the same manner. Again the select element **D** corresponds to no physical data, but the selection is dependent upon the value of **B**, so the value of **B** already stored in the value tree is accessed and the relevant branch of the syntax tree taken. Say **B** was 2, then branch **Q** is selected, so this branch of the syntax tree is followed and the corresponding branch of the value tree created. The resultant value tree is shown schematically in Figure 4. Eventually this tree will store the local representation of all selected physical data, and the user's application can proceed with stage 2 of the model (see Figure 1) or access the data using conventional methods. This link to the local representation of the values received in the physical data fulfils the task defined in stage 1 of the model.

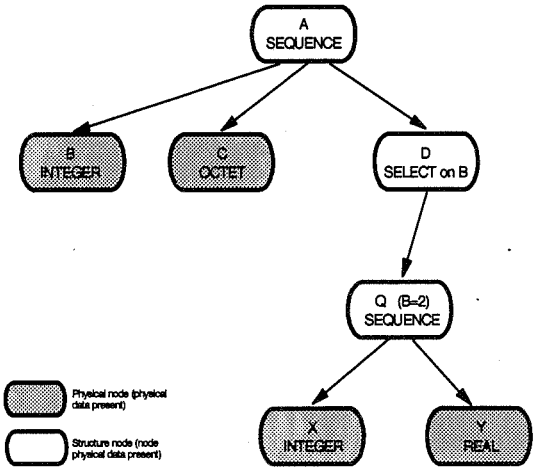


Figure 4: **Schematic of the Value Tree Generated**

Implementation and Support Tools

The MADEL Interpreter was developed on an IBM PC running MS-DOS, with Borland-C++ and MKS Lex & Yacc as major development tools. It is written in the C language.

Test data sets, along with their corresponding MADEL descriptions must be generated for test purposes. To this end support tools have been developed. A MADEL description is created with a normal text editor and fed into the MADEL Interpreter in ASCII text form. The physical data can be created interactively and a user interface mechanism has been integrated so that the user be provided with facilities to (i) select a MADEL description, then (ii) be guided in the production of associated physical test data.

Conclusions

• The study has shown the feasibility of the CCSDS concept and the analysis performed while developing MADEL has shown that, if some simple defaults and extensions are adopted (as have been defined), ASN.1 can be used satisfactorily as the basis for a suitable DDL. In particular it can fulfil the fundamental need for real time selection of data in the description of space related data. It should be noted that an important facet of the work was that MADEL, in terms of its syntax, has been kept very close to ASN.1 (as defined in the ISO 8824 standard).

• Prototype software (the MADEL Interpreter) was developed which demonstrated that typical space related data formats can be interchanged between heterogeneous computer environments and interpreted. In turn, this "proof of concept" development and testing helped to improve the overall data interchange model and to identify possible future additions to MADEL. User interface aspects were also studied.

• The MADEL Interpreter could become the basis of a set of tools for supporting data interchanges. Data product definitions or interface definitions written in MADEL could be generated and given to all users, thus avoiding the need for paper Interface Control Documents or paper format definitions. This would ensure that all users have a unique, correctly coded product description.

• In the longer term, stage 3 of the data interchange model should be looked at with the objective of defining standardised approaches and identifying common software services for the processing of advanced semantics.

EXAMPLES

Example 1: Interchanging Real Numbers from VAX to IBM

```
-- Hexadecimal dumps of VAX real numbers were generated on a VAX/VMS computer.
-- The corresponding MADEL description was created (this file) and then processed
-- by the MADEL Interpreter on an IBM PC to interpret the corresponding VAX data.

-- GFloatNumber structure on VAX
--    Word_1   {Sign:1 , Exp:11 , Mant:4}
--    Word_2    Word_3   Word_4 :   all {Mant:16}
-- FFloatNumber structure on VAX
--    Word_1   {Sign:1 , Exp:8 , Mant:7}
--    Word_2   {Mant:16}

-- Case 1
-- GFloatDump   Value =  -2.0000       Phys. Hex.    = C020 0000 0000 0000
-- FFloatDump   Value =  -2.0000       Phys. Hex.    = C100 0000

-- Case 2
-- GFloatDump   Value = -64071.5000    Phys. Hex.    = C10F 48F0 0000 0000
-- FFloatDump   Value = -64071.5000    Phys. Hex.    = C87A 4780

-- MADEL Definition starts here:

VaxVmsReals DEFINITIONS  ::=
BEGIN

List-of-VaxNumbers ::= SEQUENCE SIZE( 2 ) OF Numbers      -- 2 cases in this example

Numbers ::=  SEQUENCE {
                       GFloatNumber,
                       FFloatNumber
                      }

GFloatNumber ::= REAL {
                       BIAS (1025),          -- bias value
                       COMPLEMENT(0),        -- 0, 1 or 2's complement indicator
                       BASE(2),              -- base value
                       MANTISSA_MODE( 1),    -- algorithm for mantissa evaluation
                       MANTISSA(12,63),      -- mantissa bit positions
                       EXPONENT( 1,11)       -- exponent bit positions
                      }

FFloatNumber ::= REAL {             -- using ANSI/IEEE 754 defaults for REAL type
                       BIAS( 129)   -- except bias which is VAX specific
                      }

END
```

MADEL Interpreter Execution Report:

```
Symbol Name ...:  GFloatNumber
Size ..........:  64    Type Name ...:  REAL
Exponent_bits = (1..11),   Mantissa_bits = (12..63),
```

```
Complement = 0,   Base = 2,   Bias = 1025,   Mantissa_Mode = 1
PHYSICAL REPRESENTATION .: 11000000 00100000 00000000 00000000
                           00000000 00000000 00000000 00000000
 HOST REAL NUMBER = -2
```

```
Symbol Name ...:  FFloatNumber
Size ..........:  32    Type Name ...:  REAL
Exponent_bits = (1..8),   Mantissa_bits = (9..31),
Complement = 0,   Base = 2,   Bias = 129,   Mantissa_Mode = 1
PHYSICAL REPRESENTATION .: 11000001 00000000 00000000 00000000
 HOST REAL NUMBER = -2
```

```
Symbol Name ...:  GFloatNumber
Size ..........:  64    Type Name ...:  REAL
Exponent_bits = (1..11),   Mantissa_bits = (12..63),
Complement = 0,   Base = 2,   Bias = 1025,   Mantissa_Mode = 1
PHYSICAL REPRESENTATION .: 11000001 00001111 01001000 11110000
                           00000000 00000000 00000000 00000000
 HOST REAL NUMBER = -64071.5
```

```
Symbol Name ...:  FFloatNumber
Size ..........:  32    Type Name ...:  REAL
Exponent_bits = (1..8),   Mantissa_bits = (9..31),
Complement = 0,   Base = 2,   Bias = 129,   Mantissa_Mode = 1
PHYSICAL REPRESENTATION .: 11001000 01111010 01000111 10000000
 HOST REAL NUMBER = -64071.5
```

Example 2: Real Time Data Selection

```
-- This example is a special case of the data format discussed in Figure 3
-- and illustrates further the dynamics of data interpretation at the destination

-- MADEL Definition starts here:
Example-of-DataRealTimeSelection  DEFINITIONS  ::=
BEGIN

A ::= SEQUENCE { B, C, D }

B ::= INTEGER                -- specification of the discriminant
C ::= OCTET STRING
D ::= SELECT B {             -- B is discriminant for selecting
                 1 :  P,
                 2 :  Q,
                 3 :  R
                }
P ::= INTEGER
Q ::= SEQUENCE  { X, Y }
R ::= REAL
X ::= INTEGER    -- description of the lowest level elements in
Y ::= REAL       -- the data format A
END
```

MADEL Interpreter Execution Report:

```
Symbol Name ...:  B
Size ..........:  16    Type Name ...:  INTEGER
PHYSICAL REPRESENTATION .: 00000000 00000010
 INTEGER VALUE =  2
```

```
Symbol Name ...:  C
Size ..........:  1    Type Name ...:  OCTET
PHYSICAL REPRESENTATION .: 00101010
 OCTET STRING VALUE = '*'
```

```
Symbol Name ...:  D
Size ..........:  16    Type Name ...:  SELECT
```

```
PHYSICAL REPRESENTATION .:     see symbol  B
 SELECTED BRANCH =  2
```

```
Symbol Name ...:  X
Size ..........:  16     Type Name ...:  INTEGER
PHYSICAL REPRESENTATION .: 00000000 10000001
 INTEGER VALUE =  129
```

```
Symbol Name ...:  Y
Size ..........:  32     Type Name ...:  REAL
Exponent_bits = (1..8),    Mantissa_bits = (9..31),
Complement = 0,    Base = 2,    Bias = 128,    Mantissa_Mode = 1
PHYSICAL REPRESENTATION .: 01000000 01000000 00000000 00000000
 HOST REAL NUMBER = 1.5
```

REFERENCES

[1] "Information technology - Open Systems Interconnection - Specification of Abstract Syntax Notation (ASN.1)", ISO 8824, Second edition, ISO/IEC 8824:1990(E), International Organisation for Standardisation, December 1990.

[2] "Recommendation for Space Data System Standards: Parameter Value Language -- A Tutorial", CCSDS 641.0-G-1, Green Book, Consultative Committee for Space Data Systems, May 1992.

111183 N95-17598

354304 p.8

Cross Support Overview and Operations Concept for Future Space Missions

By William Stallings (NASA/GSFC)
and Jean-Francois Kaufeler (ESA/ESOC)

Supported by the CCSDS Panel 3 Core Group
CNES - Lionel Baize, Gerard Lapaian, Jean Yves Trebaol
CSA - Andrzej Kaminski (MPB)
DLR - Hubertus Wanke, John Dallat (CAM)
ESA/ESOC - Olivier Pujo, Klaus-Jürgen Schulz, Gerhard Theis, Hans Uhrig
NASA/GSFC - Fred Brosi (CTA), Norman Gundersen (CTA), Lionel Mitchell (CTA),
John Pietras (MITRE), Roland Weiss (CTA)
NASA/JPL - Edward Greenberg, Randy Heuser, Michael Stoloff

Abstract

Ground networks must respond to the requirements of future missions, which include smaller sizes, tighter budgets, increased numbers, and shorter development schedules. The Consultative Committee for Space Data Systems (CCSDS) is meeting these challenges by developing a general cross support concept, reference model, and service specifications for Space Link Extension services for space missions involving cross support among Space Agencies. This paper identifies and bounds the problem, describes the need to extend Space Link services, gives an overview of the operations concept, and introduces complimentary CCSDS work on standardizing Space Link Extension services.

Introduction

Future space missions will require the support of ground networks operated by multiple Space Agencies as well as support by multiple Agency organizations. Current missions are supported on a case by case basis with custom interfaces being developed each time. This is a time consuming and expensive process. CCSDS is developing recommendations for standards for interfaces and services in missions involving multiple Space Agencies. The objectives are to reduce cost and development time while increasing flexibility and efficient utilization of resources.

Future ground systems must replace custom interfaces with standard interfaces and services to be cost effective. Prior CCSDS recommendations have focused on standardizing the communication services between spacecraft and ground stations (i.e., the Space Link Subnet). The CCSDS Recommendations for Advanced Orbiting Systems (CCSDS, 1992a), Packet Telemetry (CCSDS, 1992b), and Telecommand (CCSDS, 1987a; CCSDS, 1992c; CCSDS, 1991; CCSDS, 1987b) document these Space Link services and protocols.

The proposed concept, documented in a CCSDS Report (CCSDS, 1994a) describes *Space Link Extension* (SLE) services that extend the Space Link services on the ground. Extension is accomplished over distance, in time, and by adding information. SLE services may be distributed across multiple ground facilities, such as ground stations, mission-related control centers, and data processing facilities. These facilities may be grouped to provide the services required by each mission. These SLE Services are applicable between Agencies

PAGE 1253 INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

as well as within Agencies with multiple ground networks.

Cross Support Operations Concept

Cross Support occurs when one Agency uses part of another Agency's data system resources to complement its own system in providing services.

Cross Support Environment

A space data system, for a particular mission, contains onboard spacecraft applications and ground applications. Ground applications interact with applications onboard the Spacecraft via application associations between them. The ground and onboard applications do not necessarily belong to the same Agency that is operating the spacecraft. The associations between ground and onboard applications are established and maintained using telecommunication and data transfer services built upon the Space Link communication services defined by CCSDS Recommendations.

CCSDS Recommendations are defined for Space Link services for the real-time transfer of data across the Space Link. However, space data systems generally require additional features in order to use the Space Link services to support mission application associations. These additional features, provided by SLE services, extend the application associations beyond the immediate endpoints of the space/ground link. The Space Link services are extended from onboard applications, attached to onboard local area networks, to ground applications, attached to terrestrial wide area networks. The SLE services provide the ability to hold data at one or more intermediate points between the peer applications. The Space Link and Space Link Extension domains are illustrated in Figure 1.

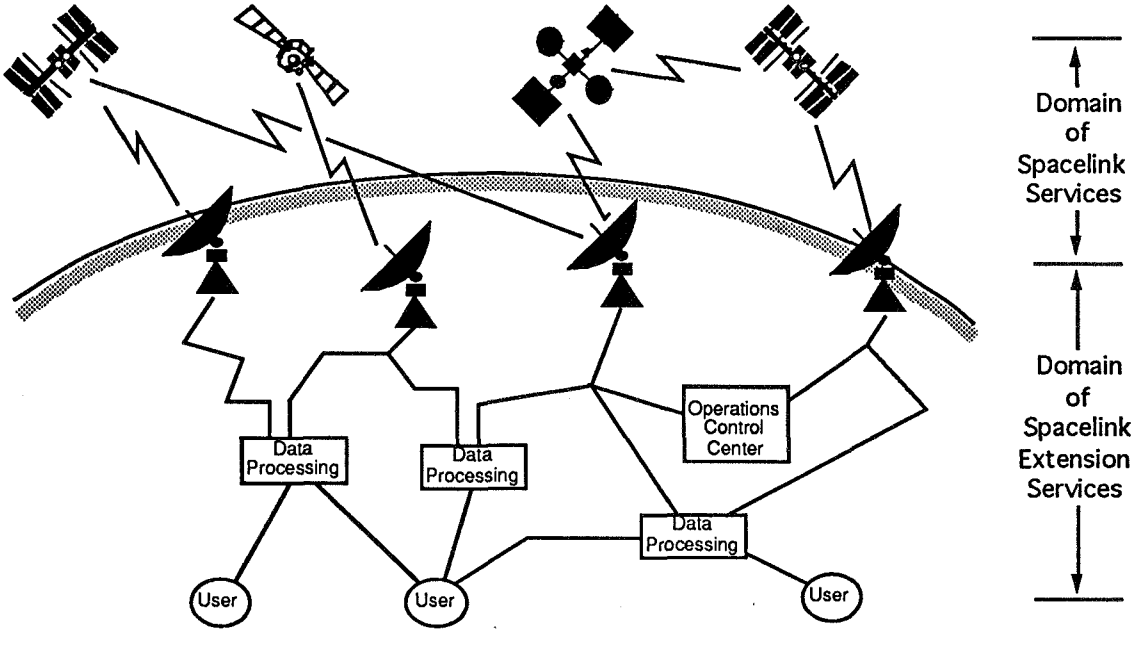


Figure 1 - Domains of Space Link and Space Link Extension Services

The ground-resident *Space Link Extension Component* (SLEC) and the onboard data system coordinate to provide SLE services. A particular mission may use all or a subset of the SLE services. In providing SLE Services, the SLEC performs: (1) RF modulation/demodulation at the ground termination of the space-ground link; (2) ground termination of the Space Link protocols used by the mission; (3) value-added annotation of the Space Link service data; (4) terrestrial networking among the ground elements that host the ground applications; and (5) interface to ground-side Space Link protocol processing and ground-side RF modulation/demodulation functions.

The SLEC has three types of interfaces with other components: interfaces over which mission data flow between the SLEC and the Spacecraft; interfaces over which space data flow between the SLEC and the ground applications; and the service management interface between the SLEC and Mission Management. Unlike the onboard data system service interfaces, the SLE service interfaces are intended to be standardized across all missions.

The SLE-Spacecraft interface operates over an RF medium and executes the Space Link protocols specified in the CCSDS Space Link Recommendations. The ground applications exchange SLE Protocol Data Units (SLE-PDUs) with the SLEC. The SLEC and Mission Management exchange service requests and service management reports over the service management interface. These service requests and management reports are used to: (1) configure/monitor the ground side of the RF link and Space Link protocol processing associated with the interface between the SLEC and the Spacecraft; (2) configure/monitor data handling within the SLEC; and (3) configure/monitor service delivery parameters for the interfaces between the SLEC and ground applications.

In addition, the Mission Management establishes an association with the Onboard Management component of the Spacecraft to configure and monitor the spacecraft side of the interface. This association uses the same set of communication services that other mission applications use. Figure 2 illustrates the associations and interfaces involved in providing the SLE Services.

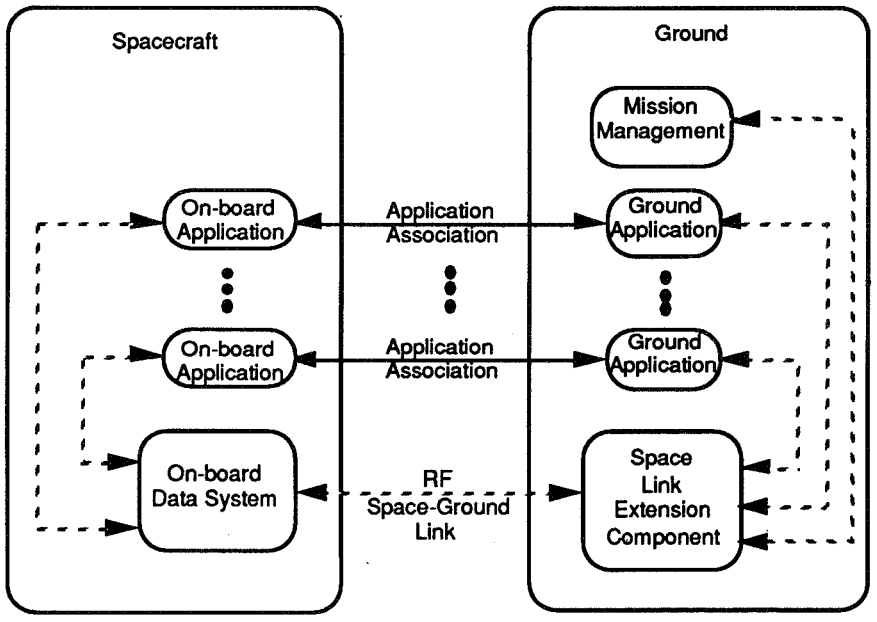


Figure 2 - SLE Associations and Interfaces

Cross Support Concepts

SLE services provide access to the ground termination of Space Link services from a remote ground-based system. They extend the various Space Link services as defined in CCSDS Space Link recommendations. This "extension" has three aspects: distance, information, and time. An SLE service may be completed at a location geographically separated from the place where the RF signal is received. Information is added to the Space Link Service Data Units (SL-SDUs) to compensate for the use of managed information over the Space Link or information about conditions at the time of receipt. Information may also be added to ensure the data will be useful at a later time. The added information is called "annotation."

The systems performing an SLE service may belong to different entities. These entities may include a different organization within the mission's own Space Agency or an organization from a different Space Agency. The supporting organizations may be of varying size or structure (e.g., Space Agency, space flight center, facility). The notion of cross support can be generalized to any situation in which multiple organizations are involved in supplying SLE Services.

As illustrated in Figure 3, the systems performing an SLE service are grouped into *Service Complexes* by the organizations that implement them. Each service complex has two components, a *Service Provision* component and a management component. The Service Provision component contains the processing functions implemented by that Service Complex. The management component manages the Service Provision component. The management of an SLE Service is distributed between Service Complexes and the Mission. The management component of a Service Complex is called *Complex Management.* The component of Mission Management responsible for management of SLE services is called *Utilization Management.* Service Management is accomplished through the management of the functions performed by the individual Service Complexes that provide the SLE services.

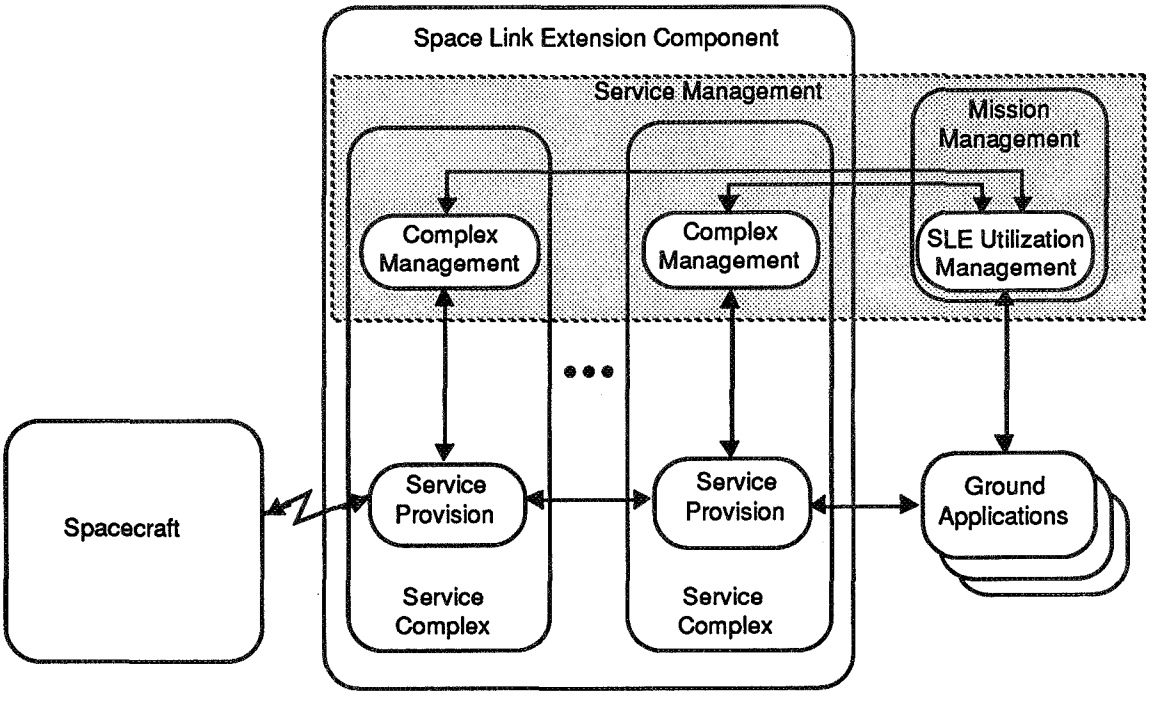


Figure 3 - SLEC Complex Interfaces

If a service is provided by multiple Service Complexes, Utilization Management coordinates with Complex Management in the multiple Service Complexes to provide the services required by a mission. Utilization Management must also coordinate and resolve conflict among multiple service users. Complex Management represents the functions performed within the Service Complex in a standard way, in terms of the SLE services provided by the complex (not in terms of the equipment used to provide those services), and without Service Complex-internal details. Complex Management provides the "firewall" that hides the complexity of the Service Complex.

Cross Support Services

Cross support may occur between ground applications and the SLEC. It may also occur between Service Complexes within the SLEC. The SLEC builds on the Space Link Services by standardizing the SLE and Service Management protocols and services. Such standardization allows a mission to interface with the SLEC, concatenating SLE services by interconnecting Service Complexes within the SLEC, no matter how or where the services are implemented. The SLE Services support both the Advanced Orbiting Systems (AOS) and Conventional Systems, Packet Telemetry (PT) and Telecommand (TC). The following list identifies the SLE services:

- Return All Frames (AOS and PT)
- Return Insert (AOS)
- Master Channel Frames (AOS and PT)
- Master Channel Frame Secondary Header (PT)
- Master Channel Operations Control Field (PT)
- Virtual Channel Frame Secondary Header (PT)
- Virtual Channel Operations Control Field (AOS and PT)
- Return Virtual Channel Access (AOS and PT)
- Return Bitstream (AOS)
- Return Space Packet (AOS and PT)
- Data Set Processing (AOS and PT)
- Return Internet (AOS)
- Forward Virtual Channel Access (AOS)
- Frame Data Routing (TC)
- Forward Bitstream (AOS)
- Forward Space Packet (AOS and TC)
- Forward Primary Header plus VCDU (AOS and TC)
- Telecommand Frame (TC)
- Forward Coded VCDU (AOS)
- CLTU (TC)
- Forward Internet (AOS)
- Forward Insert (AOS)

Cross Support Scenario

An example of cross support is illustrated in Figure 4. In this example, Agency A sends data from its spacecraft to multiple ground stations, which perform all data processing functions through the extraction of Frame Data. However, not all these ground stations belong to Agency A. The data is also transmitted to a ground station owned by Agency B which processes the data in two separate complexes. Essentially, the first complex performs the Space Link processing and delivers all frames to the second. Both Agencies deliver space Packets to Agency A's Data Processing Complex. While this does not affect the service interface, it may affect the management interfaces between the Agencies.

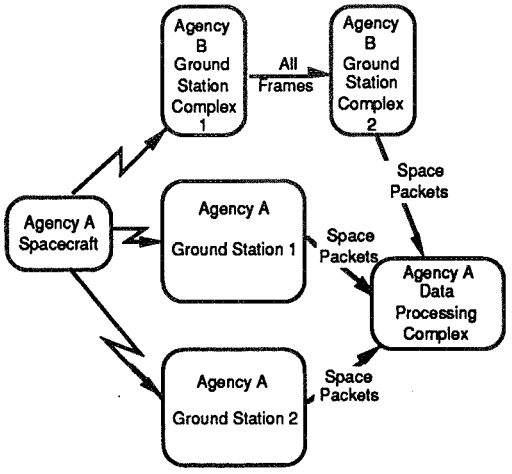


Figure 4 - Cross Support Example

Cross Support Lifecycle

Cross support of a mission involves a *Support Contract* between Agencies. A Support Contract is an agreement between a mission and one or more Agencies providing cross support. The Support Contract life cycle is divided into four phases: Agreement, Negotiation, Implementation, and Utilization.

The Agreement phase consists of the early interactions that set the stage for the technical definition of the cross support. The Agencies agree on objectives of Service Management interface and legal and financial responsibilities.

During the Negotiation phase, in the Support Contract is negotiated by the Agencies. It defines the services to be supported by the Service Complex over the lifetime of the Support Contract. The contract establishes the outer bounds of resources accessible by, and privileges extended to, the mission.

The Implementation phase is the time allowed for the Service Complexes to acquire, develop, and configure the resources necessary to satisfy the Support Contract. The Service Complexes perform testing to ensure conformance with CCSDS standards and compatibility with peer processes.

During the Utilization phase, a Service Complex provides one or more *Service Packages* to the mission. A Service Package consists of the service instances and channels in a single service complex that provide all or part of a service to a user. A Service Package duration corresponds to a Space Link session, or a "Pass."

Each Service Package has four phases: Preparation, Setup, Execution, and Debrief. Different Service Packages may be in different phases concurrently. For each Service Package the following actions occur:

- Preparation phase - Parameter values are selected for all parameters within bounds of service specified during the Negotiation phase including any schedule information applicable to a particular upcoming Execution phase.

- Setup phase - Initiation of a service, testing of service interfaces, and refinement of service parameters are performed by the Service Complex to ensure that the service selected during the Preparation phase can actually be provided during the upcoming Execution phase.

- Execution phase - Exchange of space data or the delivery of event, alarm, and status reports may take place between Service Complexes and between a Service Complex and a user.

- Debrief phase - Accounting and performance information about the Service Package Execution Phase is delivered to Service Management and/or the service user

Complementary CCSDS Work

A CCSDS Report, *Standard Terminology, Conventions, and Methodology* (TCM) (CCSDS, 1994c), provides terminology and conventions appropriate to the development of CCSDS Space Link Extension Services.

The TCM establishes a common vocabulary based on internationally standard terms and conventions for describing systems and their interactions, from conceptual level through the level at which specific technologies, protocols, and applications are applied to the development of CCSDS recommendations. It specifies the use of Abstract Service Definition

Conventions (ASDC) (ISO/IEC, 1992), a standard set of conventions that complement the better-known International Organization for Standardization (ISO) Open System Interconnection (OSI) service definition conventions. This common vocabulary and methodology can be used as a foundation for expressing concepts of operation and architectural specifications, leading to the definition of specific SLE Services and protocol specifications. All SLE Cross Support documents adhere to the TCM.

A Reference Model is also being developed by CCSDS for use in defining SLE services. The *SLE Reference Model* (CCSDS, 1994b) provides a common basis for the development of SLE Service recommendations. It provides the reference for maintaining consistency between all SLE Services. It provides descriptions as well as the provision of multiple SLE Services. The Reference Model also shows the relationships among the SLE services, the Space Link services that they extend, and the ground communication services on which they depend.

The Reference Model defines the common functionality, and provides the descriptive tools to specify service-specific functionality for a Service Complex. Examples of Service Complex functionality include: extensions to communication functions (e.g., annotation, addressing); data handling functions (e.g., data capture, post-pass retrieval); and management functions (e.g., Pass set-up, fault isolation).

CCSDS is currently working on the service specification for a single SLE Service. The Return All Frames service specification (CCSDS, 1994d) describes the most basic SLE service in the return (space-to-ground) direction. The Return All Frames service acquires, demodulates, frame-synchronizes, and decodes all CCSDS link layer frames (Packet Telemetry Transfer Frames or Virtual Channel Data Units) of a physical channel and delivers those frames to the users of the service. The service provides both on-line (i.e., near real time) and off-line (i.e., delayed or buffered) data transfer modes to accommodate the variety of access methods typical of actual space mission operations scenarios. The Return All Frames service is summarized in a companion SpaceOps '94 paper (Uhrig et al., 1994).

Future Work

CCSDS will publish Green Books for the Standard Terminology, Conventions, and Methodology and Cross Support Concept documents and Blue Books for the Reference Model and Return All Frames Service Specification documents. CCSDS Panel 3 also expects to develop service specification Blue Books for all Space Link Extension cross support services.

References

CCSDS. (1987a, January). *Recommendation for Space Data System Standards. Telecommand, Part 1 - Channel Service: Architectural Specification*. CCSDS 201.0-B-1, CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

_____. (1987b, January). *Recommendation for Space Data System Standards. Telecommand, Part 3 - Data Management Service: Architectural Specification*. CCSDS 203.0-B-1. CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

_____. (1991, October). *Recommendation for Space Data System Standards. Telecommand, Part 2.1 - Command Operation Procedures*. CCSDS 202.1-B-1. CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

_____. (1992a, November). *Recommendation for Space Data System Standards. Advanced Orbiting Systems, Network and Data Links: Architectural Specification.* CCSDS 701.0-B-2. CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

_____. (1992b, November). *Recommendation for Space Data System Standards. Packet Telemetry,* CCSDS 102.0-B-3, CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

_____. (1992c, November). *Recommendation for Space Data System Standards. Telecommand, Part 2 - Data Routing Service: Architectural Specification.* CCSDS 202.0-B-2. CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

_____. (1994a, November). *Report Concerning Space Data System Standards. Cross Support Concept, Part 1: Space Link Extension Services.* CCSDS 910.3-G-1. CCSDS Panel 3 plenary meeting, Goddard Space Flight Center, Greenbelt, MD.

_____. (1994b, November). *Recommendation for Space Data System Standards. Space Link Extension Reference Model.* CCSDS 910.4-W-0.1. CCSDS Panel 3 plenary meeting, Goddard Space Flight Center, Greenbelt, MD.

_____. (1994c, November). *Report Concerning Space Data System Standards. Standard Terminology, Conventions, and Methodology (TCM).* CCSDS 910.2-G-1. CCSDS Panel 3 plenary meeting, Goddard Space Flight Center, Greenbelt, MD.

_____. (1994d, November). *Recommendation for Space Data System Standards. Return All Frames Space Link Extension Service Specification* CCSDS CCSDS 911.1-W-0.1. CCSDS Panel 3 plenary meeting, Goddard Space Flight Center, Greenbelt, MD.

ISO/IEC. (1992, March). *Information technology - Text communication - Message-Oriented Text Interchange Systems (MOTIS) - Part 3: Abstract Service Definition Conventions, Technical Corrigendum 1.* ISO/IEC 10021-3.

Uhrig, H., Pietras, J., & Stoloff, M. (1994, July). The CCSDS Return All Frames Space Link Extension Service. *Proceedings of the Third International Symposium on Space Mission Operations and Ground Data Systems.* Goddard Space Flight Center, Greenbelt, MD.

111184 N95-17599

384309

P-7

THE CCSDS RETURN ALL FRAMES SPACE LINK EXTENSION SERVICE

Dr. Hans Uhrig
European Space Operations Centre
Darmstadt, Germany

John Pietras
The MITRE Corporation
Greenbelt, Maryland, USA

Michael Stoloff
Jet Propulsion Laboratory
Pasadena, California, USA

ABSTRACT

Existing Consultative Committee for Space Data Systems (CCSDS) Recommendations for Telemetry Channel Coding, Packet Telemetry, Advanced Orbiting Systems, and Telecommand have facilitated cross-support between Agencies by standardizing the link between spacecraft and ground terminal. CCSDS is currently defining a set of Space Link Extension (SLE) services that will enable remote science and mission operations facilities to access the ground termination of the Space Link services in a standard manner. The first SLE service to be defined is the Return All Frames (RAF) service. The RAF service delivers all CCSDS link-layer frames received on a single space link physical channel. The service provides both on-line and off-line data transfer modes to accommodate the variety of access methods typical of space mission operations. This paper describes the RAF service as of the Summer of 1994. It characterizes the behavior of the service as seen across the interface between the user and the service and gives an overview of the interactions involved in setting up and operating the service in a cross-support environment.

INTRODUCTION

Widespread acceptance of existing CCSDS Recommendations on Telemetry Channel Coding (CCSDS, 1992a), Packet Telemetry (CCSDS, 1992c), Advanced Orbiting Systems (CCSDS, 1992d), and Telecommand (CCSDS, 1987a; CCSDS, 1992b; CCSDS, 1991; CCSDS, 1987b) has facilitated cross-support between Agencies by standardizing the link between a spacecraft and a ground terminal. However, significant impediments to cross-support remain because the scope of those Recommendations does not include the link between the ground terminal and other elements of the ground data system. CCSDS is addressing that lack through the definition of a set of Space Link Extension (SLE) services that will enable remote science and mission operations facilities to access the ground termination of the Space Link services in a standard way.

The most basic SLE service in the return (space-to-ground) direction is the Return All Frames (RAF) service. Provision of the RAF service involves the acquisition, demodulation, frame synchronization, and error detection/correction of all CCSDS link-layer frames of a physical channel, and the delivery of those frames across terrestrial networks to the users of the service. *Frame* is the term used in this paper as a common name for the various CCSDS data link protocol data units.[1] The users of the RAF service split the all-frames data stream into

[1]The Version 1 return link frame is formally known as the Packet Telemetry Transfer Frame, and the Version 2 frame is formally known as the (Coded) Virtual Channel Data Unit.

subsets based on master channels and virtual channels, and extract various data products from those channels, as defined in (CCSDS, 1992c; CCSDS, 1992d) and forthcoming Recommendations for other return SLE services.

The RAF service provides both *on-line* and *off-line* data transfer modes to accommodate the variety of access methods typical of space mission operations. An online service is one that delivers its service data to the user at (nearly) the same time that the data are received from the space link. An offline service is one in which the service data are delivered at some time after that at which the data crossed the space link.

CCSDS is in the process of defining the RAF service in detail. The RAF service is the first of the SLE services for which a draft Recommendation is being developed. CCSDS expects to submit the resulting Recommendation for review and approval by its member space agencies in 1995. Recommendations for the other SLE services will follow.

This paper describes the CCSDS Return All Frames SLE service as it is defined as of the Summer of 1994. First, the RAF service environment is presented. The environment identifies the various participants in the provision of the RAF service. Next, the behavior of the RAF service is described, in terms of the interactions between the user and provider of the service. Finally, the paper briefly introduces the formal techniques being used to describe the RAF and other SLE services.

RAF SERVICE ENVIRONMENT

As defined in the SLE cross-support concept Green Book (CCSDS, 1994a) and presented in the companion SpaceOps '94 paper (Stallings et al., 1994), all SLE services are defined within the context of a *space mission* and an *SLE Component* that supports that mission. Figure 1 illustrates the Return All Frames service environment, which is a specific case of the general SLE service environment. In the terminology of the SLE reference model (CCSDS, 1994b), the various entities illustrated in Figure 1 are different types of *SLE objects*. The RAF Service User, the Mission Spacecraft, and the mission's Space Link Extension Utilization Management (SLE-UM - that part of mission management responsible for managing the acquisition and use of SLE services on behalf of the mission) are objects of the space mission. The SLE Component comprises those functions and systems that provide standard SLE services to the mission.

In the general case, the SLE Component may consist of multiple *SLE complexes*, where an SLE complex is a collection of SLE service capabilities operating in an integrated management domain (as seen by the SLE-UM). However, the RAF service, being the most basic SLE service, will be provided by a single SLE complex, referred to in this paper as the RAF Service Provider. Although the SLE service concept permits an SLE complex to be geographically-distributed, it is most convenient to think of the RAF Service Provider as being completely located at a ground station.

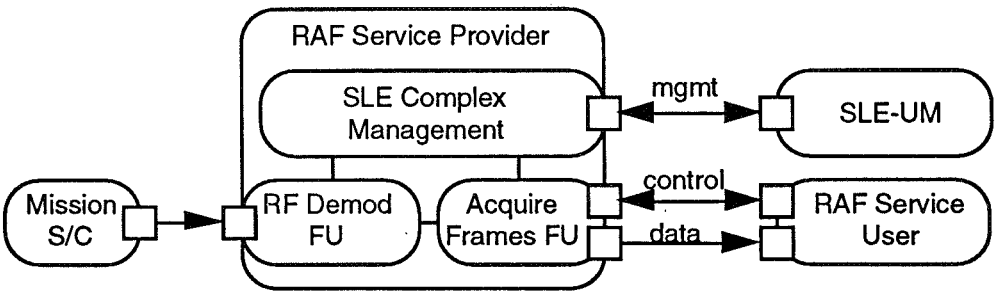


Figure 1. RAF Service Environment

The RAF Service Provider has an RF interface with the mission spacecraft, service control and data interfaces with the service user, and a management interface with the SLE-UM. The functions of the RAF Service Provider are formally modeled as two *functional unit* (FU) objects: the RF Demodulation (RFD) FU and the Acquire Frames FU. The RFD FU comprises the functions of acquiring the return RF signal from the mission spacecraft, and demodulating the signal to recover a stream of digital symbols. The Acquire Frames FU processes the symbol stream to synchronize upon and capture CCSDS frames, performs error detection and correction, and transfers the frames to the RAF Service User(s).[2] Frame synchronization and error detection and correction are performed in accordance with provisions of the Telemetry Channel Coding Blue Book (CCSDS, 1992a). The *SLE complex management* function of the RAF Service Provider coordinates the operation of the RFD FU and the Acquire Frames FU.

BEHAVIOR OF THE RAF SERVICE

The CCSDS RAF service delivers all frames from a single space link channel, including fill frames (which by definition have no data content and exist only to maintain synchronization on the link). A *space link channel* is a physical channel carrying a synchronous stream of frames, separated by *attached sync markers.*

The wide variety of mission needs and Agency capabilities requires flexibility in the delivery of the RAF service. This flexibility manifests itself through different requirements/capabilities in the areas of:

- Received data quality: what constitutes data that are usable by/acceptable to the user of the service
- Data delivery: the combination of reliability, timeliness, and completeness that best fits the user's needs
- Service session initiation: whether the service session (i.e., the connection between the service user and service provider for the purpose of transferring service data) is initiated by the provider or the user
- Service status information: how much information about the progress of the service is needed/wanted by the user, and the mechanism for delivering that information

The following sections describes the options available in these areas.

Received Data Quality Options

This area contains the options for service delivery that are affected by the quality of the frames acquired by the RAF Service Provider. These options are offered to meet the varying needs of the user community, and to match their ability to deal with errored data. Because this is the all *frames* service, each of these options deals with entities that have been frame synchronized and thus identified as frames.[3] The RAF service supports two options for received data quality: Correct Frames Only, and Correct and Errored Frames.

[2]Although only one RAF Service User is illustrated in Figure 1, there may be multiple users of RAF service receiving the same stream of data.

[3]In particular, the RAF Service Recommendation does not provide for the delivery of data symbols or bits which cannot be frame synchronized. It is recognized that, in certain rare cases, anomalies in flight or ground systems may lead to conditions under which the RAF service can not deliver any data at all because the acquired data stream does not contain properly-formed frames. Implementors of the RAF service are well-advised to provide an 'escape mechanism' for the capture and possible delivery of such anomalous data to support troubleshooting and/or extraordinary data recovery methods. However, the handling of such anomalous situations is outside the scope of the RAF Service Recommendation.

The Correct Frames Only option causes only frames that have no detected errors to be delivered. This option is available only when it is known beforehand (either due to long-term service agreements or by schedule) that the frames on the channel are either all Reed-Solomon (R-S) -encoded or all cyclic redundancy code (CRC) -protected frames. R-S-encoded frames that successfully R-S decode are transferred, without the R-S check symbols, to the user. R-S-encoded frames that do not R-S decode are discarded. CRC-protected frames that show no CRC errors are transferred to the user as complete frames. CRC-protected frames that register CRC errors are discarded.

The Correct and Errored Frames option causes all frames to be delivered, regardless of received quality. This option exists for:

- Space link channels whose frames carry data which contains sufficient "internal coding" that the user is able to reconstruct (through mission-unique methods) the data content, or
- Space link channels that carry a mix of R-S- and CRC-encoded frames.

The Correct and Errored Frames option might also be useful in helping missions to identify flight system and ground system problems.

For space link channels that are known beforehand to carry only R-S encoded frames, frames that successfully R-S decode are transferred (without the R-S check symbols) to the user with the indication **R-S-good**, and frames that do not successfully R-S decode are transferred as complete, pre-decoded frames with the indication **R-S-bad**.

For space link channels that are known beforehand to carry only CRC encoded frames, frames that have no CRC errors are transferred to the user with the indication **CRC-good**, and frames that have CRC errors are transferred with the indication **CRC-bad**.

For space link channels that may be carrying mixed R-S and CRC encoded frames, frames that are successfully R-S decoded are transferred (without the R-S check symbols) to the user with the indication **R-S-good**. Frames that are not successfully R-S decoded are checked for CRC errors, using the as-received (pre-R-S-decoded) complete frames. Frames that have no CRC errors are transferred to the user with the indication **CRC-good**, and frames that have CRC errors are transferred with the indication **CRC-bad**.

CCSDS recommendations for processing RAF service into higher-strata services (such as SLE packet services) are predicated on the use of the Correct Frames Only option. Since processing of errored frames requires mission-unique methods, the ability to process such frames is by definition not a part of the standard SLE service suite.

Data Delivery Options

The data delivery options reflect different levels of reliability, completeness, and timeliness. When complete, the RAF service specification will define online and offline data delivery options. As of this writing, only two online options, Complete delivery and Timely delivery, have been defined.[4]

An RAF service instance with the Complete delivery option delivers the frames in the sequence received, with no ground-induced errors, with no frames omitted, and with possible large delays. The RAF Service Provider buffers the data to compensate for data rate mismatch and/or retransmissions. The user can specify a maximum delay, T_{max}. If T_{max} is exceeded, the service informs the user that T_{max} has been

[4]CCSDS has developed recommendations for the use of isochronous virtual channels (CCSDS, 1992d). As of this writing, CCSDS is determining if these recommendations result in a requirement for an isochronous delivery option for the RAF service. If so, it will be added to the online delivery options.

exceeded. When the delay drops below T_{max}, the service informs the user that the delay has been recovered.

An RAF service instance with the Timely delivery option delivers the frames in the sequence received, with no ground-induced errors, possibly with frames omitted, and with an upper bound on maximum delay, T_{max}. Due to the nature of the RAF service, the data rate of delivery of the RAF service is normally a steady-state rate equal to the rate of the underlying space link channel plus some SLE overhead increment. As long as the rate of the RAF service can be kept at or below this steady-state rate, frames will be forwarded without buffering in the RAF Service Provider. However, if frames begin to be buffered in the RAF Service Provider (for example, because of excessive retransmissions to the service user), their delivery latency will grow. If this latency exceeds the maximum delay parameter T_{max}, the RAF service drops an appropriate number of frames to drop below T_{max} delay and informs the user that frames have been dropped.

Service Session Initiation Options

Service session initiation options describe the different options by which user and provider connect to support the provision of RAF service. When complete, the RAF service specification will define options for initiating online and offline service sessions. As of this writing, only the online options have been defined, which are Online Provider-Initiated and Online User-Initiated.

The Online Provider-Initiated option allows the RAF Service Provider to set up the connection to the user, based on the schedule for the activation of the space link. The preconditions for providing an instance of RAF service using this option are: (1) that the RAF Service Provider has in its configuration data base the identification, addressing information, startup data quality options, and startup data delivery options for the user(s) of the RAF service, (2) that the RAF Service Provider has been scheduled to receive the space link channel associated with the RAF service instance, and (3) that the RAF Service Provider has been scheduled to provide an RAF provider-initiated service instance to the particular user(s).

As controlled by the scheduled start time of the service instance, the RAF Service Provider connects to the service user(s) and verifies the user's ability to receive the service. Once the service session has been initiated, either user or provider can terminate the session, but the nominal responsibility for closing the session lies with the service provider, which it does after the space link channel has been deactivated and all received data has been delivered to the user.

The Online User-Initiated option allows the RAF Service User to set up the connection to the RAF Service Provider within a time window previously scheduled with respect to the scheduled activation of the space link. The preconditions for providing an instance of RAF service using this option are (1) that the RAF Service Provider has in its configuration database the identity of users permitted to receive the RAF service, (2) that the RAF Service Provider has been scheduled to receive the space link channel associated with the RAF service instance, and (3) that the RAF service provider has been scheduled to provide an RAF user-initiated service instance to the user. The user-initiated service instance can be scheduled to begin at any time after some specified time prior to the start of the space link session (the start time of the schedule window is determined on an agency/mission basis). The end time of the scheduled service instance can be scheduled at any time from the scheduled start of the service instance up until some specified time after the scheduled end of the space link session.

To initiate RAF service using the Online User-Initiated option, the service user connects to the Service Provider, in the process identifying and authenticating itself as a legitimate user of the RAF service associated with a particular space link channel. Service session requests associated with a specific space link session are valid

only during the scheduled service instance. If a service user attempts to initiate the service session before the start time of the service instance, the service session initiation attempt will be denied by the RAF Service Provider. A service user may initiate a service session at any time during the scheduled service instance. This includes service sessions that are initiated after the start of the associated space link session. Service sessions may be suspended and resumed during the scheduled service instance. Figure 2 illustrates the relationships among the space link session, scheduled service instance, a possible service session, and the resulting period of data transfer for the Online User-Initiated option. According to the figure, the service session is initiated after the start of the scheduled service instance, but before the actual start of the space link session. In this example, the service instance is scheduled to go some time beyond the end of the space link session to allow for a long buffer drawdown when a Complete delivery option is selected. The actual service session extends beyond the end of the space link session, but terminates before the scheduled end of the service instance. The resulting period of data transfer begins at the start of the space link session and ends at the end of the service session.

Once the user-initiated service session has been initiated, either user or provider can terminate the session, but the nominal responsibility for closing the user-initiated session lies with the service user. However, if the service session extends until the end of the scheduled service instance, the service session is terminated by the RAF Service Provider.

Service Instance Status Information. Service instance status information is provided to the SLE-UM and the user of the service during the execution of the service instance. Service instance status information is conveyed to the RAF service user for the purpose of providing information necessary for the proper interpretation and processing of the RAF service data units. Service instance status information is conveyed to the SLE-UM for the purpose of correlating the performance of the RAF service with the performance of the underlying RF link and the performance of multiple higher-strata services that are derived from the data contained in the frames delivered by the RAF service.

When complete, the RAF service Recommendation will define status information associated with both online and offline service instances. As of this writing, only the online service instance status information has been addressed.

Service instance status information falls into three categories:

- Annotation data, which are appended to the frames themselves

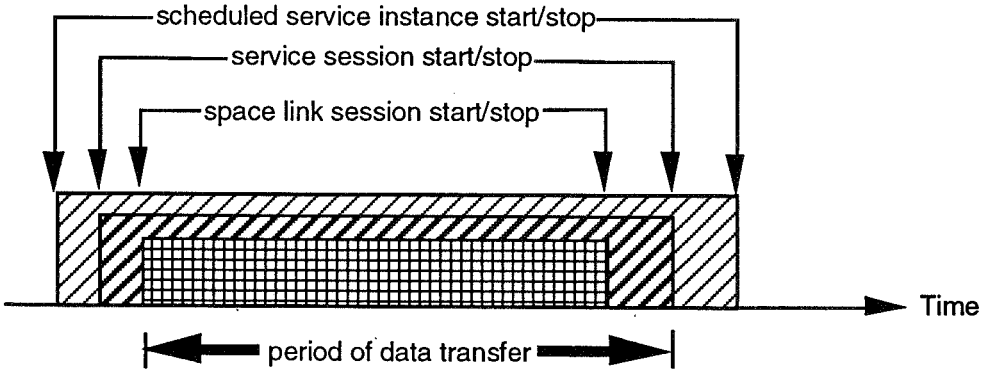


Figure 2 Relationships among Space Link Session, Scheduled Service Instance, Service Session, and Period of Data Transfer for the Online User-Initiated Mode

- Frame-sequence-dependent status data which conveys information about events of interest occurring between specific frames
- Frame-sequence-independent status data which conveys information about trends that transcend individual frames

Annotation and frame-sequence-dependent data are conveyed to the service user so that the user may correctly interpret and process the sequence of frames received. Frame-sequence-independent data are of interest to both the user of the service and to the SLE-UM: to the user, they serve as a means of accounting for the service received; to the SLE-UM, they are a source of information that can be correlated with status information about related services and RF links.

The core annotation data to be appended to every frame consists of:

- The ground receipt time of the frame
- The sequence quality of the frame (i.e., the indication of whether the frame is the direct successor of the previous frame on the space link channel)
- The quality of the frame (e.g., R-S bad, CRC good)

In addition to the above core annotation data, CCSDS is investigating methods to allow service providers to flexibly add annotation data that may be required on a mission or Agency basis.

The frame-sequence-dependent information that has been thus far identified consists of:

- Loss-of-synchronization notifications that inform the user in a timely fashion that frames are missing because of loss of synchronization (and not, for instance, because of failure of ground processing or communications)
- Data delivery threshold notifications:
  - When the Complete data delivery option is in effect, the service informs the user when a user-specified delay threshold has been exceeded, and again (if and) when the delay has been recovered.
  - When the Timely data delivery option is in effect, the service informs the user when a user-specified data latency threshold has been exceeded, and the number of frames that have been discarded in order to ensure the delivery of "fresh" data.

As of this writing, frame-sequence-independent status information is still being defined in the categories of periodic reports, event notifications, post-pass debriefing reports, and journals.

FORMAL SPECIFICATION OF THE RAF SERVICE

The SLE services, including the RAF service, are formally defined within a framework based on the International Organization for Standardization's (ISO) Abstract Service Definition Conventions (ASDC) (ISO/IEC, 1992). ASDC provides a conceptual model for constructing systems of *abstract-objects* which interact with each other via *abstract-ports*. The interactions are defined in terms of *abstract-operations*, and *abstract-services* are defined in terms of groupings of *abstract-operations* over one or more *abstract-ports*. ASDC provides a rich set of concepts and conventions for defining the various roles that the components of the SLE architecture may play, such as *user/provider*, *initiator/responder*, *invoker/performer*, and *consumer/supplier*. ASDC also provides a formal specification methodology using Abstract Syntax Notation #1 (ASN.1) *macros*, which serve as templates for the definition of the various elements of the model.

CCSDS has adapted the ASDC to the SLE environment, defining special subtypes of abstract-objects such as SLE complexes and functional units. CCSDS has also adapted the ASDC ASN.1 macros to include parameters peculiar to the SLE environment. The SLE reference model (CCSDS, 1994b) documents the SLE adaptation of ASDC.

Among other things, the reference model defines the set of ASN.1 macros that must be populated for each of the SLE service specifications. Thus, the CCSDS RAF Recommendation will contain ASN.1 macro-based specifications that complement the "plain English" definition of the RAF service.

SUMMARY

CCSDS is currently defining the Return All Frames service, one of a family of Space Link Extension services that will enable remote science and mission operations facilities to access the ground termination of the CCSDS Space Link services in a standard way. The RAF service provides all CCSDS frames received on a single space link channel. Provisions in the current draft of the RAF service specification include different service options to allow users to tailor the service to individual processing capabilities and operational philosophies. Several forms of service status information are provided to report on the status of individual frames, time-critical events, and long-term service trends. Current plans are for the CCSDS Return All Frames Service Specification Recommendation to be submitted for CCSDS member Agency approval in 1995.

REFERENCES

CCSDS. (1987a, January). *Recommendation for Space Data System Standards. Telecommand, Part 1 - Channel Service: Architectural Specification.* CCSDS 201.0-B-1, CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

____. (1987b, January). *Recommendation for Space Data System Standards. Telecommand, Part 3 - Data Management Service: Architectural Specification.* CCSDS 203.0-B-1. CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

____. (1991, October). *Recommendation for Space Data System Standards. Telecommand, Part 2.1 - Command Operation Procedures.* CCSDS 202.1-B-1. CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

____. (1992a, May). *Recommendation for Space Data System Standards. Telemetry Channel Coding.* CCSDS 101.0-B-3. CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

____. (1992b, November). *Recommendation for Space Data System Standards. Telecommand, Part 2 - Data Routing Service: Architectural Specification.* CCSDS 202.0-B-2. CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

____. (1992c, November). *Recommendation for Space Data System Standards. Packet Telemetry*, CCSDS 102.0-B-3, CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

____. (1992d, November). *Recommendation for Space Data System Standards. Advanced Orbiting Systems, Network and Data Links: Architectural Specification.* CCSDS 701.0-B-2. CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC.

____. (1994a, November). *Report Concerning Space Data System Standards. Cross Support Concept, Part 1: Space Link Extension Services.* CCSDS xxx-W-1. CCSDS Panel 3 plenary meeting, Goddard Space Flight Center, Greenbelt, MD.

____. (1994b, November). *Recommendation for Space Data System Standards. Space Link Extension Reference Model.* CCSDS xxx-W-1. CCSDS Panel 3 plenary meeting, Goddard Space Flight Center, Greenbelt, MD.

ISO/IEC. (1992, March). *Information technology - Text communication - Message-Oriented Text Interchange*

Systems (MOTIS) - Part 3: Abstract Service Definition Conventions, Technical Corrigendum 1. ISO/IEC 10021-3.

Stallings, W. & Kaufeler, J.-F. (1994, November). Cross Support Overview and Operations Concept for Future Space Missions. *Proceedings of the Third International Symposium on Space Mission Operations and Ground Data Systems*. Goddard Space Flight Center, Greenbelt, MD.

ACKNOWLEDGMENT

The draft RAF service Recommendation, upon which this paper is based, is being developed by the CCSDS Panel 3 Core Group. The authors wish to thank the members of the Core Group for their collaboration in the development of the SLE service concept and SLE service reference model, and their contributions to definition of the RAF service itself.

111185 N95-17600

354313 P. 6

PROPOSAL FOR IMPLEMENTATION OF CCSDS STANDARDS FOR USE WITH SPACECRAFT ENGINEERING/HOUSEKEEPING DATA

by Dave Welch
Senior Operations Engineer
AlliedSignal Technical Services Corporation
Flight Operations Department
Goddard Corporate Park
7515 Mission Drive
Lanham, MD 20706

(301) 805-3638
Fax: (301) 805-3130
E-mail: WelchD@ess-mail.atsc.allied.com

Abstract

Many of today's low earth orbiting spacecraft are using the Consultative Committee for Space Data Systems (CCSDS) protocol for better optimization of down link RF bandwidth and onboard storage space. However, most of the associated housekeeping data has continued to be generated and down linked in a synchronous, Time Division Multiplexed (TDM) fashion. There are many economies that the CCSDS protocol will allow to better utilize the available bandwidth and storage space in order to optimize the housekeeping data for use in operational trending and analysis work. By only outputting what is currently important or of interest, finer resolution of critical items can be obtained. This can be accomplished by better utilizing the normally allocated housekeeping data down link and storage areas rather than taking space reserved for science.

Background

This proposal began as a study to optimize the archival of spacecraft housekeeping data from the SAMPEX Small Explorer mission for use in long term data analysis and performance trending needs. As the study progressed, it became apparent that many of these optimization techniques could be put into the spacecraft itself by taking advantage of new advances in flight certified microprocessors and the options provided by the CCSDS protocol. Future missions could be programmed to detect most of the problems that the ground data systems currently look for and provide for higher resolution data to help in troubleshooting when a problem arises, filtering out unnecessary data when the spacecraft health is nominal.

When health and safety data is processed and analyzed, some data that is stored onboard in the recorder is filtered out on the ground and discarded. As long as parameters remain constant and configurations don't change, this information is redundant and unnecessary. Other data is output synchronously at to slow a rate to be of any use for anomaly analysis. This data may give indications of a problem, but not enough information to know exactly what is going on, or it may mask a problem for weeks or months, even years due too periodic sampling of the data that may be asynchronous to brief anomalous events.

It should be noted that attitude determination was not addressed in this study even though attitude data is usually considered a subset of the housekeeping data. Attitude data

PRECEDING PAGE BLANK NOT FILMED

PAGE 1270 INTENTIONALLY BLANK

packetization algorithms should be specified so as to meet science data processing requirements rather than performance analysis requirements that are usually less stringent.

Types of housekeeping data

The housekeeping data for SAMPEX fell into one of six different general categories: discrete counters, digital status data, analog data, flight software memory dumps, flight software memory dwell data and science quicklook packets. Time was not included in this study as a separate category as it is a parameter in every CCSDS packet header and therefore usually is not a part of the application data field. Obviously, time must be transmitted in such a fashion as to know when each telemetered data value was sampled.

The first category, discrete counters, is the primary means to monitor and diagnose the performance of flight software and/or the command and data handling unit. This data falls into two general subcategories. These are counters that infrequently increment and those which constantly increment. The counters that infrequently increment include command execution counters, command execution error counters and miscellaneous error counters. These types of counters are of interest only when they change value. The counters that constantly increment include time, task execution counters, and data storage accounting statistics. Some of these counters are always of interest, some are only of interest during flight software diagnostic testing, and some are only of interest during real-time.

The second category, digital status data, consists of configuration data (items that can be modified by command), error flags, environmental flags (generally indicate some orbital characteristic such as day or night delimiters) and informational data. This data is generally supplementary data that helps to determine when something happened and, like the infrequently incrementing counters mentioned above, are of interest only when they change. Examples of this type of data include spacecraft event messages, calculated onboard table checksums, flight software load and dump information and error handler takeover reasons.

The third category, analog telemetry, is probably the most important data for monitoring the health and safety of the spacecraft. What is key here is getting the right amount of data to detect problems or degradation without monopolizing the onboard data storage space or the down link bandwidth.

The next two categories, flight software memory dumps and flight software memory dwell data, are generally used for flight software maintenance purposes and would probably only be output on receipt of a spacecraft command. Handling of this data is an entire subject in itself and is not specifically referenced in this paper.

The last category, quicklook data, is generally handled by onboard microcomputers and, for SAMPEX, is only output on receipt of a command. It was only included in the SAMPEX study since it is the only source of instrument housekeeping data available in the control center. These data packets consist of a one orbit sample of various instrument rate counters and housekeeping status.

Data Processing Functions for Data Analysis and Performance Trending

The data processing functions done for data analysis and performance trending are very similar to the data processing steps for science data analysis. The first step involves a quality and accounting assessment to ensure that an adequate amount of data is recovered for data analysis and performance trending. The raw data is generally archived to provide a

backup in the event a data processing error is discovered in the future. Optional data merging may be done to combine real-time and playback data or to replace bad quality data with a better, retransmitted value. Finally, the data can be sorted by function or subsystem.

The next step generally involves ingesting the data values and affixed time-tags into a database for later access by analysis tools. This step includes processing the data and monitoring for high and low limit violations, verifying configuration and discrete state checks and optionally performing engineering unit conversions (if the storage database does not provide this function). At this point the data may also be processed to provide maximum/mean/minimum values of analog values for long term performance trending. This data may be processed for single orbits, daily or some other periodic unit of time.

After the data has been processed and stored in an on-line database, routine data analysis can be performed. This routine analysis function can generally be automated and may include creation of x/y plots for the thermal, communications or power subsystem as well as special processing for power budget monitoring and analysis or for attitude determination and control system verification.

Finally, some sort of orbit propagation may be done to provide a definitive history of actual spacecraft position over time. This data can be used both in subsequent anomaly investigation or for long term performance trending and is generally needed to isolate spacecraft problems that may be due to an environmental factor. In most circumstances, orbit accuracy requirements for science data processing are tighter than that required for performance analysis and therefore a commercial off the shelf orbit propagator, or ephemeri data provided for science data analysis, is sufficient. This data must be stored, or made available, to any plotting packages that would have access to the on-line spacecraft database and be used for analysis and trending.

Special data processing may then be required to further analyze any spacecraft anomalies. Also, short and long term trending may be done. Short term trending may involve comparing a sample orbit signature of a telemetry point with a comparable earlier orbit signature to monitor for degradation or orbit patterns. Long term trending may involve such things as plotting minimum, mean and maximum values of telemetry points (1 point per orbit or day, etc.) over a longer time span to monitor seasonal or longer term trends. Long term earth projection plots may be used to monitor single event upsets or other environmental effects on spacecraft performance.

Onboard packetization strategies

For the data that is only of interest when it changes, such as command execution counters, command execution error counters, other error counters and digital status data, onboard storage space could be saved if this data were stored only when something changes. Depending on how many telemetry points fall into this category, one or two packet formats (more if large amounts of these points exist or if separation by subsystem is desired) should be specified. To save storage space if there are more than a few of these points, two packets should be defined separating data that is expected to periodically change and data that should never, or very rarely, change.

This data could then be sampled synchronously onboard, formatted into a packet and compared to the previous sample. If the comparison showed a difference, the old and new (or just the new) packets could be stored. Else, the old packet could be discarded and the new packet saved for comparison with the next sample. The sampling rate should be frequent enough to provide the time of the change to within a few seconds and should also be frequent

enough to catch every state transition. If a relay can be powered on and back off between samples, the ground operations team may never know that a transition occurred. If there is a concern of scheduling reads to quickly, the individual subsystem could maintain a history of the last few settings of the discrete and the associated times or just keep track of all transitions between readings and set a flag if more than one transition occurred since the last sampling.

Finally, this data could also always come down in real-time, if there is enough down link bandwidth, or could be stored or down linked on command. This would provide the ground operations team a sanity check on the data to ensure that a change does not go undetected due to a lost packet. Another possibility is to treat some or all of these items as spacecraft events and issue an event message containing the telemetry mnemonic, the previous and current values and the time of the transition rather than store the full data packet that contains a sampling of all of the discrete, infrequently changing values. The configuration and counter packet(s) could then be available for storage or real-time down link on command in order to provide sanity checks.

The next type of data is the frequently or constantly changing data. This category includes analog data, flight software task execution counters and data storage accounting statistics. Analog data, when synchronously stored, is generally compromised. By this, I mean that this data is usually stored at a rate that is to frequent when the spacecraft health is nominal and not frequent enough for analysis purposes when there is a spacecraft anomaly to investigate.

One way to improve upon this is to take advantage of current flight certified onboard computer capabilities (usually required to take full advantage of the CCSDS protocol anyway) to move analog and discrete monitoring functions (limit, state and configuration checking) from the ground data system to the spacecraft. This would give the spacecraft the ability to detect its own anomalies, take immediate command response to anticipated contingencies and provide higher resolution data for use in ground analysis when a discrepancy occurs. Analog data could be stored in a circular buffer onboard the spacecraft. This buffer would be sized to hold approximately one orbit, or other suitable time increment, of high resolution analog data. If a monitor violation is detected by the onboard computer, then the contents of the circular buffer, or an appropriate subset of that data, can be transferred to the data storage recorder for subsequent ground data analysis of the problem. During the rest of the time, this data could be filtered before being recorded such that enough data is always available to do performance trending, but higher resolution data is available for anomaly analysis. By allowing this circular buffer to be stored or down linked on command, daily high resolution or "typical orbit" plots could be maintained. Filtered data would then fill in the rest of the day.

With a more sophisticated onboard computer, the function of calculating and saving the maximum, minimum and mean values for a given telemetry point, on a per orbit or other incremental period, could also be migrated to the spacecraft. This could be particularly useful for power system analysis, where it is often desired to identify when a current or voltage spike may have occurred. Currently this is like looking for a needle in a haystack as the synchronous data sampling either results in the spike not being recorded or in the inability to determine exactly how long the event actually occurred. By combining min./mean/max. data with high resolution data output when a monitor is violated, work on detecting, monitoring and isolating power spikes could be greatly enhanced. Also, min./mean/max. data could give a good, quick view of the spacecraft thermal performance.

If the min./mean/max. data was sampled directly from the analog source, or the high resolution buffer, a better data set could be obtained onboard than could be calculated on the ground from the lower resolution, filtered data that would be stored onboard when spacecraft functions were nominal. This would result in better long term trending data.

Flight software task execution counters are primarily used for diagnostic purposes. Since these counters have a possibility of rolling over multiple times each second, this data needs to be output at a high rate to be of any use. An output of delta values or messages per second, vice absolute counts, could be more useful. Also, since this data is really a diagnostic tool, it should be filtered out and only stored or down linked on command, when necessary. It is also possible to provide flags to indicate that software tasks are running and execution counters are incrementing. Actual counts would only be needed if trying to study environmental effects on task loading or to diagnose a new flight software patch. For example, on SAMPEX we attempted to see if flight software tasks were running at a significantly different load during ground contacts or when over the poles when science data output increased due to increased particle events.

Data storage accounting statistics are generally only used during ground contacts to verify that the data stored onboard was completely captured on the ground during a recorder dump. Therefore, storage of this data is usually not necessary. However, it may be of interest to do a study of how often and when data is stored based on environmental factors. Therefore it may be desired to allow storage of this data in a fashion similar to the task execution counters mentioned above.

Another way to save onboard storage space for constantly changing telemetry points is to increase the efficiency of CCSDS packetization by increasing the packet size. Each packet header requires 112 bits. Packet size can be increased by supercomming the data (multiple samples assembled within the same packet), however this requires that the ground data system have the capability to split the packet apart and extrapolate the time code. Another way to increase packet size is to specify packet contents based on output frequency rather than by source. This allows fewer, larger packet types to be managed by the spacecraft, at a higher storage efficiency, but at the expense of being able to sort data by spacecraft telemetry source once on the ground.

The final types of data packets are those which are stored and/or down linked only on command. This already implies that this data would only be generated when needed and, other than combining data packets if possible, no other optimization techniques are necessary.

Implications to spacecraft data storage sizing

Since many of the proposals in this paper suggest event driven rather than synchronous data output, it is now more difficult to optimally size the amount of storage space needed for housekeeping data. Science data storage space is not optimized if housekeeping data storage space is sized for the worst case.

Therefore it is recommended that housekeeping storage space be sized to hold the expected amount of housekeeping data under nominal conditions, allowing for any additional storage space that may be desired to allow two or more down link opportunities for any particular data dump. Then some space could be reallocated from the science allotment, if needed, in order to store higher packet output rates generated when spacecraft algorithm's explained above increase the amount of housekeeping data saved. By sharing some science storage allocation, science data output can be maximized when spacecraft operation is nominal. This shared area could then be reallocated to housekeeping when spacecraft problems cause higher packet output rates to be needed. It may even be possible to set this up in a way that less valuable science data would be lost in the event of a problem. Even though this could result in periodic losses of some of the science data, it should allow more science to be recorded during nominal periods when the housekeeping data output is reduced to a

minimum and could allow for more expedient detection and correction of small problems before they become big problems.

Summary

By taking advantage of the event driven nature of the CCSDS protocol and by migrating some of the basis data checks from the ground to the spacecraft, the output of spacecraft housekeeping data can be optimized to provide a more prudent balance with science data. By monitoring discrete telemetry, only information on state transitions or counter increments need be transmitted to the ground rather than synchronous output of redundant data. On command discrete telemetry packets can provide the ground with a sanity check. Also, by having the spacecraft monitor analog limits and subsystem configurations, analog data output can be throttled to provide increased data output rates when potential problems exist while filtering this output during nominal operations. By having the spacecraft calculate max./mean/min. data, long term trending of spacecraft performance can be greatly enhanced. Finally, by sharing recorder overflow space with science, optimum science output can be achieved when spacecraft performance is nominal and finer resolution housekeeping data can be output when there is an indication of a performance problem.

Systems Engineering

4. Systems Architectures — Page 1277

* *Presented in Poster Session*

GROUND SEGMENT STRATEGIES AND TECHNOLOGIES IN SUPPORT OF COST REDUCTIONS

K. Debatin
ESOC

Paper Not Available

PRECEDING PAGE BLANK NOT FILMED

PAGE 1278 INTENTIONALLY BLANK

111186 N95-17601
353315
P. 8

Mission Operations Centers (MOCs): Integrating Key Spacecraft Ground Data System Components

Randy Harbaugh
National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, Maryland, U.S.A.

Donna Szakal
Computer Sciences Corporation
Laurel, Maryland, U.S.A.

ABSTRACT

In an environment characterized by decreasing budgets, limited system development time, and user needs for increased capabilities, the Mission Operations Division (MOD) at the National Aeronautics and Space Administration Goddard Space Flight Center initiated a new, cost-effective concept in developing its spacecraft ground data systems: the Mission Operations Center (MOC). In the MOC approach, key components are integrated into a comprehensive and cohesive spacecraft planning, monitoring, command, and control system with a single, state-of-the-art graphical user interface. The MOD is currently implementing MOCs, which feature a common, reusable, and extendable system architecture, to support the X-Ray Timing Explorer (XTE), Tropical Rainfall Measuring Mission (TRMM), and Advanced Composition Explorer (ACE) missions.

As a result of the MOC approach, mission operations are integrated, and users can, with a single system, perform real-time health and safety monitoring, real-time command and control, real-time attitude processing, real-time and predictive graphical spacecraft monitoring, trend analysis, mission planning and scheduling, command generation and management, network scheduling, guide star selection, and (using an expert system) spacecraft monitoring and fault isolation. The MOD is also implementing its test and training simulators under the new MOC management structure.

This paper describes the MOC concept, the management approaches used in developing MOC systems, the technologies employed and the development process improvement initiatives applied in implementing MOC systems, and the expected benefits to both the user and the mission project in using the MOC approach.

INTRODUCTION

The National Aeronautics and Space Administration (NASA) Goddard Space Flight Center (GSFC) Mission Operations Division (MOD), in partnership with the Computer Sciences Corporation (CSC) Control Systems Technology Group (CSTG), developed the Mission Operations Center (MOC) concept to improve the MOD's spacecraft ground data systems. The focus of this effort was to enhance system operability and increase functionality while lowering development and operational costs and shortening development time.

Four key advances within and outside the MOD arena contributed to the development and refinement of the MOC concept: reengineering of the MOD mission operations concept, restructuring of management to a mission-oriented structure, industry development of enabling technologies, and application of improvements in system development processes.

Reengineering of the MOD mission operations concept provided the framework for developing the MOC concept. Restructuring from a multimission to a mission-oriented management organization provided the vehicle for efficiently and effectively implementing the concept. Enabling technologies such as powerful workstations and industry standards

PRECEDING PAGE BLANK NOT FILMED

contributed to the feasibility of the concept. Improved system development processes in all life-cycle phases contributed to the cost-effectiveness of the concept.

DEFINING THE MOC CONCEPT

Driven by user demands for mission-unique systems with improved operability and increased functionality, mission profiles with accelerated spacecraft schedules, and NASA budgets in steady decline, the MOD reengineered its overall mission operations concept. This activity viewed mission operations from an MOD-wide level, with the goals of maximizing the operations that a single user can perform while minimizing system development time and reducing operational and development costs. The MOC concept makes significant strides toward achieving these goals.

Before the MOC, the MOD developed ground data systems and conducted mission operations in host-based, multimission environments supported by separate, independent branch organizations. For example, the Control Center Systems Branch (CCSB) (GSFC Code 511) developed Payload Operations Control Centers and the Spacecraft Control Programs Branch (GSFC Code 514) developed mission planning and command management systems. As a result of the reengineering activity, which encompassed the operational functionality of all of the MOD's independent systems, the MOC system was defined as an integrated, comprehensive, mission-unique system with a single user interface and the capabilities necessary to support the MOD's mission operations.

With a MOC system, the user can, from a single workstation, perform traditional mission operations including real-time spacecraft health and safety monitoring, real-time spacecraft command and control, trend analysis, mission planning, command generation and management, and network scheduling as well as newly added operations such as mission operations planning and scheduling, real-time and predictive graphical spacecraft monitoring, real-time attitude processing, guide star selection, and spacecraft subsystem monitoring and fault isolation .

The broadened view of the MOD's mission operations, free from the past organizationally induced partitions of functionality, enables comprehensive system engineering that considers only the technical aspects of the MOC system definition. The resulting MOC system eliminates redundant capabilities within the MOD; eliminates or simplifies interfaces to and within the MOD; and allows for cost-effective, systemwide solutions. Because of these improvements, a MOC system can be developed in less time and at lower cost than the traditional, independent ground data system implementations.

MANAGING MOC DEVELOPMENT

The concept of developing an integrated MOC ground data system naturally led to the concept of an integrated, mission-oriented management structure. To provide the vehicle for efficiently and effectively implementing each MOC system, both the MOD and CSC restructured their management organizations to create a single, mission-oriented MOC management team. Recognizing the potential for improving coordination and communication between the mission's MOC system and the mission's standalone test and training simulator [traditionally developed under the Simulations and Compatibility Test Branch (GSFC Code 515)], the MOD and CSC placed development of the simulator under the MOC management structure.

The resulting mission-oriented MOC organization is headed by a system manager and is supported by a MOC system engineer; managers of the major MOC components and the simulator; and knowledgeable, technical component experts. This approach retains the expertise of the traditional organizations and, for the first time, combines the functionality of the MOD ground data systems previously developed by independent organizations under a single MOD-level system manager.

The major advantages of this management approach over the traditional approach include consolidation of mission budgets; closer coordination of system capabilities and schedules; integration of a major portion of the mission ground system earlier in the ground

system life cycle; provision for a single point of contact to the mission projects and users; and improvements in communication, coordination, and cooperation among the experts from the various ground data systems. Clearly, these advantages could only be realized when the management authority, responsibility, and control rested in the hands of a single, system manager whose primary focus was to manage development of the MOC system and the simulator.

Consolidating mission budgets under a single manager with the requisite responsibility and authority simplifies the planning of projected budgets and the reporting of actual spending on a per-mission basis. Further, single-manager responsibility for most of the MOC components results in increased flexibility in assigning resources among the components that need it. With the MOC approach, timely, system- and component-level budget decisions can be made within the MOC organization from a balanced and informed viewpoint.

Closely coordinating component development schedules and the capabilities to be implemented according to the consolidated MOC master schedule significantly improves the readiness of a major portion of the mission's total ground system. In the traditional approach, one ground data system's capabilities and schedules were usually developed with limited insight into the needs of the other ground data systems. This lack of close coordination sometimes resulted in the need for additional temporary software to simulate missing capabilities and delayed mission ground system testing of these capabilities. In the MOC approach, simulator and component schedules and capabilities are closely coordinated so that each MOC component fully supports the others and the MOC and simulator systems fully support each other at the scheduled time. This level of coordination significantly reduces time spent waiting for independent ground data systems to get synchronized in support of mission ground system testing. Although planning a MOC development schedule is slightly more time consuming and complex than in the traditional approach, monitoring projected and actual schedules is much quicker and easier because there is one composite schedule.

Integrating major portions of the mission's ground system during the development phase of the life cycle, rather than during the integration and test phase, significantly reduces mission ground system interface, integration, and end-to-end test time. The schedules and capabilities of the MOC system components are not only coordinated, but the major efforts of integrating and testing them and also testing the integrated MOC system with the simulator are accomplished by the development team before delivery of either system to GSFC. In the traditional approach, this integration and testing occurred after delivery of each of the independent ground data systems, when interface problems are difficult to isolate and repair quickly. The extensive, advanced planning of the MOC master schedule, which considers the project's test needs, coupled with the development team's expertise in integrating and testing the MOC system, improves the overall quality and readiness of the ground system earlier than previously possible.

Because the MOC organization provides a single point of contact, the MOD speaks with a unified voice to the mission project and MOC system users. Traditionally, a mission project had to communicate with each of the MOD branch organizations, an inefficient and time-consuming process. Also, users had to communicate with the developers from each of the MOD independent ground data systems to convey and receive information. The MOC approach ensures a direct, timely, and consistent flow of information from the MOC team to the mission project and the users. For example, with a MOC system, there is a single set of comprehensive, formal reviews (e.g., a single system requirements review, preliminary design review, and critical design review) to attend and critique; there are fewer documents to review and approve than with the traditional approach (e.g., a single comprehensive requirements specification rather than multiple ones); and, as an added benefit, the resources needed to prepare, present, and maintain these formal reviews and documents are reduced.

Improved communication, coordination, and cooperation among the technical experts from the various ground data systems ensures the timely development of robust, cost-effective MOC systems. The single, cohesive MOC team

shares a common focus and a common goal: the successful implementation of the MOC system. The team makes decisions to support this goal, relinquishing conflicting demands and diverse approaches from the originating organizations in favor of a unified management and technical approach.

The MOC system engineer and component experts regularly share their expertise and insight with each other. Cross-checks of understandings highlight discrepancies early, allowing them to be solved when resolution is less costly. For example, on the X-Ray Timing Explorer (XTE) MOC, discrepancies existed in early mission documentation describing the telecommand packet checksum calculation. Because the simulator experts on the team knew how the flight software performed this calculation, they were able to resolve the problem quickly and with no cost impact. Typically, with the limited cross-check of understandings between simulator and control center system experts in the past, a problem such as this would not have been found until actual ground system integration testing with the spacecraft or the simulator, when problem repair is more costly.

The MOC team has also used their broadened view of the system to identify and implement more robust technical solutions. For example, the traditional simulator, control center, and command management systems each used different approaches and different data base software to process the mission's project data base. For the single, integrated MOC system, the team has identified and implemented a more rigorous relational data base solution with increased functionality over any of the traditional systems.

Working as a cooperative team, the MOC component experts have identified and eliminated redundancy among the components, reducing the amount of software that must be developed, tested, and maintained. Traditional capabilities, as well as new functional capabilities, are available earlier. For example, the selection of a single user interface means that the time traditionally spent developing and maintaining multiple user interfaces can be spent enhancing the functionality of the selected user interface. For the integrated MOC system, the team has also eliminated the formal interface between the control center and command management systems. Significant savings have been realized in eliminating the formal definition, negotiation, control, integration, and testing of this interface because these efforts are now performed within the MOC organization. Traditionally, several separate MOD and mission project organizations needed to be involved.

The most important challenges in defining the MOC management approach were to develop a mission-oriented organization that retained the expertise of the various components of the MOC system and to minimize the risks to the success of the mission while implementing the new technical and management approaches.

The MOC management approach capitalizes on the use of technical and management expertise from each of the ground data systems. The depth of knowledge provided by these component experts, coupled with the breadth of knowledge of the system engineer, is essential to the success of any MOC implementation. Management risks are minimized because component experts are part of the MOC team and because the best practices from the originating MOD branch organizations have been selected and implemented. Techniques such as the use of multimission working groups and matrices of expertise have been expanded to encompass the full MOC functionality. These techniques, successfully demonstrated in the traditional organizations, make possible high levels of software reuse across and within mission implementations. In the control center area, for example, software reuse levels of over 70 percent are regularly achieved. Technical risks are minimized because the selected MOC architecture is a natural extension of the in-place, highly successful system architecture described below. Use of these management and technical strategies minimizes the risk of the overall MOC concept.

To further reduce the risks, the MOD initiated a pilot project in October 1992, selecting the XTE mission as the first MOC system implementation. The MOD, CSC, and the mission project have closely monitored the progress of this MOC via various technical reviews and regular management reviews. By June 1993, the

XTE MOC pilot project showed such early promise and enthusiastic user support that the MOD reassigned the TRMM mission, which started development as separate control center, command management, and simulator ground data systems, as an integrated MOC system implementation and a standalone simulator under the new MOC management structure.

IMPLEMENTING THE MOC

MOC Architecture

The availability of enabling technologies such as powerful workstations and networks, distributed processing, commercial off-the-shelf (COTS) products, and industry standards contributed to the feasibility of the MOC concept. The effectiveness of their use was successfully demonstrated in the MOD's CCSB-developed Transportable Payload Operations Control Center (TPOCC) system philosophy and architecture. The MOC concept extends the use of TPOCC to cover broader functionality.

The TPOCC architecture is based on the use of industry standards, COTS components, custom reusable components, and distributed processing using client/server technology. It features interconnected hardware that provides systemwide access to data and distributed processing that is flexible and transparent to the user. It supports the dedicated use of a workstation for isolated functions or the use of a single workstation for multiple functions. It also allows single functions to be spread across multiple processors to provide needed levels of processing and data throughput. The state-of-the-art graphical user interface, which features a windowing environment, significantly increases system operability.

The TPOCC architecture is designed to be evolutionary in that new technology can be inserted into the basic system framework without disrupting the overall architectural approach. This extendable architecture easily supports integration of independently developed components that follow its fundamental precepts.

Each MOC system, which uses TPOCC's hardware architecture approach, is sized to meet its mission's data and operational needs and consists of a network of inexpensive, heterogeneous COTS workstations, X-terminals, and front-end processors (i.e., single-board computers). The architecture reflects a commitment to industry standards such as VME, Ethernet, RS-232, RS-422, and SCSI. For the MOC, a RAID array, optical disk, CD ROM, and 3-D graphics devices are added to the basic TPOCC architecture to support the broader functionality of a MOC.

The MOC software architecture approach, like TPOCC's, consists of distributed processing using client/server technology, adherence to open system communications standards, extensive use of COTS products, and implementation of reusable custom code. Most of the MOC software is written in C or C++ and is designed to be independent of the hardware, thus making it easily portable to other platforms.

All MOC software components are implemented following open system communications standards such as the Transmission Control Protocol/Internet Protocol (TCP/IP), external data representation (XDR), and network file system (NFS). The commercial standards for the MOC's graphical user interface include X-window and Open Software Foundation's Motif software. The use of industry standards facilitates incorporation of COTS products, generic systems, and independently built components without impacting the overall software architecture.

The MOC system is flexible and extendable. It supports, from a single workstation, the functionality previously dispersed among many minicomputers and mainframe systems, thus increasing the number of operations that a single user can perform. The MOC system reduces operational costs because multiple, independent systems are consolidated; workstations replace more expensive minicomputers and mainframe systems, and computer operators are no longer needed to support multimission computer facilities.

Process Improvements

Application of improved processes in the requirements, design, implementation, integra-

tion, and test phases of system development contributed to the cost-effectiveness of the MOC concept. In an atmosphere of continuous process improvement, the MOC development teams have applied several improvement initiatives to the development of the MOC systems.

The mission MOC teams have improved the process of defining the MOC system requirements. During the requirements definition phase, joint developer and user teams, sometimes referred to as Joint Application Development (JAD) teams, define the requirements. Using the JAD approach, users familiar with the specific mission requirements and operational needs and developers familiar with existing software capabilities are able to quickly identify mission-unique needs. The JAD team uses an existing set of requirements from other, similar missions as a base for defining the new mission's requirements. This approach results in the timely definition of requirements because the JAD team, rather than starting from scratch, simply analyzes the baseline and makes additions or deletions as appropriate. This approach also maximizes the reuse of existing software, limiting detailed requirements analysis to mission-unique areas. The Advanced Composition Explorer (ACE) MOC JAD team is using this approach with the XTE MOC requirements providing the basis for requirements discussions.

The ACE MOC JAD team is also piloting the concept of users and developers jointly documenting requirements rather than each group independently writing and cross-referencing separate, configuration-controlled documents. The team documents requirements on-line using the Requirements Generation System (RGS) data base tool. This approach is also expected to save considerable time and effort.

The mission MOC teams have also implemented improvements in the design process. During the design phase, extensive technical exchange meetings are held both within a specific mission MOC (i.e., cross-function) and across the MOCs of other missions (i.e., cross-mission). Each MOC's system engineer and component experts regularly hold cross-function technical exchange meetings to design portions of the software so that they can be used by multiple components, thus maximizing reuse within a mission MOC. In addition, the MOC system engineers and component experts regularly hold cross-mission technical exchange meetings to design specific components into generic and mission-unique building blocks, thus maximizing reuse across MOC missions (i.e., generic component software is designed with mission-unique "hooks"). This MOC design approach results in a comprehensive, cohesive system design that eliminates organizationally induced walls between functional components.

During the implementation phase, the mission MOC teams' strict adherence to system development standards and use of a standard user interface permits multiple components (i.e., multiple portions of the system) to be developed concurrently. Although this is not a new process, its implementation during MOC system development is essential. The mission MOC teams have also expanded the use of advanced COTS software development tools such as SoftBench, Branch Validator, and Purify to assist them in writing and debugging software.

The implementation of these development improvements allows the mission MOC teams to capitalize on three aspects of software reuse: reusing existing custom-built and generic software components; designing custom software with new functionality for future reuse; and integrating existing, standalone generic systems and COTS products. Each MOC team works with users to define mission requirements that maximize the reuse of existing custom-built and generic components (e.g., existing mission software, TPOCC generic software) while still meeting each mission's unique needs. Sharing requirements expertise across each mission allows the MOC teams to design custom code for future reusability because generic components are identified and developed to permit mission-unique extensions. Each MOC's design also integrates COTS products (e.g., ORACLE) and standalone generic systems such as the Generic Spacecraft Analyst Assistant (GenSAA) and the Generic Trend Analysis System (GTAS). The use of these techniques reduces the amount of new code needed while increasing functionality. For example, approximately 50 percent of the

first MOC's code and over 80 percent of the second MOC's code consists of reusable components (not including the integrated generic systems or COTS products). This percentage is expected to increase as additional MOC capabilities are implemented for future reuse and spacecraft standards continue to be formulated and implemented.

The mission MOC teams have also instituted improvements in the integration phase. One of the major challenges for any MOC system is the integration of the many components that it comprises. Successful integration of a MOC system is a special and complex problem. The complexity of integrating "externally" developed components (i.e., components developed by other organizations), for example, encouraged definition of a formal integration procedure. This procedure includes requirements for extensive planning, preparation, and monitoring of the integration activity. For example, for components developed within the MOC organization, the MOC system engineer and component managers require demonstrations of, and explicit documentation about, each developer's software before that software is integrated with the total MOC system. In addition to these improvements, for the first time, the mission's test and training simulator has been collocated with the MOC system in the development environment, substantially improving the developers' ability to test the integrated MOC system. The MOC and simulator developers' ability to extensively exercise their systems before they are delivered to GSFC significantly improves the quality and robustness of each system.

During the test phase, the test process has been improved by combining traditionally separate system, acceptance, and user test teams into a single test team (independent of the development team) and by moving this level of testing from the traditional postdelivery timeframe into the predelivery timeframe. The combined, concurrent testing by this team reduces overall MOC system test time while increasing testing effectiveness. When the test team finds problems that must be repaired before the system is deemed ready for operational use, the development team corrects the problems. This extensive, independent, predelivery testing of the integrated MOC system reduces the amount of time necessary for mission-level ground system interface and integration testing because not only have some of the traditional interfaces been eliminated, but also a major portion of the overall ground system has been tested.

BENEFITS OF THE MOC APPROACH

The MOC approach provides major benefits to its users. Probably the most important of these benefits is the integration of mission operations with mission specialists collocated in the MOC's office-like, workstation environment. Traditional, host-based systems located in various multimission computer rooms required that users be able to operate several independent systems. On each of these systems, a user could perform only one operation from each terminal, requiring that user to monitor up to three or four terminals at a time, depending on the number of simultaneous operations to be performed. The MOC's mission-oriented, integrated system, with a windowing environment and distributed processing, allows the user to perform and monitor multiple operations from a single workstation, a vast improvement over traditional systems.

A second important benefit to the user is the MOC's state-of-the-art, standardized graphical user interface that provides the same "look and feel" across all components of the MOC. In addition to traditional tabular data displays, this interface supports graphical data representations such as plots, bars, dials, pie charts, and timelines, enabling users to rapidly distinguish anomalous situations. Menus and input panels are intuitive to operate, and, with only one consistent user interface to learn, user system training as well as cross-component training is simplified. This improved system operability, coupled with the increased functionality provided by a MOC system, provides the user with all the tools needed to perform operational duties.

The MOC approach provides many benefits to the mission project. The MOC management structure provides the mission project with a single point of contact for a major part of the developing mission ground system. This improves and simplifies communication both to and from the mission project.

Another major benefit to the mission project is that MOC systems, as opposed to traditional implementations, are less costly and achieve operational readiness in a shorter period of time. With fewer system interfaces, operational and system development complexity and associated costs are reduced. MOC approaches such as elimination of redundant code among components, extensive software reuse, integration of COTS products and existing generic systems, and commitment to expanding the library of reusable custom components by designing for future reuse are recognized approaches that reduce costs. The MOC systems contain more capability and higher stability early in the development cycle because of the extensive reuse of existing, tested software and COTS products.

SUMMARY

The MOC approach to ground systems development makes great strides toward integrating the MOD's mission operations. This approach significantly increases the number of operations that a single user can perform simultaneously, substantially improves system capability and operability, and simplifies user training, while reducing operations and development costs and shortening development time.

Only 2 years since its inception, the MOC concept has realized its initial goals. As the MOC approaches continue to mature, and as more functionality is incorporated into its systems, the benefits to mission projects and the user community are expected to grow.

REFERENCES

Koslosky J., Mahmot R.., Mandl D., and Stratton W. (June 1990). *Transportable Payload Operations Control Center,* Proceedings of European Space Agency Symposium: Ground Data Systems for Spacecraft Control (pp. 653-658).

Beach E., Giancola P., Gibson S., Mahmot R.. (November 1992). *Customizing Graphical User Interface Technology for Spacecraft Control Centers,* Proceedings of SPACEOPS '92 (pp. 485-490).

Carlton D., Vaules, Jr. D., Mandl, D. (November 1992). *GSMS and Space Views: Advanced Spacecraft Monitoring Tools,* Proceedings of SPACEOPS '92 (pp. 375-380).

111187 N95-17602

354317 p.8

ATOS: Integration of Advanced Technology Software within Distributed Spacecraft Mission Operations Systems.

M Jones, J Wheadon, W O'Mullane
European Space Operations Center
Robert-Bosch Str. 5,
64293 Darmstadt,
Germany.

D Whitgift, K Poulter
Logica UK Ltd, UK

M Niézette, R Timmermans
Space Applications Services, Belgium

Iván Rodríguez, R Romero
GMV, Spain

Abstract: The Advanced Technology Operations System (ATOS) is a programme of studies into the integration of advanced applications (including knowledge based systems (KBS)) with ground systems for the support of spacecraft mission operations.

Keywords: OO, ATOS, AAM, DARPA, KQML, KIF, Knowledgebase, Infrastructure, Operations, Interface.

1 Introduction

The aim of ATOS is to allow data to be shared by multiple knowledge based and traditional applications.

At ESOC several studies have applied knowledge based techniques to specific areas of mission operations, producing a number of independent prototype KBSs. The studies showed that a great deal of common information was used in many of the applications. However, the various prototypes used different KBS tools and knowledge representations which meant they could not be easily integrated. The initial objective of ATOS is to find a solution to this integration problem without imposing a knowledge representation on all applications.

Section 2 examines the problem domain of ATOS. Section 3 outlines the ATOS architecture and section 4 discusses a prototype of the architecture and of applications which use it.

2 SMOS Integration Problem

A Spacecraft Mission Operations System (SMOS) comprises the set of facilities needed to carry out all the mission operations. Mission operations can be split into four areas:

- *Mission Preparation*. The tasks for the preparation and configuration of the Mission Control System (MCS) prior to the start of the mission, as well as the maintenance and updating of the basic reference mission knowledge of the MCS, during the mission.
- *Mission Planning*. The planning and scheduling of mission operations activities.
- *Mission Operations*. All tasks involved in control monitoring, and reporting of the mission.
- *Training / retraining* of operations staff.

In general, independently developed software, possibly running on different platforms will support each of these areas. The areas, however, are far from independent: mission preparation produces the database for operations, mission planning produces the plan of operations to be executed by mission operations and the progress of mission operations conditions the plan. Furthermore, applications use complex data and may use knowledge based techniques.

There are a number of considerations with such systems:

- Because such systems are large and difficult to implement, re-use of many applications is necessary to avoid loss of investment.
- The required capabilities change during the system's working life, for example, for new missions, as users' needs change, or when new technology (platforms, networks...) is introduced.

- The various applications making up the whole system are frequently inflexible, with rigid and restricted interfaces. For example, replanning in the event of failures can be cumbersome because of the interface between the control and planning systems.
- The applications also make use of knowledge about the spacecraft, the ground systems and the operational procedures. Parts of this knowledge are held centrally, but a significant amount is held locally by the applications, each of which may use its own representations and conventions. This leads to potential duplication and inconsistency of knowledge and related problems in system maintenance.

These problems demand solutions: budget restrictions no longer allow the luxury of reimplementing large parts of systems for new missions.

2.1 Solution: The ATOS Approach

A combination of the following two approaches helps to address the problems outlined above:

- Implementation of generic applications from which specialisations may be built.
- Use of principles of federation to integrate heterogeneous applications into a single system.

2.1.1 Generic Systems

Individual parts of a mission operations system can take the approach of developing generic applications which can be specialised for particular missions. The new generation of ESA Spacecraft Control Operations System (SCOS-II) [6] is a good example of this. In SCOS-II, the basic functions of spacecraft control and monitoring are implemented as an object-oriented class library, from which mission systems can be built. Specialisations to mission needs can be provided using "Implementation by Difference" [3].

2.1.2 Federation

Development from generic components is fine, but a problem is usually approached from several user perspectives resulting in several specialist applications. To integrate these we need a generic communication mechanism for software components, analogous to what the SCSI protocol does for hardware components.

ATOS is a federation-enabling technology. Each application (of a federated MCS) has only to provide an interface to the ATOS infrastructure to allow it to use data (and have its data used) by other applications. In effect this extends the object-oriented philosophy to the application level by providing a consistent interface to a set of applications which a developer may use without knowing implementation details of the individual applications.

Although ATOS has been motivated by the needs of mission operations, the concepts (and possibly even the emerging tools) are not restricted to that discipline. The approach could be used in any area in which heterogeneous applications, possibly originally designed to be "stand alone" (i.e. without regard to eventual integration) must be made to work together.

3 The ATOS Architecture

This section describes the architecture which realises the approach to integration outlined above. Section 4 describes prototypes of this architecture and of applications which use it, and illustrates how the architecture works in practice.

3.1 The Mission Information Base

Each application in the ATOS environment is called an ATOS Application Module (AAM). AAMs communicate with each other via the ATOS infrastructure.

As shown in figure 1, each AAM has its own knowledge base. The Mission Information Base (MIB) is defined to be the union of the knowledge bases of all the individual AAMs. The scope of the MIB is thus very broad and encompasses:

- *Flight Operation Plans (FOP)*, including timelines (a scheme of mission operations activities for a particular mission phase or scenario) and Flight Control Procedures.
- *Documents*, including text and graphics, for example the spacecraft users manual.

- *Design information*, including the behaviour of components of the spacecraft.
- *Traditional spacecraft databases*, for example parameter characteristics and telecommand characteristics.
- *Rules and operational constraints* of the mission; for example, if the spacecraft is in eclipse then the payload is on standby.

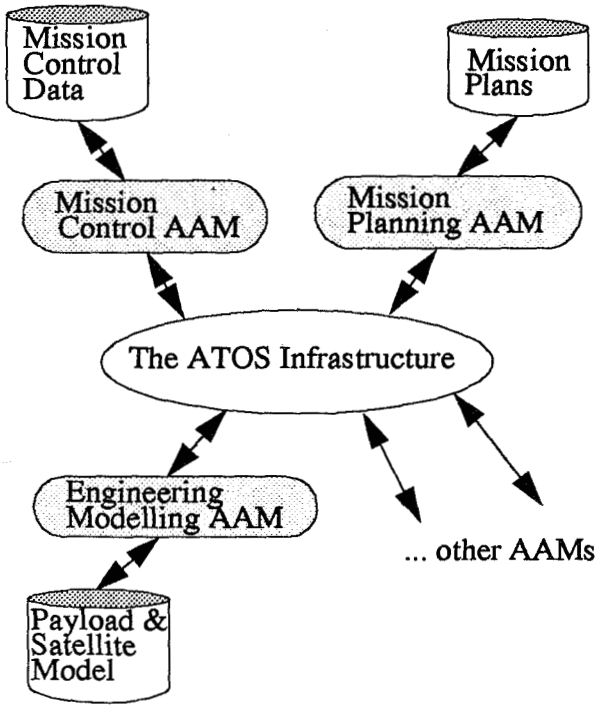


Figure 1 AAMs and the ATOS Infrastructure

The AAMs which manage components of the MIB may be physically distributed, may use different approaches to structuring knowledge (relational, object-oriented, rule based) and may use different tools for storing knowledge. The MIB is thus a federated database of loosely coupled, heterogeneous components.

3.2 The Functions of the Infrastructure

The ATOS infrastructure is the glue which integrates AAMs. The infrastructure "facilitates" (in the sense discussed in [1]) the integration of AAMs by:

- *Routing a message* to an AAM which provides the information or service required by the message.
- *Maintaining links* between information items in different components of the MIB.
- *Detecting significant changes* in the state of the MIB and informing AAMs accordingly.
- *Controlling access* to the information and services provided by AAMs.
- *Maintaining a timetable* that describes which AAMs can use which services of other AAMs and when. This timetable is updated by a mission planning AAM.
- *Logging messages*, as requested.
- *Buffering* messages before they are read.

Clearly some of these services are more innovative and interesting than others. Later sections of this paper concentrate on message routing, link management and detection of change in the MIB.

3.3 The Ontology of Shared Knowledge

AAMs must be able to share knowledge. For example, the results of mission planning are inputs to mission execution; details of a detected anomaly are the basis of fault diagnosis. [3] includes a detailed discussion of the importance of knowledge sharing in spacecraft operations.

To share knowledge AAMs must have a common understanding of concepts and terms which is provided by the ontology.

In ATOS, the ontology is written in a declarative, formally defined language called Ontolingua [2] which is:

- Expressive, so that rules and behavioural knowledge can be shared between AAMs.
- Independent of any particular approach to structuring knowledge.

Note that although Ontolingua allows terms and concepts to be defined using rules, the ATOS infrastructure does not infer knowledge from these rules - that is the responsibility of the AAMs which use the concepts.

The most basic use of the ontology is as a paper standard to which AAMs comply. Thus if there is a standard definition of the terms "resource", "schedule" and "activity" then

AAMs which comply with the standard are guaranteed to use these terms in the same way.

A second use of the ontology is to derive an AAM's knowledge structures. The ontology is written in a formal language (rather than, for example, English); it can therefore be translated into the tool-specific knowledge structures used by an AAM. This approach gives greater assurance that the AAM complies with the ontology and it can also reduce the effort of developing the AAM.

The next section discusses further uses of the ontology, including that of routing a message on the basis of its content.

3.4 The Metabase

As discussed in section 1.2, the MIB is distributed among AAMs, it is not held by the ATOS infrastructure. The infrastructure does, however, record the types of information in the MIB (as defined by the ontology) as well as information about selected objects in the MIB. The infrastructure does not actually store these objects but records their existence, some of their attributes and links between them. This view of the MIB is called the metabase.

Figure 2 depicts three example AAMs which each manage part of an MIB. One AAM uses a relational database, one uses a hierarchical database and one stores documents. The existence of certain MIB objects is recorded in the metabase; this is shown in the figure by a dashed line from the MIB object to its metabase image. The figure also shows links in the metabase between objects which are managed by different AAMs. For example, a tuple in the relational database might be described by a document to which it is linked.

The three major roles of the metabase in supporting knowledge sharing are outlined in the following three subsections. Each of these roles is based upon the global view of the MIB which the metabase provides.

3.4.1 Links between MIB components

The above discussion of touched on the first of these roles: the metabase stores links between objects in different components of the MIB which allow AAMs to navigate the MIB. Such links cannot be stored by any one AAM which only knows about objects in its own component of the MIB.

Link types are defined in the ontology and have different properties. For example, a link type might be defined to be many-to-one, or to be acyclic.

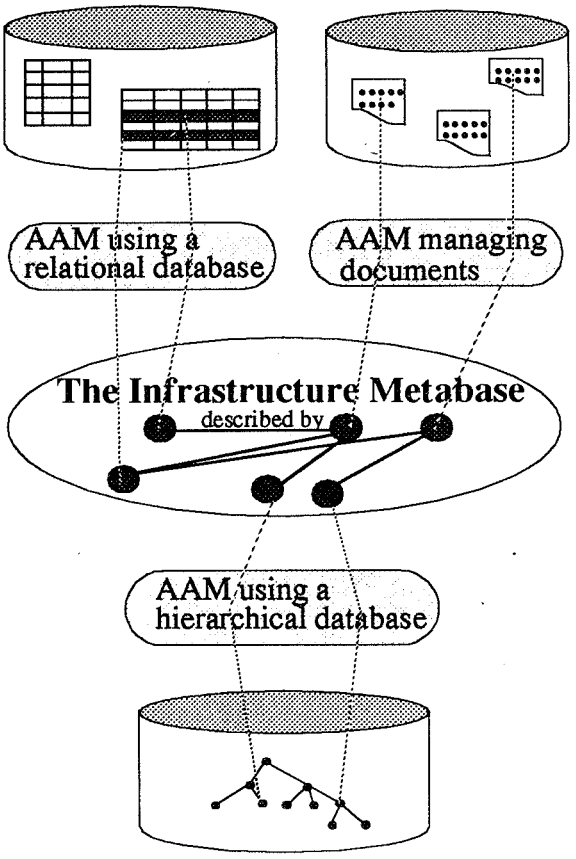


Figure 2 The Infrastructure Metabase

3.4.2 Detecting change

The second role of the metabase is in detecting significant changes in the state of the MIB. One AAM can request a second AAM to perform a specified action when a specified condition is triggered.

For example, an AAM may ask to be informed when the voltage of a battery falls below a certain level. A condition is expressed in terms of the state of the MIB; more precisely, a condition is a query over the MIB which evaluates to true or false; the condition is triggered when the value of the query changes from false to true.

Unfortunately, this approach to detecting change has two limitations. The first is that not all AAMs will be sophisticated enough to detect changes expressed as arbitrary conditions. The second is that an AAM can only monitor its own component of the MIB for change; no single AAM can effectively monitor a condition which involves two or more components of the MIB.

The two limitations are addressed by the infrastructure and its metabase: the infrastructure is able to detect changes to the state of the metabase and, because the metabase is a global view of the MIB, the scope of the condition is not limited to one component of the MIB.

This role of the metabase allows dependencies between components of the MIB to be managed. Imagine, for example, two AAMs one of which plans a mission and the other of which maintains information describing the design of the spacecraft. The metabase holds an abstraction of the plan and the spacecraft design, as well as links corresponding to dependencies of the plan upon the design. The planning AAM instructs the infrastructure to inform it when there is a change or correction to the design which requires the mission to be replanned. The planning AAM then obtains details of the change by querying the design AAM directly.

3.4.3 Content-based routing

When an AAM sends a message, it normally specifies explicitly the AAM which is to receive the message. Sometimes, however, an AAM does not know to which AAM to send the message. In this case the AAM instructs the infrastructure to send the message to the AAM which can best provide the required service or information. This is called content-based routing: AAMs first advertise their abilities to process messages, then the infrastructure routes messages on the basis of these advertisements.

As a simple example of content-based routing, an AAM might advertise its ability to provide the voltage all batteries. The infrastructure can then route a message which asks the voltage of a specified battery to this AAM.

The infrastructure can perform more sophisticated content based routing. Suppose, for example, an AAM advertises its ability to provide information about the power supply subsystem. The infrastructure can then route to this AAM a message which queries the current from the solar array if it knows from its metabase that the solar array is part of the power supply.

3.4.4 Managing the metabase

The metabase records the existence of some of the objects in the MIB and contains some of their attributes. Except for links, the metabase is a partial copy of the MIB. Objects, attributes and links should only be in the metabase if they are required for one of the roles of the metabase described above. The metabase is not expected to be large. For example, an AAM which manages documents might record in the metabase the existence of each document and the date of its most recent issue, but it would not hold the text of the document.

The accuracy of the metabase is, of course, the responsibility of the AAMs. For example, if the metabase records the voltage of a battery then the AAM which manages the corresponding part of the MIB must update the metabase when the voltage of the battery changes.

Which parts of the MIB should an AAM copy to the metabase? The simple answer is that this is a question for the designers of the system who decide how to integrate the AAMs. A more sophisticated and dynamic approach is that the AAM is sent a message which specifies the knowledge which it must copy to the metabase.

3.5 Messages and their Structure

In a spacecraft control system, AAMs and the infrastructure share knowledge by sending each other messages. The meaning of these messages is defined at three levels:

- As discussed in section 3.3, the ontology is a dictionary of the terms and concepts of spacecraft operations and is expressed in Ontolingua.
- A language called Knowledge Interchange Format (KIF) is used to express knowledge using the terms and concepts of the ontology.

- A protocol called Knowledge Query Manipulation Language (KQML) which AAMs use to communicate at run time.

This approach to knowledge sharing is based upon the work of the DARPA Knowledge Sharing Effort [5].

KIF expresses first order predicate calculus in a LISP-like syntax. It is not expected that AAMs use KIF internally; indeed, it is important that AAMs are not constrained to use a particular representation format. AAMs must therefore translate shared knowledge to and from KIF.

KQML provides performatives, i.e. message types, which define the intent of a message. Consider, for example, the following simple KIF sentence:

```
(position sample lower-most)
```

This sentence could be the content of any of the following three KQML performatives:

- `ask`, a query "Is the sample in its lower-most position?" with answer yes or no.
- `reply`, an answer to a question such as "What is the position of the sample?"
- `assert`, informing the receiving AAM that the sample is in its lower-most position.

The KQML performatives used by ATOS are adapted from those described in the draft KQML standard and include performatives in the following areas:

- Asking and replying to a question. Multiple answers to a question can be sent as one long reply or as a stream of replies each containing one answer.
- Asserting a fact to be added to the receiver's knowledge base.
- Advertising the sender's capability to perform a service.
- Instructing the infrastructure to route a message to the AAM best able to process it.
- Instructing the receiver to perform an action when a condition arises.

Possible arguments of a message include:

- *Content*. This is the body of the message, for example the actual query of an ask message.
- *Language*. The language of the content. Normally this is KIF but AAMs can communicate using other languages such as SQL and SGML. If they do so the content is not understood by the ATOS infrastructure.
- *Receiver*. The AAM to which the infrastructure should send the message.
- *Reply-with*. Whether the sender of the message expects a reply, and if so a tag for the reply.
- *Receipt*. Whether the sender requires a return receipt when the message is read.
- *Log*. Whether the message should be logged by the infrastructure.

Most messages are from one AAM to another (via the infrastructure, of course). Some messages are intended for the infrastructure alone; for example, messages which advertise an AAM's ability to perform a service, and messages which query or update the metabase.

All knowledge held by the infrastructure can be queried using KIF and KQML. There are two distinct parts of this knowledge:

- The *metabase*, the structure of which is defined by the ontology of spacecraft operations, as discussed in section 3.4.
- The *infrastructure database*, which contains the housekeeping data held by the ATOS infrastructure. For example, AAMs and their capabilities, message logs... The structure of this database is defined by the infrastructure ontology.

4 Prototypes and Example AAMs

4.1 The Infrastructure Prototype

The ATOS infrastructure has been implemented as a prototype which runs on Unix workstations.

AAMs normally run on different workstations which communicate with the infrastructure using TCP/IP. At the time of writing AAMs must also run on Unix workstations, however, it would be straightforward to port to other

platforms the software which must be linked into the client AAMs.

Two interesting aspects of the prototype are:

- *Storing persistent data.* The metabase and the infrastructure database are stored using a relational database. This requires that the ontology is translated to SQL data definition language and that KIF queries and assertions are translated to SQL data manipulation language.

  Translations must also be performed by AAMs which do not use KIF internally. An important difference is that each AAM has its own specific knowledge structures; it does not need to translate arbitrary KIF queries unrelated to these structures.
- *Content-based routing.* Advertised capabilities, and messages to be routed on the basis of their content, are both expressed in KIF. Matching a message with a capability involves conversion of the two KIF expressions to a normal form and then unifying them using the knowledge in the metabase.

4.2 AAM Prototypes

[7] describes a prototype tool called AMFESYS which maintains a model of a payload: a microprocessor controlled remotely programmable Automatic Mirror Furnace (AMF) for growing crystals in zero gravity.

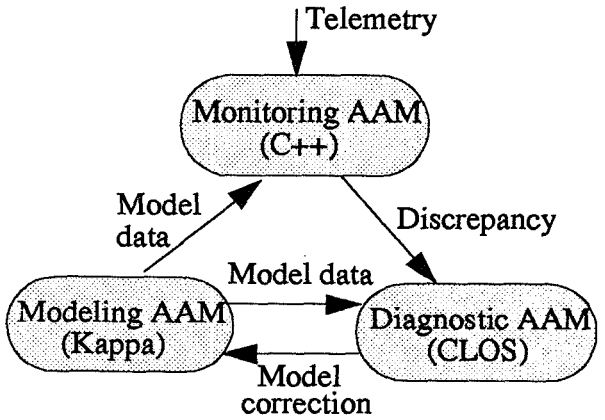


Figure 3 AAMs derived from AMFESYS

The AMFESYS tool has been decomposed into three AAMs as shown in Figure 3. The Modeling AAM maintains a model of the AMF which the Monitoring AAM compares with telemetry from the spacecraft. If a significant discrepancy is detected the Diagnostic AAM performs a rule-based diagnosis of the fault and then corrects the model.

The three AAMs use different approaches to structuring knowledge (C++, CLOS and Kappa) and interact with each other via the ATOS infrastructure. The ontology defines the structure of the AMF.

The following is a fragment of the AMF ontology which specifies a subclass of devices called `Spindles` and specifies that each spindle has a height. The fragment also identifies the instances of the class of spindles in the AMF: `SampleConvSpindle` (the sample conveyance spindle) and `LampDiskSpindle` (the lamp disk spindle).

```
(define-class Spindles (?x)
:def (devices ?x))

(define-relation Height (?x ?y)
:axiom-def (and(single-valued Height)
           (range Height number)
           (domain Height Spindles)))

(define-instance
   SampleConvSpindle (Spindles))
(define-instance
   LampDiskSpindle (Spindles))
```

With this ontology, the AAMs can converse about the height of spindles. For example, the Diagnostic AAM might send the following message to the Modeling AAM:

```
(ask-one :receiver MODEL
 :content (Height LampDiskSpindle ?h)
 :reply-with rid)
```

which might then reply

```
(reply :receiver DIAGNOSTIC
 :content (Height LampDiskSpindle 70)
 :in-reply-to rid)
```

In these messages `ask-one` and `reply` are two KQML performatives; `:receiver`, `:content`, `:reply-with` and `:in-reply-to` label the message arguments. The language of the content of each message is KIF (the default language of all messages). The meaning of the terms in the content of each message is defined by the ontology.

Following the approach discussed in section 3.3, the AMF ontology has been automatically translated into the data structures used by the Diagnostic AAM. This AAM is written in CLOS, so the translation is from Ontolingua to CLOS.

5 Conclusion

The paper has discussed the problems of implementing complex mission operations systems and has described a two-fold approach to their solution: build generic applications and adopt a standard integration framework.

We have not mentioned here the possible overlap of with other integration technologies such as CORBA [4]. They could provide a basis for the more advanced ATOS features.

Much has been achieved since the initiation of the ATOS programme in 1992. We also believe that the approach we have adopted may be effective not only for the space industry but for any industry which needs to integrate applications to build complex systems.

6 Acknowledgments

This work was funded by ESA's Advanced Systems and Technology Programme (ASTP) which is managed by ESA's Directorate of Telecommunications. ASTP promotes the development of new technologies for the space domain, and supports European industry in implementing these in the form of marketable products.

The authors also acknowledge the contributions of Howard Smith (Logica UK) and Herwig Laue (ESOC) to the ATOS programme.

7 Bibliography

[1] Genesereth M.; *An Agent Based Framework for Software Interoperability*, Software Technology Conference '92, Los Angeles, April 1992.

[2] Gruber T., Ontolingua: *A Mechanism to Support Portable Ontologies*, Knowledge System Laboratory Technical Report, Stanford University, June 1992.

[3] Laue H., Kaufeler J-F, Poulter K., Smith H.; *The Advanced Technology Operations System ATOS*; Proc. 2nd Int. Sym. Ground Data Systems for Space Mission Operations. SPACEOPS 1992 Pasadena, California, USA. JPL Publication 93-5.

[4] Object Management Group; *The Common Object Request Broker: Architecture and Specification;* OMG Document number 91.12.1 10 December 1991.

[5] Patil R. et al; *The DARPA Knowledge Sharing Effort: Progress Report*, Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation, Cambridge MA, Morgan Kaufman, 1992.

[6] Poulter K. J., Smith H. N.; *ATOS-1: Designing the Infrastructure for an Advanced Spacecraft Operations*; Proc. 2nd Int. Sym. Ground Data Systems for Space Mission Operations. SPACEOPS 1992 Pasadena, California, USA. JPL Publication 93-5.

[7] Wheadon J.;*AMFESYS: Modelling and Diagnosis Functions for Operations Support.*; Proc. 2nd Int. Sym. Ground Data Systems for Space Mission Operations. SPACEOPS 1992 Pasadena, California, USA. JPL Publication 93.

N95-17603

111188

354318

P. 7

The NASA Mission Operations and Control Architecture Program

Paul J. Ondrus
Head, Missions Operations Systems Office
Mission Operations Division
Goddard Space Flight Center

Richard D. Carper
Senior Engineer
Science Applications International, Inc.

Alan J. Jeffries
Systems Engineer
Science Applications International, Inc.

Abstract

The conflict between increases in space mission complexity and rapidly declining space mission budgets has created strong pressures to radically reduce the costs of designing and operating spacecraft. A key approach to achieving such reductions is through reducing the development and operations costs of the supporting mission operations systems.

One of the efforts which the Communications and Data Systems Division at NASA Headquarters is using to meet this challenge is the Mission Operations Control Architecture (MOCA) project. Technical direction of this effort has been delegated to the Mission Operations Division (MOD) of the Goddard Space Flight Center (GSFC).

MOCA is to develop a mission control and data acquisition architecture, and supporting standards, to guide the development of future spacecraft and mission control facilities at GSFC. The architecture will reduce the need for around-the-clock operations staffing, obtain a high level of reuse of flight and ground software elements from mission to mission, and increase overall system flexibility by enabling the migration of appropriate functions from the ground to the spacecraft.

The end results are to be an established way of designing the spacecraft-ground system interface for GSFC's in-house developed spacecraft, and a specification of the end to end spacecraft control process, including data structures, interfaces, and protocols, suitable for inclusion in solicitation documents for future flight spacecraft. A flight software kernel may be developed and maintained in a condition that it can be offered as Government Furnished Equipment in solicitations.

This paper describes the MOCA project, its current status, and the results to date.

Introduction

Most current spacecraft are extensively supervised from the ground, and spacecraft command and control systems have been re-invented by almost every new flight mission. This seriously affects ground systems reusability, and therefore costs for systems development, training, software maintenance, and sharing of operators among projects. This traditional approach is in serious conflict with the realities of declining space mission budgets.

The Communications and Data Systems Division at NASA Headquarters, through the Mission Operations Division (MOD) of the Goddard Space Flight Center (GSFC), is addressing this problem by sponsoring the Mission Operations Control Architecture (MOCA) project. The objective of this program is to develop a spacecraft control and data acquisition architecture which will guide the development of future spacecraft and mission control facilities. The architecture is intended to reduce the need for around-the-clock staffing of operations control centers (partly by increasing spacecraft autonomy), enable a high level of reuse of both flight and ground software from mission to mission, and allow the allocation and migration of functions between ground and spacecraft missions as is appropriate for a given mission requirements set.

MOCA is using a three pronged approach: deep involvement of the ultimate implementing and operating community at GSFC; analysis of current mission operations systems, leading to a redefinition and standardization of architecture; and a survey and assessment of available technologies, subsystems, and commercially available products, with analysis of how to make it all fit together.

Organization and Process

In order to provide a broad base of knowledge and to enhance the ease of acceptance of results, the MOCA project is being conducted by the MOD as a cooperative effort among itself, the GSFC Flight Projects Directorate, and the GSFC Engineering Directorate. The latter is the GSFC's flight systems engineering organization. The organizational tools used to implement this cooperative structure are an ad hoc MOCA Steering Group, with members from management from NASA Headquarters and from each of the three directorates, and a MOCA Users Forum, which is constituted primarily of selected, experienced, engineering level persons from each organization.

MOCA is divided into two phases, the Exploratory Phase (which began in February, 1994) and the System Design Phase. Each phase will last from one year to eighteen months, as required. The Exploratory Phase is a rapid but in-depth survey of the complexity and scope of the problem and an examination of potential solutions. The System Design phase will both develop and deploy the new capabilities required for the system.

When agreement on the architecture is achieved, one or more spacecraft will be selected to use as a prototype to finalize and prove the data structures, protocols, and interfaces between modules defined by the architecture. Ultimately, flight software elements and corresponding ground control modules will be developed, maintained, and configuration-controlled by an inter-directorate team. Therefore, the MOCA is an architecture, a set of interface definitions, supporting protocols and application layer languages, that enable the standardized commanding and supervision of remote space vehicles.

As this work progresses, it will be presented to the American Institute of Aeronautics and Astronautics (AIAA) Spacecraft Control Working Group. It is hoped that a NASA or U. S. Government agreement on an architecture for spacecraft control and a suite of supporting standards will result through this channel. However, the MOCA project focuses on the needs of GSFC specifically.

Approach

The aim of MOCA is to substantially reduce the end-to-end life cycle cost of future flight programs by radically reducing ground operations costs, including development costs.

MOCA disputes the contention that "cheap programs mean dumb spacecraft". Instead, MOCA asserts that when the end-to-end life cycle costs of a program are considered, "cheap programs need smart spacecraft". MOCA further contends that the technology is currently available to have smart spacecraft at very little increase in development cost, and that in fact most of the basic enabling technologies (for example, increased computational power, increased memory, large solid state data storage) are already in flight use. And that therefore what is required to achieve the mission operations cost reduction objectives are the development and implementation of the necessary operations concepts, architecture, and standards.

Preliminary Functional Architecture

The following is very preliminary, and will undoubtedly undergo major changes as the MOCA project matures.

The context of MOCA is "Mission Operations Functions", as shown in Figure 1. Therefore the figure shows the external interfaces to MOCA. There are two fundamental points made by the figure. First, it is important to note that mission operations functions are the domain, regardless of whether the functions are performed on the spacecraft or on the ground. Second, and equally important, flight subsystems and ground supporting subsystems are not in the MOCA domain, but the interfaces with them (and therefore the relevant functions performed by them) are.

The primary driver of mission operations is the science planning entity which provides both strategic planning information (the science plan) and part of the detailed or tactical planning inputs (instrument commands). These inputs are provided in cycles of various time intervals.

The MOCA functions and processes use these inputs to plan and schedule resources, coordinate the execution of the plan across the resources, monitor and assess the status of the resources, and feedback lessons learned into the process for the next cycle. Since the MOCA functions operate in this cyclic manner, the architecture described in this paper decomposes the MOCA functional architecture based on this planning-execution- assessment nature of mission operations. Figure 2 depicts the three functions which make up the first level of

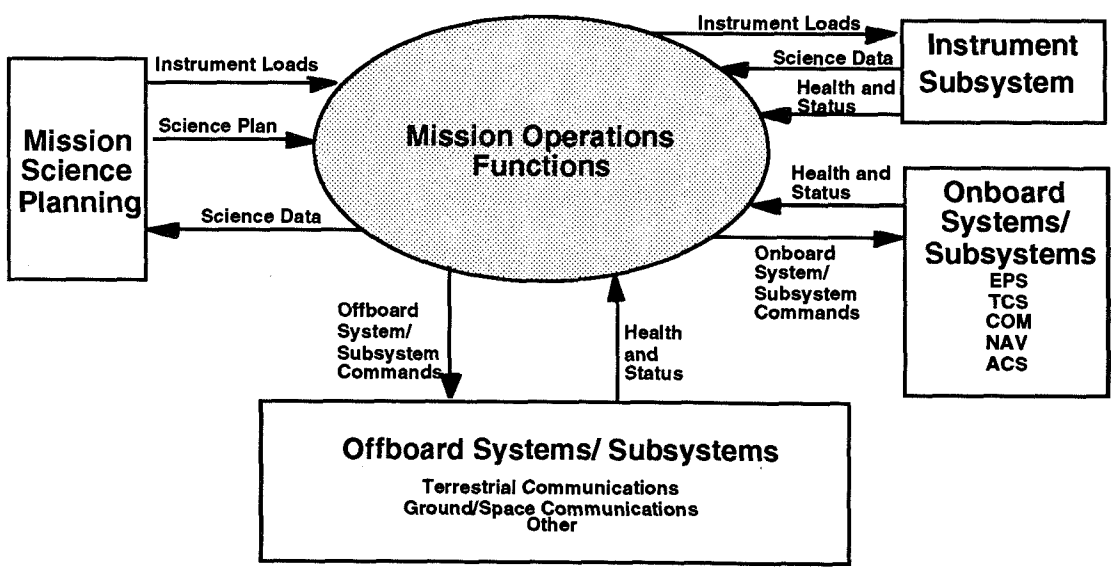


Figure 1: MOCA External Interfaces

the MOCA functional architecture. Also shown in Figure 2 are two entities utilized by all three functions: the Mission Model and the Mission Database.

The Mission Model constitutes an accurate representation of all the resources the MOCA functions have visibility into and interaction with. The Mission Database is a repository of actual data points either used or generated by the mission model and MOCA functions. All three of the primary MOCA functions use these resources but in unique and different ways. For instance, the Planning and Scheduling function uses the Mission Model to predict the events and actions of resources for the next cycle. The Mission Command and Control function uses the Mission Model to compare the real time events and actions of resources against the predicted events and actions to ensure operations are proceeding as planned and within

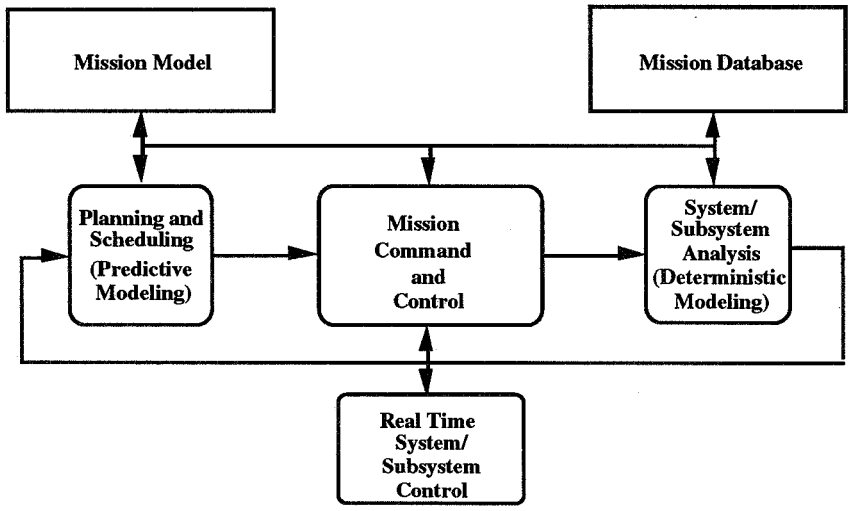


Figure 2: First Level MOCA Functions

tolerances. The System/Subsystem Analysis function uses the mission model to determine why events and actions did not perform as predicted and to provide feedback into the model as resources degrade or change over the life of the mission.

Figures 3 through 5 show the next level of decomposition for the three primary MOCA functions. This paper will not go into detail on these lower level representations except to note that the Scheduling and Planning and the System/Subsystem Analysis functions have been further decomposed based on short term and long term processes.

Preliminary Target Characteristics

A preliminary analysis of current mission operations has lead the MOCA to identify the following as highly desirable characteristics which should be included in the MOCA concept of operations, and enabled by the MOCA architecture. These are very early ideas, and will undoubtedly be subject to significant modifications, expansions, and deletions as the project progresses.

It appears highly desirable to minimize the number of contacts between a spacecraft and the ground, as is feasible within the constraints of mission safety and mission performance. The planning, scheduling, initiation, conduct, and termination of a space/ground contact is expensive in itself, and the cost is much more sensitive to the number of contacts than to duration or data rates. This minimization should be accomplished by making spacecraft more autonomous than at present. The feasibility and acceptability of increased autonomy should be realized by designing the process of achieving autonomy to reduce risk, minimize life cycle costs, and maintain flexible control of the process by project management. The process should include the development of ground based backup

capability to onboard functions, and by achieving the autonomy via function migration from ground to space as operational experience is gained.

Spacecraft should be made to look operationally as much alike as possible. Through the use of interface, format, and procedural standards to implement a "virtual spacecraft" concept, spacecraft should be made to appear to the ground systems as operationally identical as is

example of such existing standards are the tailored communications standards that can be adopted from other non-MOCA sources (e.g. Consultative Committee for Space Data Systems (CCSDS), Space Communications Protocol Standards group (SCPS)). Other standards, such as for the operations functions (i.e. at the Application Layer) will be selected by or developed within MOCA.

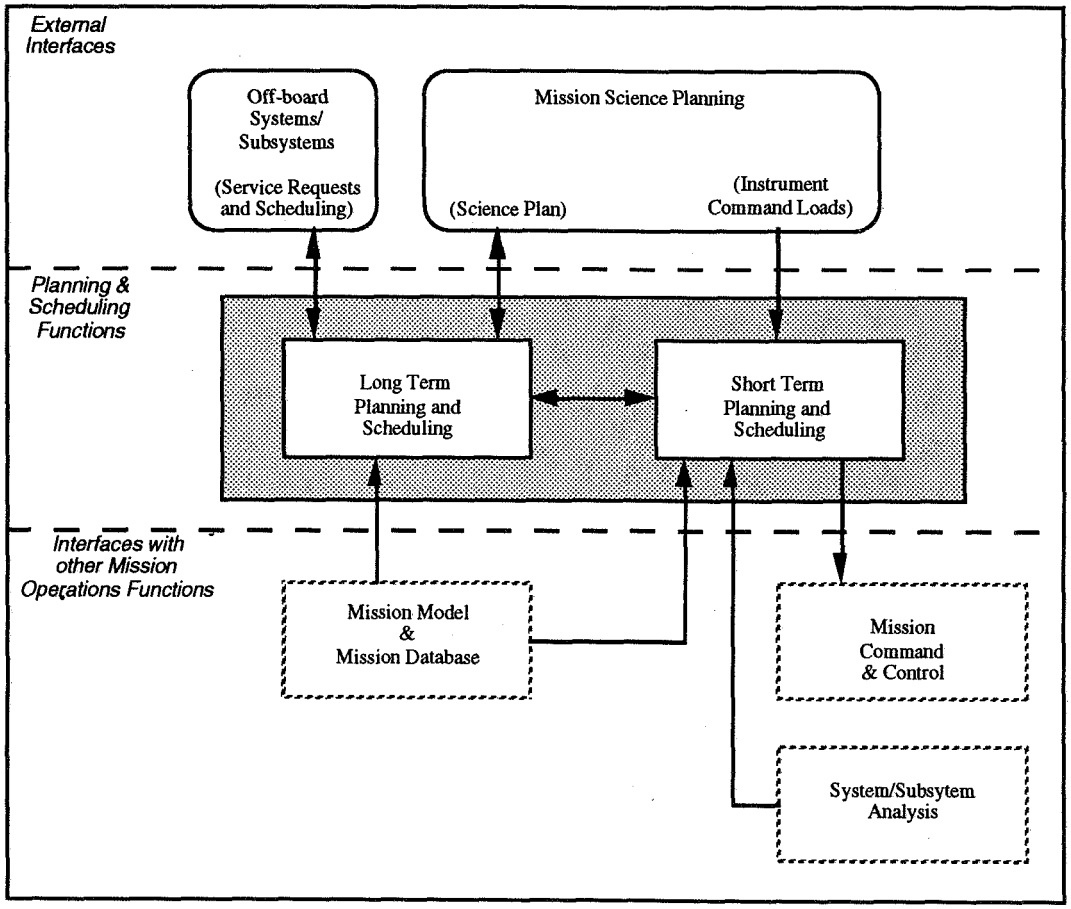


Figure 3: Functional Decomposition of the MOCA Planning and Scheduling Function

feasible. This will eliminate large parts of development and training costs, allow operations crews to be shared among several spacecraft, and increase the reliability of operations.

Standards should define all operational interfaces. Standards should be selected, adapted, or, as necessary, developed and emplaced at all operational interfaces. An

The Standards should be used across different projects. To achieve the above targets, the same standards must be used for each flight project. This approach minimizes the non-recurring ground system development and modification costs as well as substantially reducing recurring mission operations costs.

Implementation of the MOCA concepts, architecture, and standards should be accomplished to the maximum extent possible through the use of existing standards, existing technologies, work accomplished by other similar NASA and Department of Defense activities, commercial off-the-shelf products, and through use of existing testbeds and flight opportunities for proof of concept and validation. Major redesign efforts and all new development for control facilities at GSFC should be accomplished in conformance with the MOCA standards.

The Future

Although MOCA is still in an early phase, several key concepts are beginning to emerge which appear to be technically feasible and economically desirable. Among these are: communications between ground systems and spacecraft by an intermediate or high level process control language, rather than by commands and telemetry; on-demand

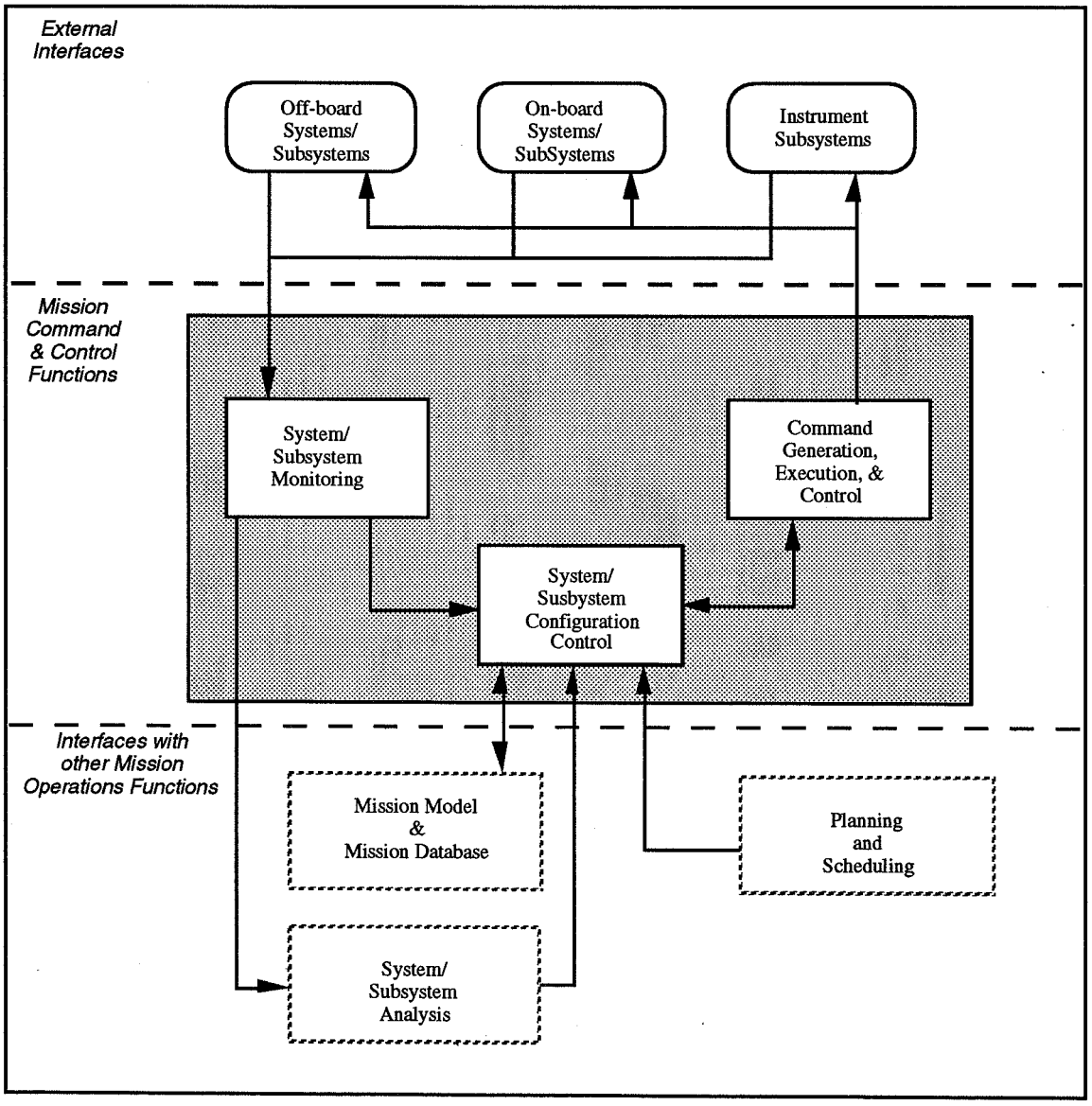


Figure 4: Functional Decomposition of the MOCA Mission Command and Control Function

space/ground communications *by spacecraft demand*; and eventually a reversal of current roles in that a spacecraft may view its supporting ground systems as a collection of on-call resources to help it meet its mission objectives.

It appears at this time that there are no insuperable technical or cost hurdles to achieving greatly decreased end-to-end life-cycle mission operations costs through the techniques of increased spacecraft autonomy, appropriate standards for critical operations interfaces, and standard protocols, all structured by a common mission operations architecture.

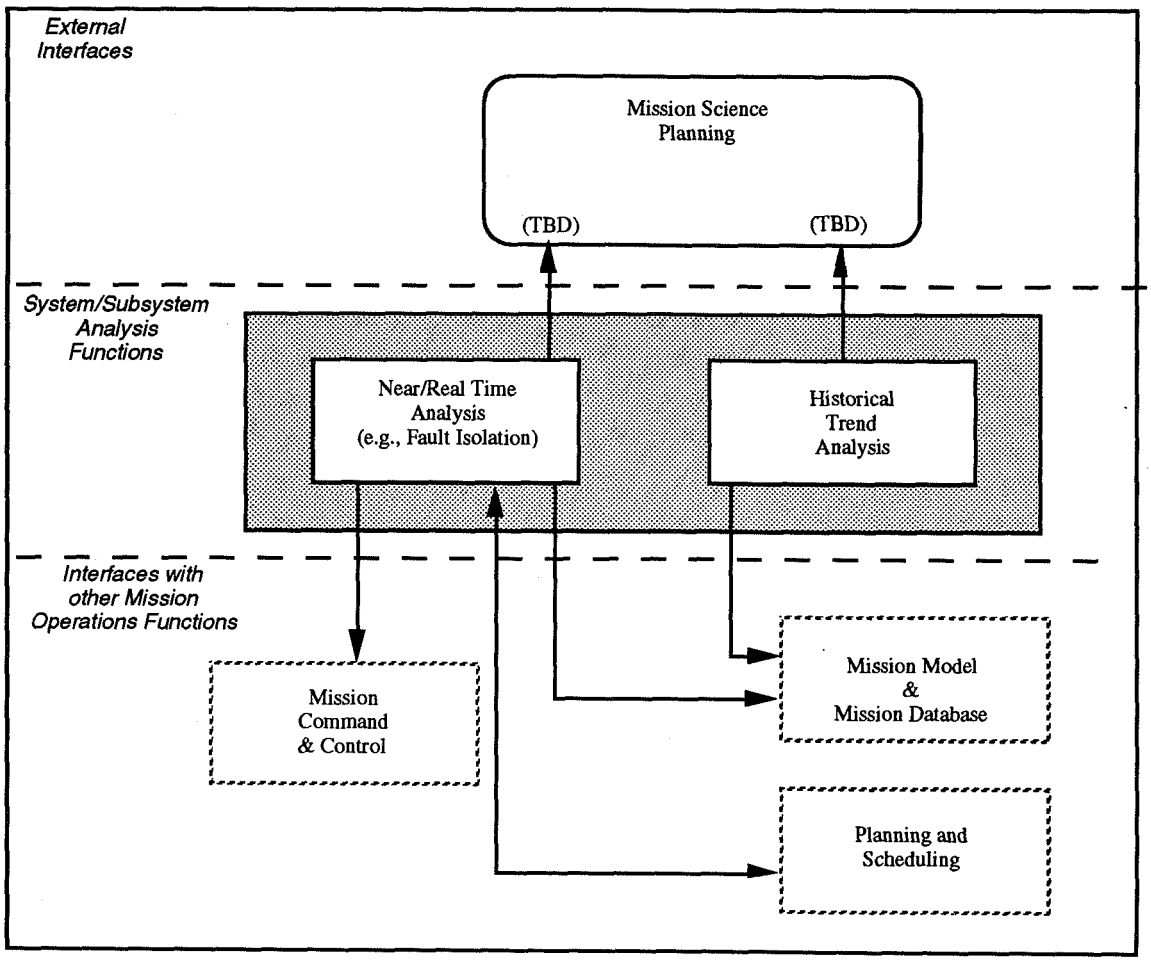


Figure 5: Functional Decomposition of the MOCA System/Subsystem Analysis Function

111189 N95-17604

354319 p. 12

Renaissance Architecture for Ground Data Systems

Dorothy C. Perkins
Mission Operations Division
Mission Operations and Data Systems Directorate
NASA/Goddard Space Flight Center

Lawrence B. Zeigenfuss
Systems Engineering Office
Mission Operations and Data Systems Directorate
NASA/Goddard Space Flight Center

ABSTRACT

The Mission Operations and Data Systems Directorate (MO&DSD) has embarked on a new approach for developing and operating Ground Data Systems (GDS) for flight mission support. This approach is driven by the goals of minimizing cost and maximizing customer satisfaction. Achievement of these goals is realized through the use of a standard set of capabilities which can be modified to meet specific user needs. This approach, which is called the Renaissance architecture, stresses the engineering of integrated systems, based upon workstation/local area network (LAN)/fileserver technology and reusable hardware and software components called "building blocks." These building blocks are integrated with mission specific capabilities to build the GDS for each individual mission. The building block approach is key to the reduction of development costs and schedules. Also, the Renaissance approach allows the integration of GDS functions that were previously provided via separate multi-mission facilities. With the Renaissance architecture, the GDS can be developed and operated by the MO&DSD or all, or part, of the GDS can be operated by the user at their facility. Flexibility in operation configuration allows both selection of a cost-effective operations approach and the capability for customizing operations to user needs. Thus the focus of the MO&DSD is shifted from operating systems that we have built to building systems and, optionally, operations as separate services.

Renaissance is actually a continuous process. Both the building blocks and the system architecture will evolve as user needs and technology change. Providing GDS on a per user basis enables this continuous refinement of the development process and product and allows the MO&DSD to remain a customer-focused organization. This paper will present the activities and results of the MO&DSD initial efforts toward the establishment of the Renaissance approach for the development of GDS, with a particular focus on both the technical and process implications posed by Renaissance to the MO&DSD.

INTRODUCTION

The MO&DSD provides end-to-end mission support for National Aeronautics and Space Administration (NASA) low earth orbit scientific space flight projects. This support ranges from establishing the radio frequency (RF) link with the user spacecraft for data acquisition, tracking and spacecraft commanding to distribution of captured instrument data to scientific investigators. In meeting its charter over the last three decades, the MO&DSD developed significant expertise within its organizational elements in building and operating systems to meet requirements in these functional areas. As an example, flight dynamics support is provided by one MO&DSD Division. That Division builds and operates institutional, multi-mission systems to support flight missions. Other Divisions are responsible for other areas of support. In general, Division systems were

PAGE 1304 INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

housed in multi-mission facilities and based on large mainframe computer architectures, to provide efficient and cost-effective ground support operations for missions.

These MO&DSD systems and services reflect a technology environment where large computers provided the only viable system solutions, and a science environment that often advocated large and complex science objectives and correspondingly complex spacecraft. But now both the nature of the users and the technology environment have changed significantly. NASA science programs have embraced the "faster, better, cheaper" philosophy as a means of survival in the present fiscally constrained environment. Smaller spacecraft are being built with substantially reduced budgets and development schedules from those of their predecessors. At the same time, the modern computer technology trend is embodied in small, powerful workstations connected via a network in appropriate configurations to meet specific processing needs. This combination of smaller missions and flexible technology has created enormous opportunity for users and providers to find innovative ways of doing business.

BACKGROUND

Drivers for change within MO&DSD have come from numerous sources, both external and internal to the MO&DSD. Acknowledgment of these led the MO&DSD to actively and collectively seek new ways of serving its customers' needs. The drivers and initial analysis activities targeted at addressing the consequent challenges are discussed.

External Drivers for Change

The nature of the newer scientific missions and the prevailing technology have led to a desire and ability on the users part not just to receive data, but to actually operate all or part of the ground data system in order to meet the objectives of their missions. The MO&DSD past approach to mission operations with shared institutional systems does not possess the flexibility to meet such changes in customer needs. Chiefly, because this approach ties solutions for all users to a common technology, it also does not possess the resiliency to implement cheaper, streamlined systems for mission support where these are appropriate. In addition, customers have felt that dealing with several separate multi-mission facilities added complexity in dealing with MO&DSD for mission support. This is especially true for the smaller and simpler space flight projects that tended toward a more consolidated approach for mission and science planning and operations. Finally, with the dramatic reduction in the cost of computing power resulting from the evolution of data processing technology, the previous cost advantage of the multi-mission approach based upon mainframe computer architectures has evaporated. Flight project customers perceive the MO&DSD past approach to mission operations as not being the most cost-effective.

Internal Drivers for Change

Recent downsizings of mission budgets coupled with the availability of rapid technology advancements have led the organizational elements within MO&DSD to seek alternate solutions for development and operation of ground data systems within their functional areas. Utilization of workstation/LAN architectures, formalized software reuse, and adoption of commercial standards have all been successfully demonstrated within the Divisions for several years. For example, a paper published in the SPACEOPS 92 Proceedings entitled "SAMPEX Payload Operations Control Center Implementation" described the first development of a Payload Operations Control Center (POCC) based upon the Transportable POCC (TPOCC) architecture. These technology innovations have been beneficial and demonstrated significant cost savings. However, before Renaissance they remained generally localized within the various MO&DSD facilities. While there were benefits that accrued from collaboration across Division facilities, they had not yet gained supremacy as a standard way of accomplishing the MO&DSD mission.

Architecture Analysis Activities

Recognizing these drivers, the MO&DSD initiated activities to explore new, consolidated approaches for development and operation of GDS for flight mission support. The goals of minimizing cost, maximizing flexibility for meeting customer requirements, and reducing complexity were established for this effort. A study was commissioned to identify GDS architecture approaches that offer significant reductions in cost and development schedules as well as increased flexibility for meeting individual customer requirements in establishing mission support capabilities. The report recommended an architecture approach that addressed implementations from simple to complex missions through integration of support functions in a mission-specific instantiation. Implementations would employ reuse over multiple missions and incorporate effective standards for commercial product inclusion

An Architectural Steering Group (ASG) was established to evaluate the recommendations from the architecture study report and to determine if the scope of the study should be expanded to address space-to-ground and ground-to-ground communications mission support functions. The ASG also chose to commission two additional studies: to look at current and future Directorate operations concepts to assure that operations as well as development improvements would be realized in the new architecture. The ASG activities ultimately led to the identification of an architecture approach that stressed the engineering of integrated systems that encompassed the MO&DSD mission support functions of flight dynamics, spacecraft command and control, and data capture and distribution. These systems would be based on workstation/LAN/fileserver technology and reusable hardware and software components called building blocks. The name "Renaissance" was applied to this architecture approach. ("Renaissance" is an acronym that stands for Reusable Network Architecture for Interoperable Space Science, Analysis, Navigation, and Control Environments.) The ASG also determined that the institutional nature of communication services should not be changed, though in fact the technology of implementing these services will be improved as Renaissance evolves.

THE RENAISSANCE CONCEPT

The MO&DSD has embraced change through Renaissance on two fronts. First, is willing to view itself as a provider of both systems and services, rather than primarily a provider of services. Secondly, is the espousal of particular objectives to facilitate achievement of Renaissance.

MO&DSD Renaissance Services

Renaissance divides the domain of MO&DSD mission support capabilities into three areas: mission operations, science operations, and centers of expertise (see Figure 1).

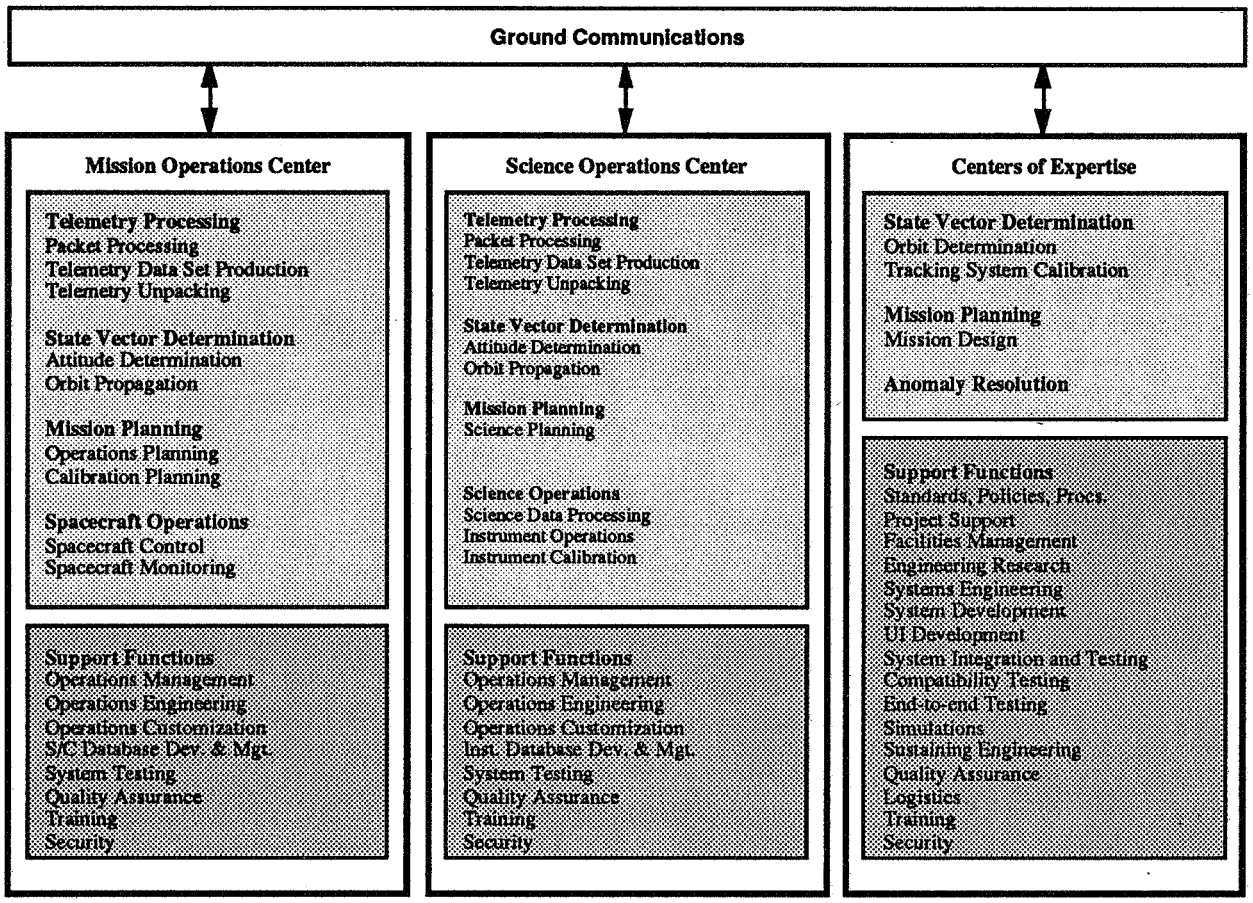


Figure 1. Renaissance Mission Support Domain

Mission operations are those functions that are integrated to operate the spacecraft to meet the specific requirements for that mission. Similarly, science operations are integrated to achieve the science objectives of a specific mission. The functional elements associated with mission and science operations are physically embodied within a Mission Operations Center (MOC) and a Science Operations Center (SOC), respectively, for each flight mission. Under the Renaissance concept, the MOC and the SOC could be integrated into a single center, collocated in a shared facility, or geographically dispersed. There is no requirement that either be located at the Goddard Space Flight Center (GSFC).

The third area of the MO&DSD domain, the Centers of Expertise (COE), comprise a permanent institutional infrastructure that contains the resources to support the set of flight missions over their entire life cycle. Obviously, the key resource within the COE is the personnel with the technical skills and experience in the development and operation of GDS. The balance of the COE resources include the tools, materials, and processes that are applied by the personnel to develop and operate the mission support functions.

Renaissance Objectives

The Renaissance architecture approach espouses three major objectives for achieving the goals of cost-effectiveness, flexibility, and simplicity for providing mission operations support to space flight project customers. Firstly, to assure that developed systems permit integrated operations in a stand-alone environment for unique mission support. Development and operation of mission-

specific systems provides maximum flexibility for customizing the GDS to meet the user's particular requirements. The integration of mission support functions for command and control, flight dynamics and data processing also presents an opportunity for reducing the cost of ground system development through the elimination of redundant functions that had been replicated within each of the multi-mission systems, (e.g., telemetry unpacking). Developments, however, would not preclude recombining of functions into multi-mission facilities if this should prove cost-effective.

The second objective inherent with Renaissance is reuse of support capabilities. Here reuse means a systematic, planned approach for developing reusable components of ground data systems rather than reuse on an ad hoc basis. It implies well-defined interfaces and use of standards to implement systems, and an ability to insert new technology readily over time. The construction and use of these reusable components, or "building blocks", is the key to reducing the cost of ground system development. MO&DSD established a Renaissance Project Team to define those "building blocks".

The third significant objective associated with Renaissance is the projectized approach to GDS development and operation which is aimed at reducing the complexity associated with the present user interfaces. Previous mission support systems have of course been coordinated among MO&DSD Divisions, but subsystems were implemented in separate facilities and not functionally integrated. The new approach calls for completely integrated requirements and integrated testing. Establishment of a mission team, led by a Ground System Project Manager (GSPM), provides a focal point within the MO&DSD for matters relating to the development and operation of MO&DSD GDS. The team defines the mission system, and integrates reusable Renaissance "building blocks" with mission-unique building blocks that it develops. The team also assures that space flight project customer needs receive strong advocacy within the MO&DSD.

APPROACH TO ACHIEVING RENAISSANCE

The ASG selected the Advanced Composition Explorer (ACE) mission for the initial implementation of a Renaissance GDS. The mid-97 launch date was close enough to provide for a relatively quick demonstration of Renaissance without incurring the risks related to interruption of system developments that were already well under way [e.g., X-Ray Timing Explorer (XTE) and Tropical Rainfall Measuring Mission (TRMM)].

The challenge faced by the MO&DSD teams was to achieve the modular "building block" goals of Renaissance, while simultaneously meeting the near-term mission needs of the ACE Mission. Schedules, climate and budget would not allow for an extended period of time to prototype Renaissance concepts before instantiating them in a mission. Building blocks could not be created first, followed by mission-unique pieces. Parallel paths and integrated planning were required to achieve the ACE Mission.

However, this challenge was not as daunting as it might appear. Two factors created a climate for success: tight integration of the Renaissance and ACE implementation teams; and extensive availability of predecessor systems that meet Renaissance goals.

Implementation Teams

The Renaissance Project Team, a core group of highly capable engineers, was charged with Renaissance building block definition. Their charter was as follows:

- Identify building blocks through examination of ACE and other system (e.g., XTE) requirements to determine generic functionality.
- Identify predecessor systems and Commercial Off-the-Shelf (COTS) products that could meet the building block specifications.
- Identify standards and development processes useful in the Renaissance era.
- Develop plans for transitioning into the Renaissance approach.
- Work with the mission teams, initially the ACE team, to apply the Renaissance architecture.

This core team was augmented by additional MO&DSD engineers who served to provide input to their efforts and to review and critique the outcomes. In particular, those engineers charged with implementing the ACE Mission were early and constant participants in the Renaissance effort.

Predecessor Systems

Despite the use of the Renaissance name to capture a system concept within the MO&DSD, many predecessor efforts on other missions were aligned with the Renaissance concepts, and influential in molding its goals and giving confidence that the architecture would be successful. The Renaissance concept, in fact, was merely an acceleration and consolidation of various efforts that were already occurring naturally within the Directorate. Reliance on these efforts gave credence to the possibility of meeting both ACE and Renaissance goals within the aggressive schedule. Examples of forerunner efforts include:

- The TPOCC UNIX-based software that supports real-time spacecraft command and control and that, prior to Renaissance, was in use or planned for use for the Small Explorer missions (SAMPEX, FAST, SWAS), ISTP WIND, POLAR and SOHO, XTE and TRMM. The TPOCC brings a legacy of substantial software reuse as well as workstation and LAN-based processing.
- The VLSI-based Level Zero Processor (LZP) employed on the FAST mission. This system captures spacecraft science data and removes transmission artifacts before forwarding it to science investigators. It is integrated into the mission command and control facility, and as such is a predecessor to the Renaissance operations approach.
- The Packet Processor (PACOR) II distributed system, a multi-mission data capture and level zero processing system planned for use by SWAS, TRMM, XTE, HST and GRO. PACOR illustrates both system (hardware and software) reuse and distributed processing.
- The Generic Support System (GSS), a reusable system for attitude determination.
- The Generic Spacecraft Analyst Assistant (GenSAA), a tool that allows spacecraft analysts to create graphics and rule-based systems to assist in monitoring spacecraft health and safety, and other decision-based situations.
- The Generic Trend Analysis System (GTAS), a reusable spacecraft trending tool.

All of these systems support in some measure the Renaissance objectives of reusability and integrated, stand-alone systems. Figures 2 and 3 capture the extent to which these goals are met in missions prior to ACE.

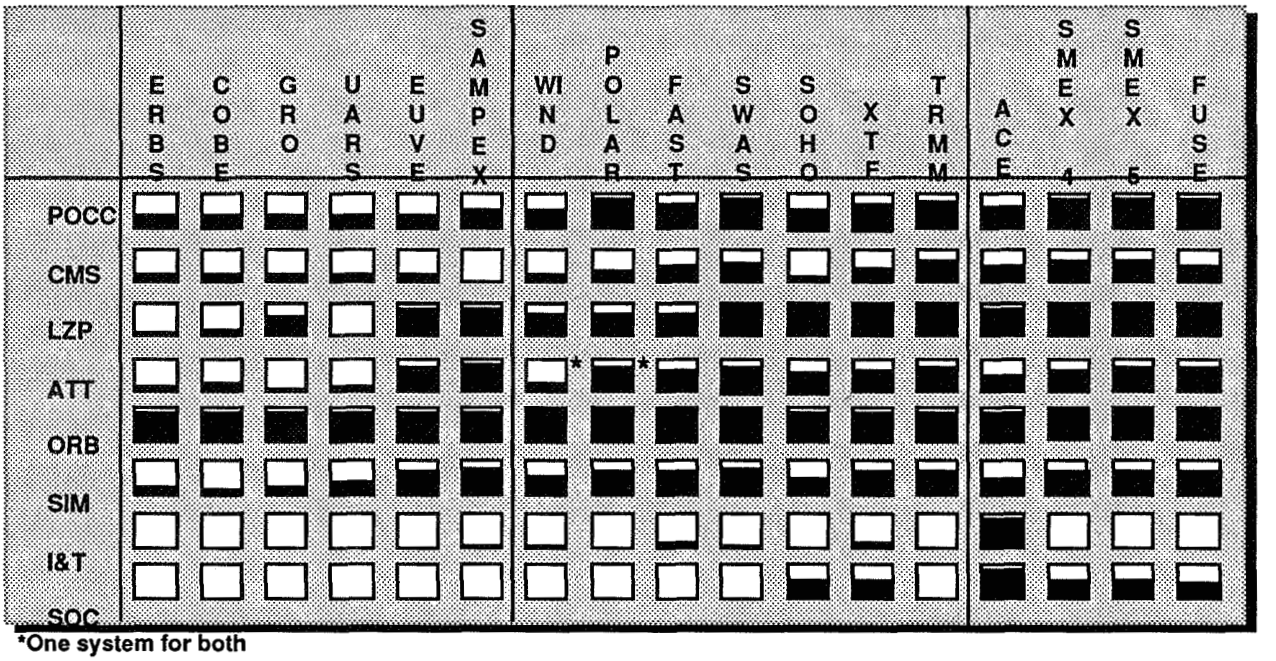


Figure 2. Percent Reuse

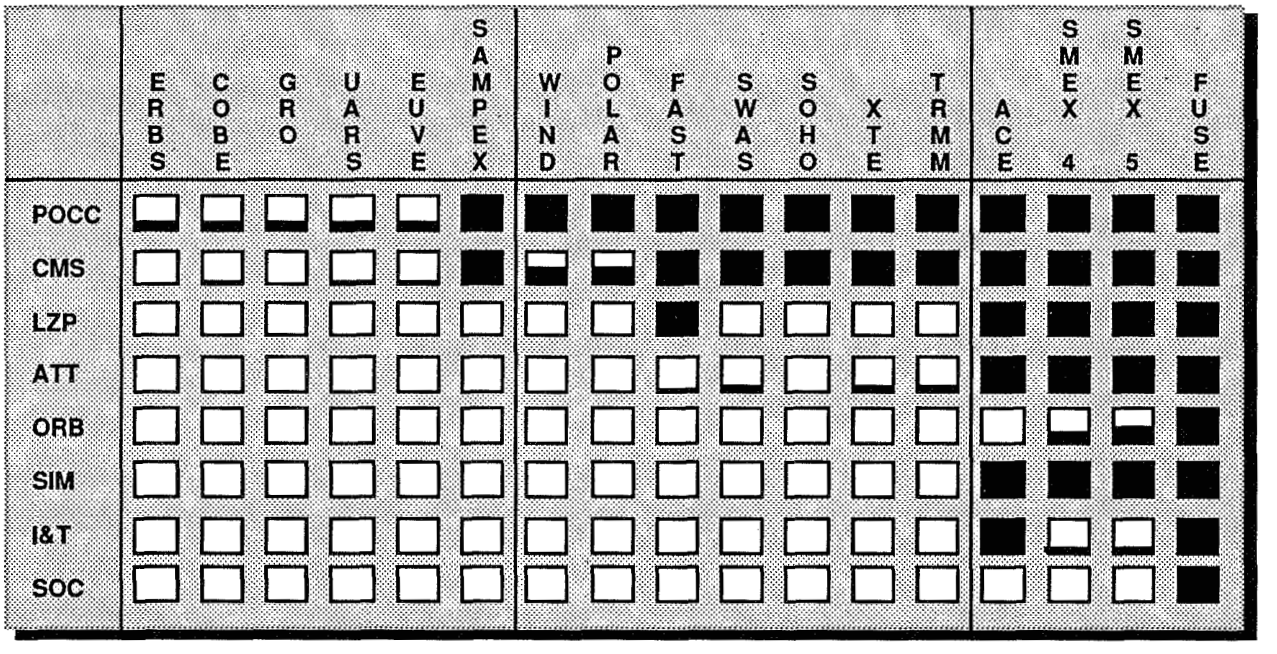


Figure 3. Percent Independent of Multi-Mission Facilities

ACHIEVEMENTS TO DATE

Early efforts have partitioned the Renaissance GDS into four sets of services. A Renaissance Project Team working group is defining functionality and building blocks for each set. The groupings are as follows:

- Spacecraft Communication Services (reconstruction of telemetry packets, command transmission, packet annotation, time correlation, and archiving).

- Spacecraft Data Distribution Services (real-time, quick-look and routine data delivery, data subsetting, output logging, and delivery validation).

- User Services (user interface, user tools and application builders, system configuration and monitoring, system security, system time synchronization, and data management).

- Application Services (spacecraft services such as telemetry monitoring, trend analysis, attitude support, command verification; planning and scheduling applications for spacecraft, science and network activity planning; and uplink applications such as real-time commanding and load generation).

Two additional working groups are assigned to issues that cross all Renaissance services:

- Architecture group, charged with defining the integrated architecture.

- Simulation and testing.

Many of the promised innovations of Renaissance are being realized within the context of the ACE ground system, including largely integrated operations, consolidation of functionality, incorporation of new technology and standards, and reliance on the legacy of past systems. Figure 4 illustrates the architecture proposed for ACE.

Integrated Operations

It is intended to incorporate real-time command and control, command and load generation, attitude determination and data capture and distribution within the ACE mission operations center. The only major MO&DSD ground system function that will still be treated within a separate facility for this mission is orbit determination and the ancillary production of maneuver planning aids. The ground station and SOC will remain separate from the MOC. (These facilities are not implemented by MO&DSD and are traditionally separated from MO&DSD facilities. Future directions will lead to the consolidation of MOC and SOC functions. See, for example, related paper in this conference, "A New Systems Engineering Approach to Streamlined Science and Mission Operations for the Far Ultraviolet Spectroscopic Explorer (FUSE)."

Consolidation of Functionality

In two significant arenas, functionality previously developed and performed within multiple facilities will be consolidated. Firstly, there will be a single front-end for frame synchronization, Reed-Solomon processing, virtual channel separation, and data quality annotation. This front-end will be located at the Deep Space Network (DSN) ground station (vs. former performance of portions of this function in three MO&DSD facilities). Data will be forwarded from there to the ACE MOC. This system will also allow data to be forwarded directly to the SOC for processing, if this proves desirable.

Secondly, a consolidated simulator that will meet the testing needs of all ground system functions is planned for development

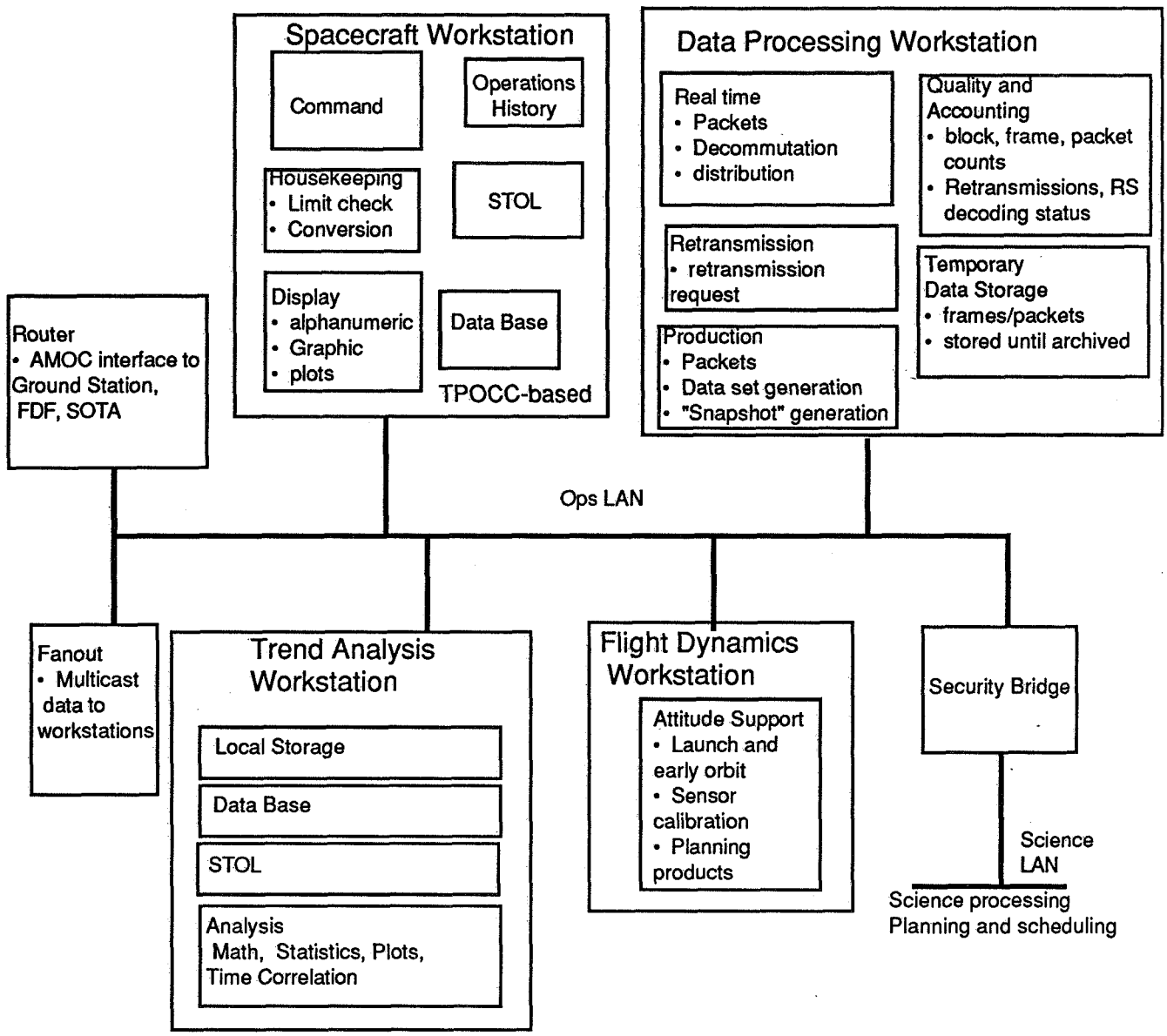


Figure 4. Renaissance Architecture for ACE MOC

Incorporation of Technology and Standards

A particular innovation within the ACE ground system is the use of the commercially available Transmission Control Protocol/Internet protocol (TCP/IP) protocol to transmit data from the ground station to the MOC. This allows replacement or encapsulation of the traditional 4800-bit NASA Communications (Nascom) blocks, and eliminates the need for custom systems to handle these blocks. Use of this protocol paves the way to ultimately reduce institutional cost for providing ground communications because of wide commercial availability and ability to eliminate Nascom blocks in the future.

The ACE ground data system will consist of a series of workstations supporting POSIX and communicating via a LAN. Software will be developed in ANSI C, C++ or Ada. While each workstation will have particular functionality assigned to it, the ability to move functions among workstations for load balancing or recovery from anomalies will be supported.

The ground system will also use X-windows and MOTIF as interface standards.

Predecessor Systems

The ACE ground system will depend heavily on building blocks, either implemented particularly to support ACE as the first mission, or derived from predecessor systems within MO&DSD. Table 1 lists ACE services and their derivation.

CONCLUSION

The consolidated Renaissance effort is less than a year old. Progress is substantial, however, due to an aggressive project team and integration of predecessor systems already aligned with Renaissance goals. Reliance on Renaissance products is allowing the MO&DSD to work with its customers in defining low-cost systems whose operations are tailored to the customers needs, for example, with the FUSE and the upcoming Small Explorer missions. Early results are promising, and the Directorate is committed to sustaining a management approach that will allow Renaissance goals to be achieved.

Table 1. Derivations of Renaissance Services for ACE

	Renaissance Building Blocks	Legacy Building Blocks	Mission Specific Building Blocks	Commerical Off-the-Shelf
User Services	•Interactive UI	•Event User Interface •Display •Report •STOL •HUD •Sim UI •LZP Ops •LZP QA	•MOC Init	•Window System •Network Mgmt
Application Services	•Mission Planning •Image Maintenance •RTADS •Commanding •Off-line ADS	•Gen Equation Processor •Maneuver Planning •GMT Sync •Contact Prediction •Config Monitor •State Manager	•Load Generation •Clock Correlation •Eqn Processor •Recorder Mgmt	
Data Services	•LZP Product Generation •LZP RT Proc •DSN Monitor Block Process •CMD Echo	•GMT Router Server •Event Logging •Telemetry Decom •Data Server •History	•Load Database	•File Server •DBMS
Space Comm	•Packet Services •Raw Data Logging		•Embedded. Frame Sync	•Comm Software •Multicast Server •Time Server

Nomenclature

ACE	Advanced Composition Explorer
ASG	Architectural Steering Group
COE	Centers of Expertise
COTS	Commercial Off-the-Shelf
DSN	Deep Space Network
FAST	Fast Auroral Snapshot Explorer
FUSE	Far Ultraviolet Spectroscopic Explorer
GDS	Ground Data Systems
GenSAA	Generic Spacecraft Analyst Assistant
GRO	Gamma Ray Observatory
GSFC	Goddard Space Flight Center
GSPM	Ground System Project Manager
GSS	Generic Support System
GTAS	Generic Trend Analysis System
HST	Hubble Space Telescope
ISTP	International Solar Terrestrial Physics
LAN	Local Area Network
LZP	Level Zero Processor
MO&DSD	Mission Operations and Data Systems Directorate
MOC	Mission Operations Center
NASA	National Aeronautics and Space Administration
Nascom	NASA Communication
PACOR	Packet Processor
POCC	Payload Operations Control Center
POLAR	Polar Plasma Laboratory
RENAISSANCE	Reusable Network Architecture for Interoperable Space Science
RF	Radio Frequency
SAMPEX	Solar Anomalous and Magnetospheric Explorer
SOC	Science Operations Center
SOHO	Solar Oscillator Heliospheric Observatory
SWAS	Submillimeter Wave Astronomy Satellite
TCP/IP	Transmission Control Protocol/Internet Protocol
TPOCC	Transportable Payload Operations Control Center
TRMM	Tropical Rainfall Measuring Mission
WIND	International Physics Laboratory
XTE	X-ray Timing Explorer

111190 N95-17605

354322 1.8

Architecture of a Distributed Multimission Operations System

Takahiro Yamada

The Institute of Space and Astronautical Science (ISAS)
3-1-1 Yoshinodai
Sagamihara 229, Japan

ABSTRACT

This paper presents an architecture to develop a multimission operations system, which we call DIOSA. In this architecture, a component used as a building block is called a functional block. Each functional block has a standard structure, and the interface between functional blocks are defined with a set of standard protocols. This paper shows the structure of the database used by functional blocks, the structure of interfaces between functional blocks, and the structure of system management. Finally, examples of typical functional bloks and an example of a system constructed with this architecture is shown.

Key Words: System architecture, Mission operations, spacecraft control

1. INTRODUCTION

In order to reduce the cost of developing an operations system for a spacecraft, an approch of developing a system by integrating reusable components has been proposed (Holder et al., 1992; Mandl et al., 1992). In order for such an approach to be successful, the function of the components must be defined in a structured model of the entire system, and the interfaces between components must be standardized.

This paper presents an architecture to develop a multilission operations system, which we call DIOSA (DIstributed Operations System Architecture). In this architecture, a component used as a building block is called a functional block. Each functional block has a standard structure, and the interfaces between functional blocks are defined with a set of standard protocols.

If the functions provided by a functional block can be customized by only changing parameters, the functional block can be utilized by many missions. The key to do this is the standardization of database. This paper shows an example of a structure of standard spacecraft database. To make a distributed system reliable, the interfaces between components must be simple and understandable. This paper presents a simple interface structure with which functional blocks can communicate with each other easily. Automating management activities is the key to reduce operational labor (Newsome et al., 1992). This paper proposes a scheme for managing a distributed operations system.

Finally, examples of typical functional blocks and an example of a system constructed with this architecture is shown.

2. SYSTEM ARCHITECTURE

2.1 Overall Architecture

To enable the definition of system components and interfaces of a distributed system, the architecture of the entire system needs to be defined. In this subsection, the definition of the concepts of system, complex, domain and functional block is given.

A spacecraft operations system consists of some complexes (Fig.1). A complex is an aggregated set of operations facilities located at one location. Typical examples of a complex are (1) a ground station, (2) a

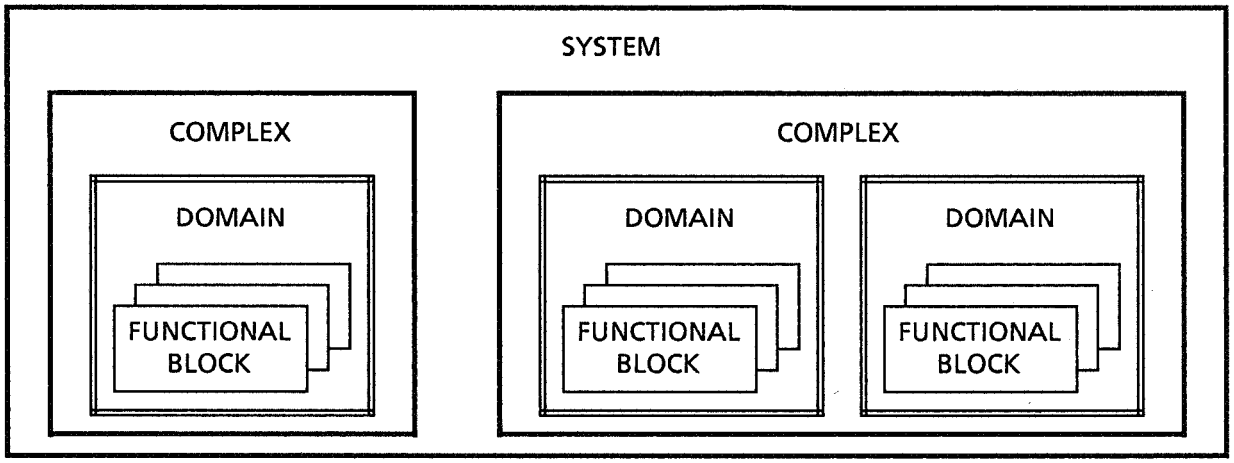


Fig. 1 DIOSA Overall Architecture

spacecraft operations center, and (3) a science analysis center.

A complex is divided into domains. A domain is a set of operations facilities managed as a single system. The system configuration of a domain is controlled independently from that of other domains. In other words, a configuration change of a domain does not affect the configuration of other domains. Typical examples of a domain are (1) a set of equipments for one antenna at a ground station, (2) a spacecraft control system for one spacecraft at a spacecraft operations center, and (3) a payload operations system for one payload at a science analysis center.

The distinction between complex and domain may not be important in some cases. For example, if an entire complex is managed as a single system, the notion of domain is useless. In what follows, however, we assume that the domain is the unit for management.

A domain consists of functional blocks. Functional blocks are basic building blocks of DIOSA. Each functional block performs a set of functions to operate a spacecraft, and has a standard structure which will be defined in the next subsection.

2.2 Structure of a Functional Block

Each functional block has (1) data ports, (2) a management port, and (3) a local SIB (Fig. 2).

The data ports are used for receiving data to be processed by the functional block and for sending data which has been processed by the functional block. The communications protocols to be used for the data ports are discussed in Section 4. The other end of a data port is another functional block.

The management port is used for receiving configuration control information and for sending status information (Newsome et al., 1992). The protocols to be used for the management port are discussed in Section 5. The other end of the management port is usually the Domain Management Functional Block.

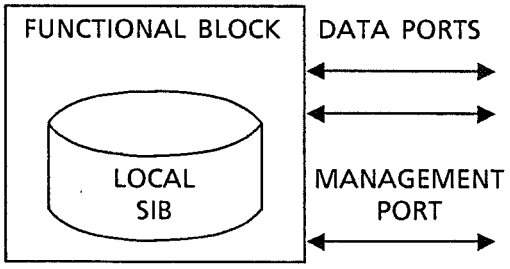


Fig. 2 Structure of Functional Block

The local SIB is a local copy of a subset of the Spacecraft Information Base (SIB) which will be discussed in Section 3. Each functional block retains its own copy of SIB, which is downloaded from the Master SIB of a spacecraft. Most of the basic functional blocks can be used for several spacecraft by changing the local SIB and (maybe) replacing some of the software modules.

3. SPACECRAFT INFORMATION BASE

3.1 Structure of SIB

The Spacecraft Information Base (SIB) is a database which stores all the information needed to operate a spacecraft. SIB consists of three parts, namely Data Definition Part, Behavior Definition Part, and Procedure Definition Part (Fig. 3). The Data Definition Part is equivalent to a traditional command and telemetry database found in most spacecraft operations systems. The Behavior Definition Part and Procedure Definition Part is an online version of the flight operations manual (Cipollone et al., 1992). In the future, higher-level knowledge on spacecraft should be further combined with SIB using a technique proposed by Kaufeler et al. (1992a).

Each part of SIB is generated from the spacecraft specifications and the flight operations manual. It is important that SIB should be used from the spacecraft integration and test phases by the spacecraft integration team and then handed to the flight operations team in an electronic manner.

In the operations system, SIB is maintained in the Master SIB Functional Block, and its appropriate portions are occasionally distributed to other functional blocks (Fig. 4).

3.2 Data Definition Part (SIB Part 1)

The Data Definition Part of SIB defines the structure of binary data contained in telemetry and command packets (or frames). For each command item, any information needed to generate a command packet from its mnemonic expression is stored (Fig. 5). And for each telemetry item, any information needed to extract its value or status from a telemetry packet is stored (Fig. 5). The parameters of the RF links of the spacecraft are also stored in this part.

A standard of packet formats like ESA's Packet Utilization Standard (Kaufeler et al., 1992b) greatly facilitates standardizing this part of SIB, thus increasing portability of SIB from mission to mission.

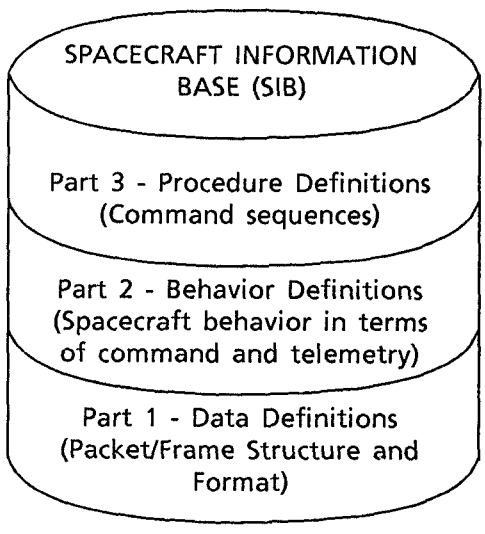


Fig. 3 Spacecraft Information Base (SIB)

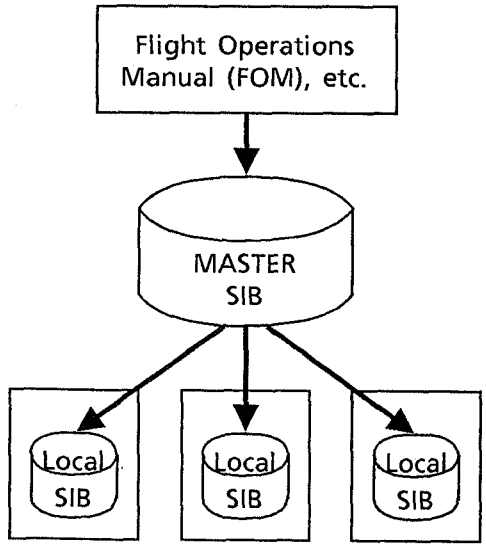


Fig. 4 Distribution of SIB

3.3 Behavior Definition Part (SIB Part 2)

This part defines the behavior of a spacecraft in terms of command and telemetry. This data is used to decide whether or not the spacecraft is acting normally, and to give the operator a message on what actions to be taken in case of an anomaly (Fig. 5).

Examples of information stored in this part are: (1) How the spacecraft reacts to each command in terms of telemetry, (2) Actions to be taken (or commands to be sent) when the spacecraft does not reacts to the transmitted command properly, (3) Telemetry limit values, (4) Actions to be taken (or commands to be sent) if a telemetry limit value is exceeded.

3.4 Procedure Definition Part (SIB Part 3)

This part stores command procedures. A command procedure is a sequence of commands to accomplish an objective. Other command sequences can be called as subprocedures in a command sequence. Information on resource requirements and operational constraints should be stored with each command sequence.

4. INTERFACE STRUCTURE

Standard interfaces between functional blocks are obtained by standardizing data formats and communications protocols. In this section, standard formats and protocols to be used for the data ports of functional blocks are presented.

4.1 Data Types

Data formats to be used for the data ports of functional blocks are standardized in two categories, namely Raw Data Type and Text Data Type (Fig. 6).

Examples of data of the Raw Data Type are command and telemetry packets (or frames) and radiometric data. To increase the portability of some software, any raw data used for spacecraft operations should be formatted in a data unit whose structure resembles that of CCSDS packets. In this way, for example, navigation data generated by a spacecraft and doppler data obtained at a ground station can be displayed on the same screen easily.

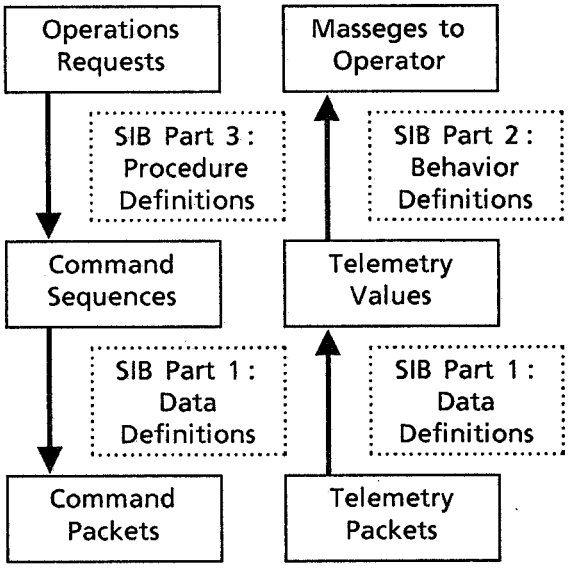


Fig. 5 Usage of SIB

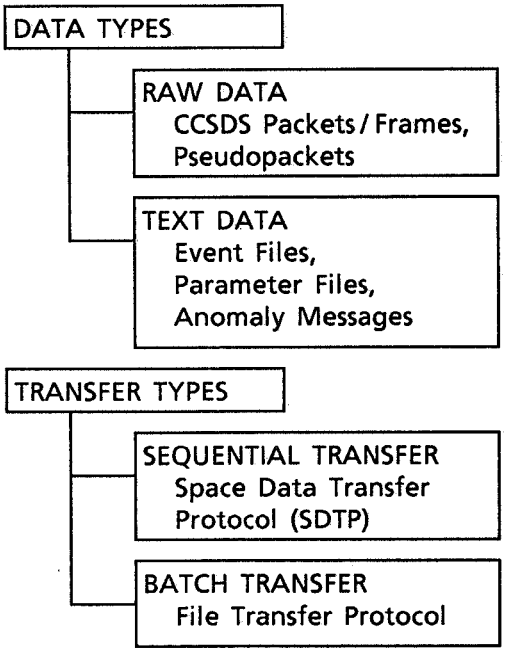


Fig. 6 Data Types and Transfer Types

To process raw packets, some ancillary information needs to be attached to each packets. The type of ancillary information depends on the application data. For telemetry packets, for example, reception time at the ground station and data quality information should be attached to each packet as ancillary information. In **DIOSA**, this information is called Application Specific Ancillary Information (ASAI).

Examples of data of the Text Data Type are event files (sequence of events and commands), parameter files (listing of parameters), anomaly messages (indicating the ocurrence of an anomaly), and state vectors. These kinds of data are transferred as text files with a standard format so that they can be processed with standard UNIX tools like AWK.

4.2 Transfer Types

Communications protocols to be used for the data ports of functional blocks are standardized in two categories, namely Sequential Transfer Type and Batch Transfer Type (Fig. 6).

The protocols of the Sequential Transfer Type are used for transferring data which needs to be transferred while it is being generated. These protocols can be used for delayed transfer as well (e.g. receiving telemetry data stored at a ground station) if the user wants to use the same software for both realtime and delayed data.

A data delivery protocol, which is called the Space Data Transfer Protocol (SDTP), is the standard application layer protocol of **DIOSA** for sequential transfer. This protocol is used together with a standard transport protocol such as TCP/IP or X.25 (Fig. 7). Details of SDTP are described in the next subsection.

The protocols of the Batch Transfer Type are used for transferring files. A standard file transfer protocol such as FTP or FTAM can be used for batch transfer (Fig. 7).

Most raw data will be transferred with the sequential transfer protocols. However, the

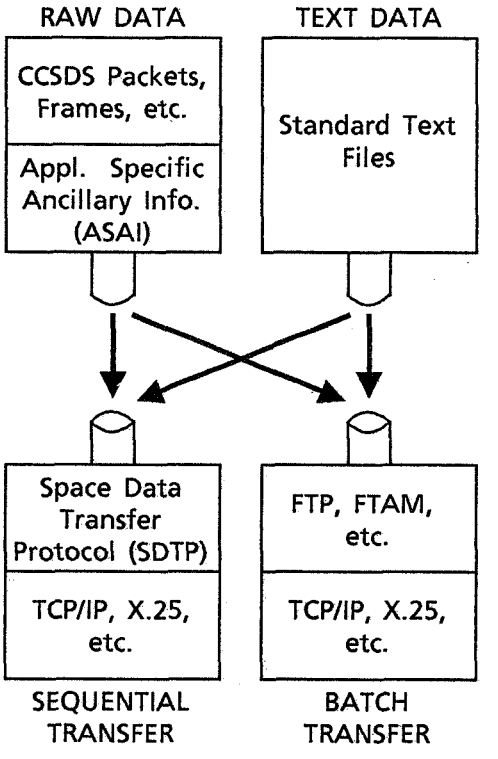


Fig. 7 Protocol Stack Used for Data Ports

batch transfer protocols can be used for transferring raw data which is already stored in a file. Most text data will be transferred with the batch transfer protocols. However, the sequential transfer protocols can be used for transferring text data if it has to be transferred immediately after its generation.

4.3 Space Data Transfer Protocol (SDTP)

The Space Data Transfer Protocol (SDTP) is a connection oriented protocol used for delivering sequential space data (e.g. sequence of CCSDS packets) from a functional block to another functional block. SDTP has a capability of (1) requesting data transfer, (2) specifying Spacecraft ID (SCID), Application Process ID (APID) and other attributes of data, and (3) notifying of any events related to data delivery (e.g. loss of RF signal). The formats of the Protocol Data Units (PDUs) of SDTP is shown in Fig. 8.

SDTP is used as follows. When Functional Block *Tom* wants to receive a sequential data

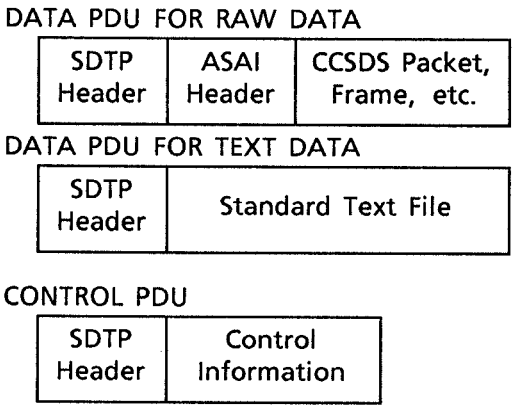


Fig. 8 Format of SDTP_PDU

from Functional Block *George*, *Tom* opens a connection of SDTP with *George* saying "Hi, George. I want to receive packets with APID=5 and SCID=32 in realtime" (Connect Request). Then *George* answers "OK, Tom" (Connect Response) and starts data delivery. When the data delevery is (or must be) terminated, either end of the connection can disconnect the connection.

SDTP can be used together with a data distribution service as explained below as well as in a bilateral mode.

In a distributed space operations system, a data stream often needs to be delivered to several destinations simultaneously (multicasting). For example, some telemetry data may be monitored by several workstations simultaneously. In **DIOSA**, data distribution and multicasting are performed by the Data Distribution Functional Block (DDFB). A DDFB is placed in every domain where it is needed. The DDFB of a domain acts as the data server of the domain. The DDFB receives sequential data from the DDFB of another domain or from another functional block in its own domain, and distributes the received data to functional blocks in its domain (Fig. 9).

In such a situation, when a functional block wants to receive a sequential data, it opens a SDTP connection with the DDFB of that domain requesting transfer of that data. Then the DDFB checks whether or not it is

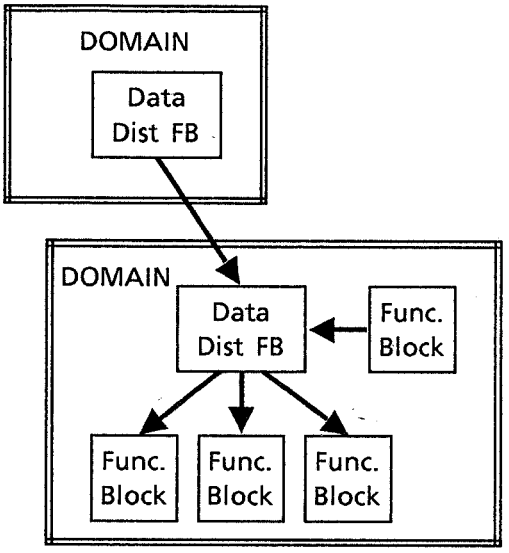


Fig. 9 Data Distribution With Data Distribution Functional Block (DDFB)

receiving that data, and if it is not, it sends a request for that data to another DDFB. Therefore, the requesting functional block does not have to know where the data originally comes from. It always sends requests to the DDFB of its domain.

5. MANAGEMANT STRUCTURE

Each domain has a functional block, called the Domain Managemet Functional Block (DMFB), which manages the functional blocks of the domain. Management within a domain

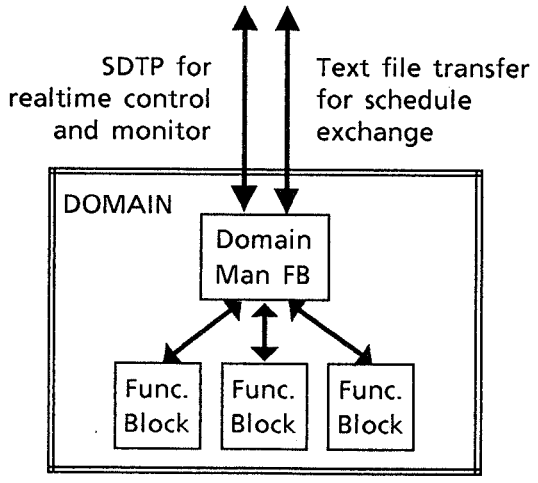


Fig. 10 Intterdomain Management With Protocols of Fig. 7

Table 1 Some Examples of Functional Blocks

Functional Block Name	Typical Location	Function	Data Input	Data Output
Domain Management	Every Domain	Manage functional blocks of the domain	Event file (Text Data, Batch Transf.), Config. change req. (Text Data, Seq. Tsf.)	Status information (Text Data, Seq. Transf.)
Data Distribution	Every Domain	Distribute data	Various data (Raw Data, Seq. Transf.)	Various data (Raw Data, Seq. Transf.)
Command Transmission	Ground Station	Transmit commands	Command data (Raw Data, Seq. Transf.)	Command signal (Analog)
Telemetry Reception	Ground Station	Receive and decode telemetry	Telemetry signal (Analog)	Telemetry data (Raw Data, Seq. Transf.)
Radiometric Data Collection	Ground Station	Collect range and doppler data	Radiometric signal (Analog)	Range/doppler (Raw Data, Seq. Transf.)
Spacecraft Control	Ops Center	Generate commands and verify results	Event file (Text Data, Batch Transf.), Telemetry data (Raw Data, Seq. Transf.)	Command data (Raw Data, Seq. Transf.), Anomaly msg (Text Data, Seq. Transf.)
Timeline Generation	Ops Center	Generate timelines	Ops request (Text Data, Batch Transf.)	Event file (Text Data, Batch Transf.)
Orbit Determination	Ops Center	Determine orbit	Range/doppler (Raw Data, Seq. Transf.)	State vector (Text Data, Batch Transf.)
Data Archive	Science Center	Archive data	Telemetry data (Raw Data, Seq. Transf.)	Telemetry data (Raw Data, Batch Transf.)
Data Analysis	Science Center	Analyze data	Telemetry data (Raw Data, Batch Transf.)	Papers to be published in journals

is performed with the management port of functional blocks and a standard network management protocol like SNMP.

Managinment between domains can be performed either with a network management protocol or with the protocol suit described in Section 4. In the latter case, schedule information is exchanged in Standard Text Files with the File Transfer Protocol, while configuration change messages and status information messages are exchanged with SDTP in realtime (Fig. 10).

6. EXAMPLES OF FUNCTIONAL BLOCKS

In Table 1, some examples of functional blocks are given. Please note that this is not a comprehensive list of functional blocks. An example of an operations system constructed with **DIOSA** is shown in Fig. 11.

7. CONCLUSION

This paper presented the concept of a distributed multimission operations system.

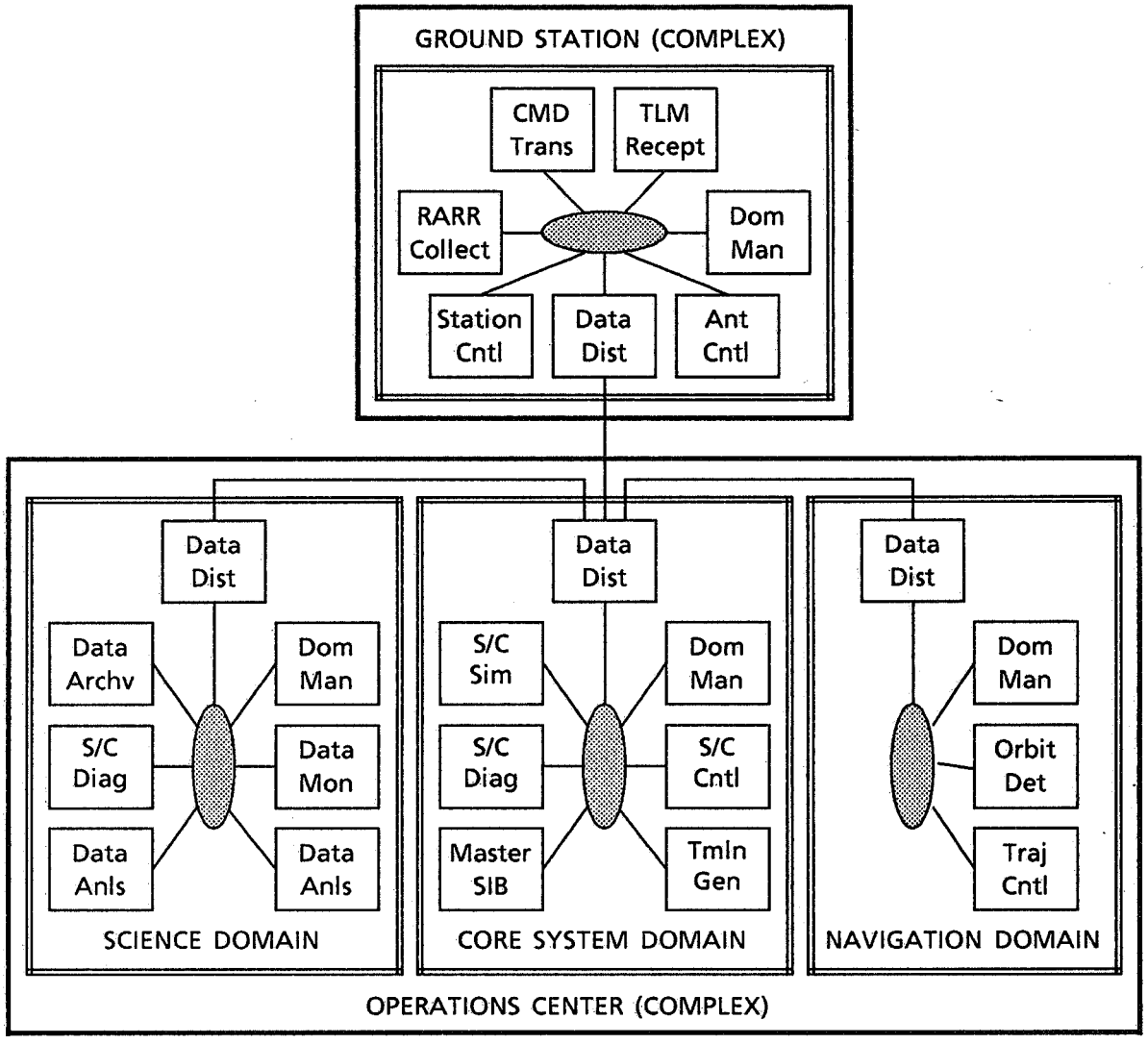


Fig. 11 Example of Spacecraft Operations System

We plan to develop a prototype in a few years to verify the validity of this concept.

8. REFERENCES

Cipollone, G., & McKay, M. (1992). An engineering database management system for spacecraft operations. *Proc. of the Second Int. Symp. on Ground Data Systems for Space Mission Operations*, 787-790.

Holder, B., & Levesque, M. (1992). Making adaptable systems work for mission operations: A case study. *ibid*, 327-331.

Kaufeler, J.-F. et al. (1992a). The advanced technology operations system ATOS. *ibid*, 425-434.

Kaufeler, J.-F. et al. (1992b). The European space agency standard for space packet utilization. *ibid*, 813-818

Mandl, D. et al. (1992). SAMPEX payload operation control center implementation. *ibid*, 63-68.

Newsome, P., & Otranto, J. (1992). The advanced orbiting systems testbed program: results to date. *ibid*, 501-506.

Systems Engineering

5. Systems Engineering Tools

* *Presented in Poster Session*

111191 N95-17606

354323 p. 8

RE-ENGINEERING THE MISSION LIFE CYCLE WITH ABC & IDEF

Daniel Mandl, Michael Rackley
NASA/GSFC Code 511
Greenbelt, MD 20771

Jay Karlin
Viable Systems Inc.
12236 Stoney Bottom Rd. Germantown, MD 20874

ABSTRACT

The theory behind re-engineering a business process is to remove the non-value added activities thereby lowering the process cost. In order to achieve this, one must be able to identify where the non-value added elements are located which is not a trivial task. This is because the non-value added elements are often hidden in the form of overhead and/or pooled resources. In order to be able to isolate these non-value added processes from among the other processes, one must first decompose the overall top level process into lower layers of sub-processes. In addition, costing data must be assigned to each sub-process along with the value the sub-process adds towards the final product.

IDEF0 is a Federal Information Processing Standard (FIPS) process-modeling tool that allows for this functional decomposition through structured analysis. In addition, it illustrates the relationship of the process and the value added to the product or service. The value added portion is further defined in IDEF1X which is an entity relationship diagramming tool. The entity relationship model is the blueprint of the product as it moves along the "assembly line" and therefore relates all of the parts to each other and the final product. It also relates the parts to the tools that produce the product and all of the paper work that is used in their acquisition.

The use of IDEF therefore facilitates the use of Activity Based Costing (ABC). ABC is an essential method in a high variety, product-customizing environment, to facilitate rapid response to externally caused change. This paper describes the work being done in the Mission Operations Division to re-engineer the development and operation life cycle of Mission Operations Centers using these tools.

1. Introduction

With NASA budgets becoming tighter each year, the Mission Operations Division (MOD), which is part of the Mission Operations and Data Systems Directorate at Goddard Space Flight Center (GSFC), has been forced to reevaluate and change how it has traditionally built Ground Data Systems (GDS). The MOD, as an enterprise, could very simply not afford to continue doing "business as usual".

The traditional GDS approach was to implement large facilities that supported multiple, simultaneous missions, with each facility providing a specific type of operational support function. The systems were also typically developed using the traditional development life cycle model, with formal reviews for requirements and design, and large amounts of formal documentation. This GDS architecture and development approach may have been appropriate given the technology and budgets

PRECEDING PAGE BLANK NOT FILMED PAGE 1326 INTENTIONALLY BLANK

available at that time, but the MOD could no longer afford this approach. The development cycle was proving to be too long and expensive, and the operations costs associated with the architecture were accounting for too much of the overall budget. The MOD enterprise thus set out to improve itself in these two areas.

A new GDS approach has been adopted that takes advantage of the relatively recent advances in technology and industry standards. The new concept is to build a GDS that is tailored to a mission or family of missions. This required the development of an underlying architecture approach that was flexible, scalable and evolvable.

As mentioned, the MOD also set out to reduce the development life cycle cost. Analysis showed that while operations costs could be reduced with the new architecture, development costs associated with that architecture were not experiencing comparable cost savings. This was surprising to many since the new architecture employs high levels of software reusability across missions. The MOD came to the conclusion that though reusability was an important factor in reducing costs, any further substantial savings could only be achieved by improving the development process itself.

The remainder of this paper focuses in particular on the MOD's efforts thus far to improve the process for requirements analysis. Sections 2 through 4 introduce a costing method referred to as Activity Based Costing (ABC), and the Integration Definition for Modelling (IDEF) modelling used to support this method. The remaining sections describe how this method and tool were applied to the MOD's process improvement experiment.

2. Process Engineering with ABC; Pricing a Requirement

In order to manage processes effectively and to make appropriate decisions about changing them, a detailed and accurate set of metrics is essential. Pooled resources tend to distort the actual cost of a process. When a resource is shared, the traditional method for assigning cost to a project is to use the average cost of past missions, instead of assigning a cost that is tailored to the true needs of the project. With the new GDS approach of tailoring the system to each mission's needs, a comparable costing method was needed so that actual mission requirements could be individually costed, thus yielding a more accurate overall project cost estimate.

ABC is a method devised to model the cost of any process which has first been decomposed through modeling into primitive activities that serve as its building blocks. Once the primitive activities have been identified, costs can be assigned to those primitives. Then optimization of the general process can be performed in forums such as process improvement committees.

IDEF is a Federal Information Processing Standard (FIPS) that can be used as a tool to perform ABC. IDEF actually consists of an integrated pair of tools: the activity modeler (IDEF0) and the data modeler (IDEF1X). IDEF0 is used to model the activities that occur to produce a product or service and therefore shows the interrelationships of work being done in different groups. IDEF1X shows what is being passed between processes by defining a template (i.e. a data structure) for each item. This provides for more accurate, rapid and meaningful insight into interactions among groups. An example

of this might be a form sent to request a service from another group. IDEF therefore acts as an intergroup coordination tool by providing the overall blueprint for the entire process. The best way to use this tool to coordinate different groups is to put IDEF on a distributed network that is accessible on-line to all participants in the process.

3. The Power of IDEF & ABC

IDEF with ABC allows one to continuously assess the implementation of an overall process and thereby determine the point at which the implementation needs to be changed in order to reduce costs.

For example in figure 1, a conceptual process is depicted. The goal is to get from X to Z, however there is a constraint that regardless of what path is taken, it must cross Y as an intermediary point or constraint. For example, X might be the start of a project and Z might represent having a design. Before one can have a design, one must have the system requirements which is represented by Y. There are a variety of paths to get from X to Y, each costing a different amount. The cost to get from X to Y is the sum of the cost of each of the activities traversed to get from X to Y. In this case, path 2 happens to be the least expensive.

But what happens when a technology comes along that causes the cost of activity 3 to decrease from $3.00 to $0.50? This causes the least expensive path from X to Y to become path 1 instead of path 2. But what happens if there are thousands of activities performed by loosely coupled groups, with each injecting technology to perhaps automate an activity in their area? Thus what

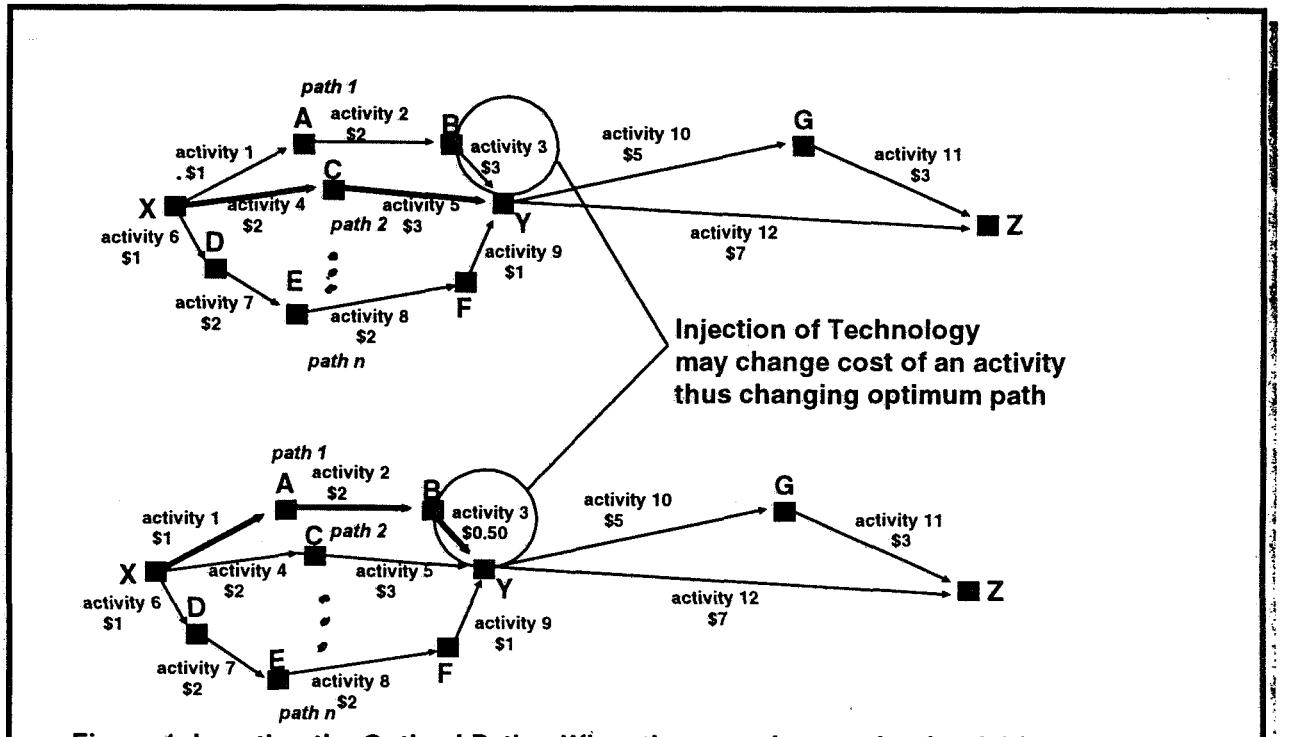


Figure 1 Locating the Optimal Path - When there are thousands of activities choosing the optimal path is nontrivial if it must occur on a continuous basis. Note that least expensive path between X and Y changes when technology changes cost of underlying task.

looked relatively simple in this example is in reality very complex!

What happens when the organization chooses to solve problems with the same processes without considering cost impacts of increasing complexity? Without examining the activities within processes and removing non-value added old activities , unneeded constraints are carried along like deadwood at extra expense. For example, it may be necessary to derive the system requirements, but it may not always be necessary to have a formal System Requirements Review (SRR) if there is a high degree of reusability.

4. IDEF Nomenclature

Of course, models created through the use of IDEF are more sophisticated than the conceptual drawing in figure 1. Figure 2

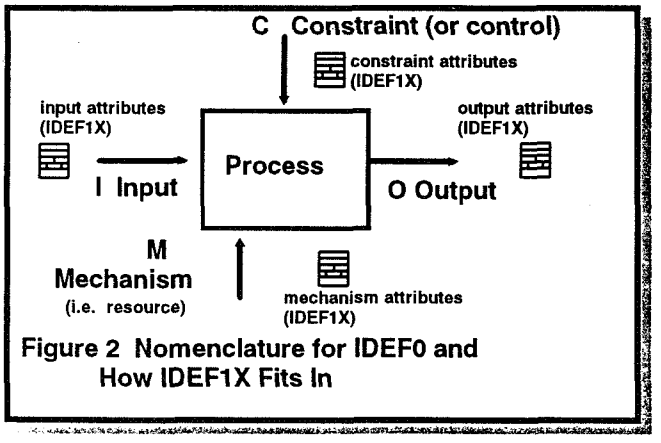


Figure 2 Nomenclature for IDEF0 and How IDEF1X Fits In

depicts the key to reading an IDEF0 drawing. Note the acronym ICOM helps to identify the key elements of an IDEF drawing where I represents Input, C represents Constraint, O represents Output and M represents Mechanism. One of the key features of IDEF is its ability to link the drawings to an underlying database. Also, the drawings are hierarchical to allow one to reveal more and more detail as needed.

5. Experiment Approach

The following steps were taken in conducting the MOD process improvement experiment:

a. Identify target process for improvement.
b. Gather baseline cost data.
c. Define activities that comprise the process using IDEF0 (referred to as AS-IS process).
d. Identify potential problem areas.
e. Identify the underlying business rules using IDEF1X
f. Develop improved process (referred to as TO-BE process).
g. Quantify potential cost improvement using ABC.
   (1) Show main cost driver activities.
   (2) Identify resource cost drivers, e.g., needing specialized skill only half-time but required to hire a full-time person.
h. Measure the new process to verify improvement.

6. Target Process Selection

In order to define the target process for improvement, a typical mission life cycle was first defined as follows:

- Develop System Requirements
- Design System
- Build and Test System
- Operate System
- Maintain System

This top level process is illustrated in figure 3. Although this figure applies specifically to the development and use of a Mission Operations Center (MOC), it was actually derived from a higher level diagram of the

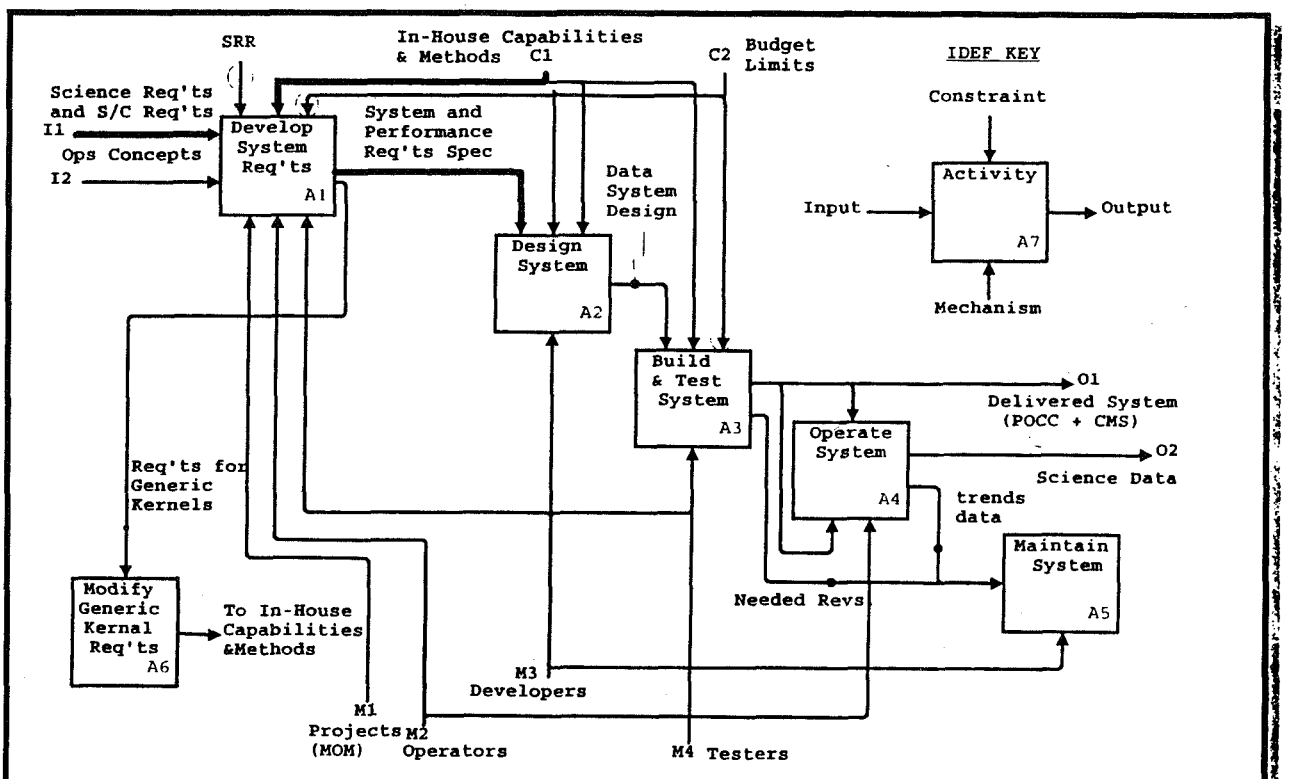


Figure 3 Top level "AS-IS" Life Cycle Model

typical life cycle for the entire GDS.

The next step involved taking some gross measurements of the different life cycle phases to see where the largest portion of the resources was spent. The build phase of the life cycle was found to be an increasingly smaller portion of the total cost. This was due to the employment of reusable building blocks. This meant that the major cost drivers no longer resided in the generation of software. They instead resided in the other life cycle phases, primarily developing system requirements and testing. Therefore the greatest remaining potential for cost savings was in these other phases. The requirements analysis phase was thus chosen as the target process for improvement.

7. Process Analysis

The "AS-IS" and "TO-BE" requirements processes are illustrated in figures 4 and 5 respectively. Figure 6 shows the underlying cost spreadsheets and figure 7 is an example of the underlying entity relationship model.

A cursory examination of figure 4 , suggested the following problems:

a. Function A13 "Negotiate Exceptions" is a trigger on Functions A11 and A12. That is, A13 is required to feed certain previously identified exceptions back to their source for reconsideration. This reiteration (loop-back) multiplies the effort.

b. Additionally, there is conflict between inputs into the "Analyze Mission Req." from two different sources. This indicates that the changes resulting from the unresolved constraints and inputs (exceptions) perturb the on-going preparation of other requirements, necessitating

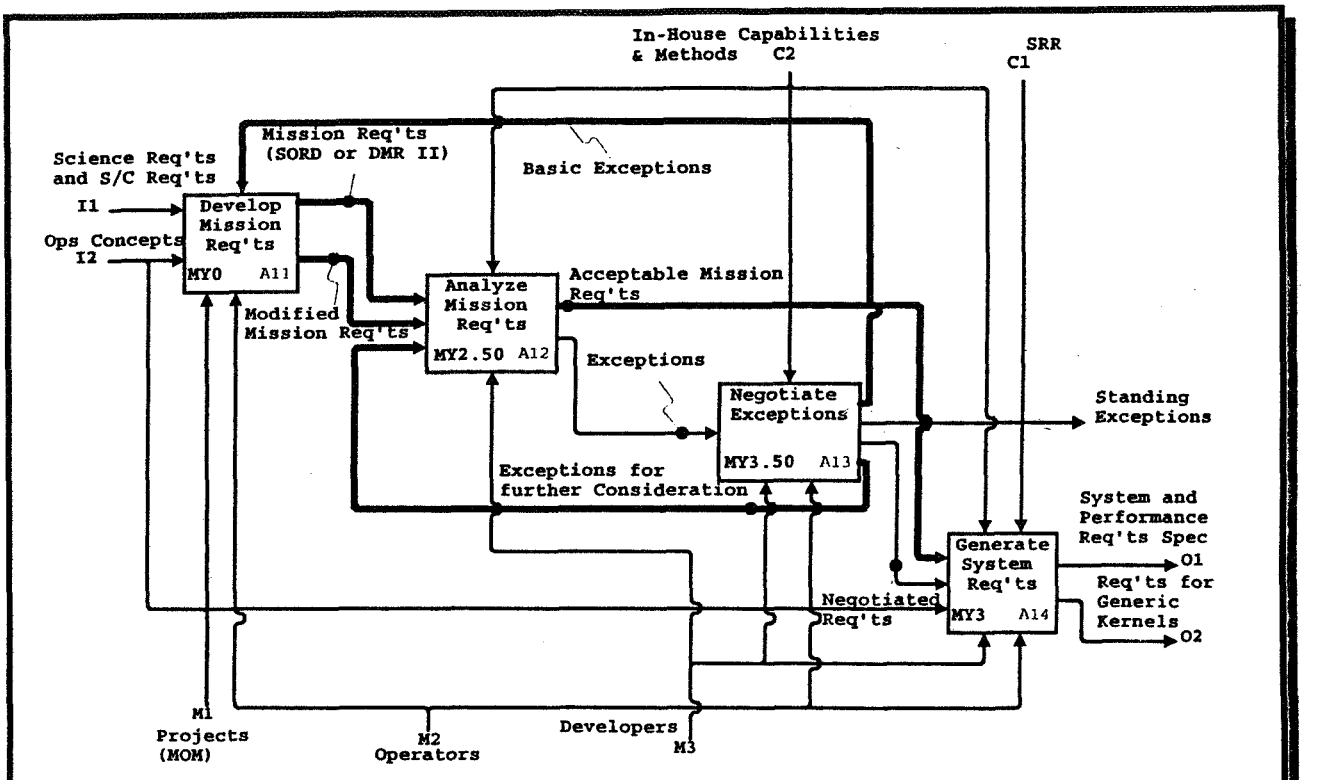


Figure 4 "AS-IS" method to develop system requirements

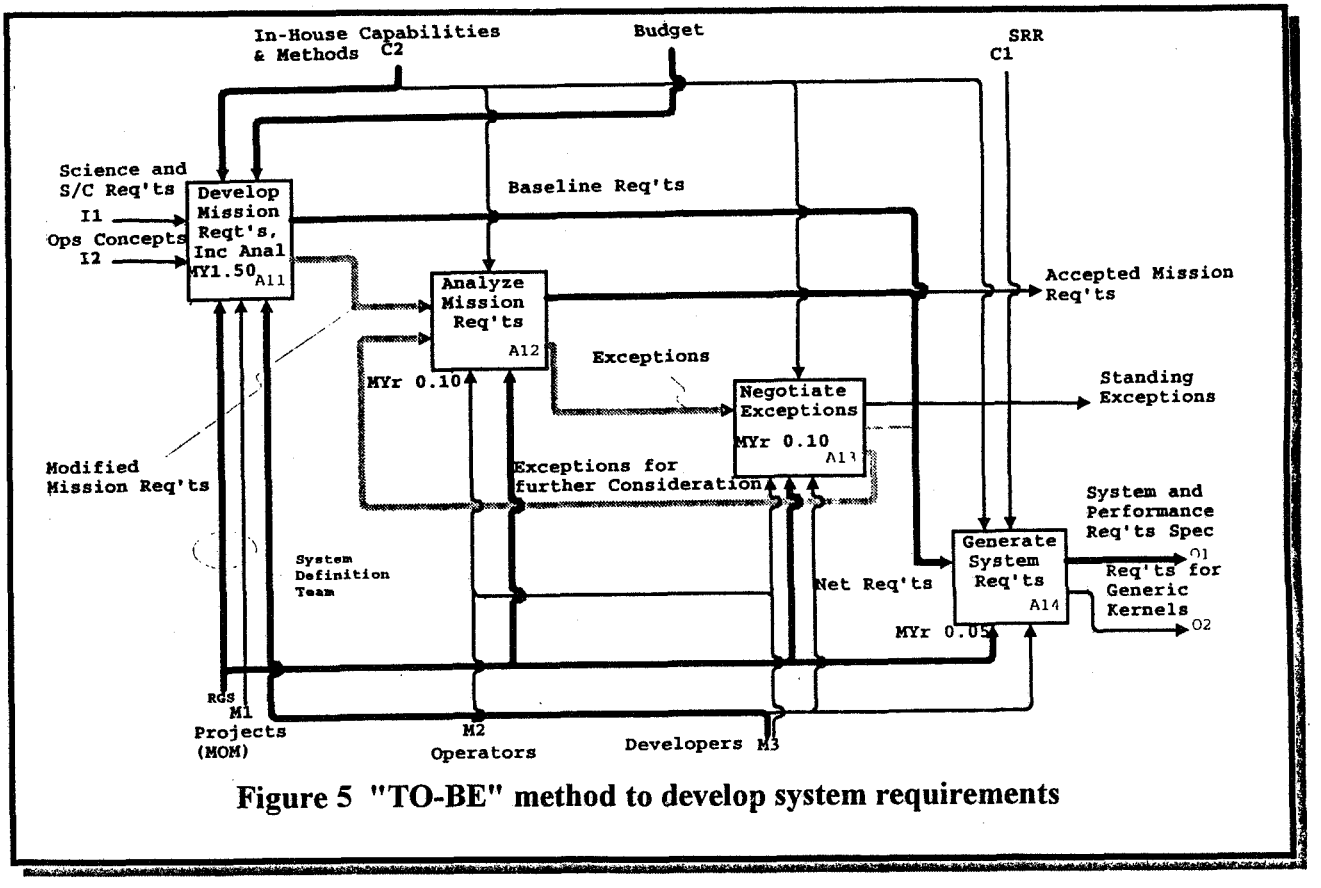


Figure 5 "TO-BE" method to develop system requirements

coordination. Thus, management and developer resources, as well as operator and projects' personnel time, is consumed unnecessarily.

8. Process Improvement

The AS-IS process has been redesigned to largely eliminate feedback as shown in fig. 5, TO-BE, while retaining its basic functionality. This was accomplished by making the Developers, mechanism M3, and their personal knowledge of the constraint C2, "In-House Capabilities", available to A11, which is the initiating point in the requirements process. This early developer involvement has removed many of the information interfaces that used to require translation and documentation "at-a-distance" between A11, A12 and A13, and that had been burning up a significant amount of manpower. To facilitate dynamic person-to-person interaction, the process designers specified there be a System Definition Team (SDT) in order to ensure "eyeball-to-eyeball" operator and developer physical proximity, which eliminated the shuttling of documents back and forth. The Requirements

Node	Develop System Requirements "AS-IS"	Developers	Myrs
A0	Develop and Operate a Project Ops Control Center	9.00	9.00
A1	Develop System Req'ts	9.00	9.00
A11	Develop Mission Req'ts	0.00	0.00
A12	Analyze Mission Req'ts	2.50	2.50
A121	Decompose Mission Req'ts	0.60	0.60
A122	Document Exceptions	0.70	0.70
A123	Determine Acc. and Unacc. Mission Req'ts	1.20	1.20
A13	Negotiate Exceptions	3.50	3.50
A131	Evaluate by Proj/Ops	0.00	0.00
A132	Consider Possible Alternatives	1.50	1.50
A133	Interact on Issues with Affected Parties	2.00	2.00
A14	Generate System Req'ts	3.00	3.00
A141	Develop Initial Ops Scenarios	0.00	0.00
A142	Compare With In-House Methods	1.50	1.50
A143	Identify Performance Req'ts	1.50	1.50
A2	Design System	0.00	0.00
A3	Build & Test System	0.00	0.00
A4	Operate System	0.00	0.00
A5	Maintain System	0.00	0.00
A6	Modify Generic Kernal Req'ts	0.00	0.00

Node	Develop System Requirements "TO-BE"	Developers	M Yrs
A1	Develop System Req'ts	1.75	1.75
A11	Develop Mission Req't's, Inc Anal	1.5	1.5
A111	Document Mission Req'ts	0.5	0.5
A112	Compare with In-House Capabilities	0.5	0.5
A113	Reconcile Differences	0.5	0.5
A12	Analyze Mission Req'ts	0.1	0.1
A121	Decompose Mission Req'ts	0.05	0.05
A122	Develop Initial Ops Scenarios	0	0
A123	Determine Acc. and NonAcc. Req'ts	0.05	0.05
A13	Negotiate Exceptions	0.1	0.1
A131	Evaluate Documentation	0.05	0.05
A132	Document Exceptions	0.05	0.05
A133	Interact on Issues with Affected Parties	0	0
A14	Generate System Req'ts	0.05	0.05
A141	Develop Initial Ops Scenarios	0	0
A142	Compare With In-House Methods	0	0
A143	Identify Performance Req'ts	0	0

Figure 6 Underlying cost spreadsheets

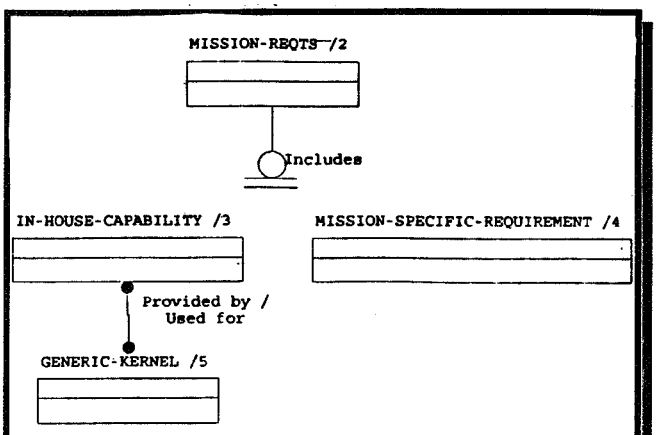


Figure 7 Example of underlying entity realtionship model

Generation System (RGS) replaced this physical function by recording the agreements in real time, thus furthering the cost reduction effect.

9. Results

Chart 1 shows the results that were achieved. Note that, in spite of the fact that they were favored by very high levels of reuse, previous missions still required significant developer effort to get through the requirements analysis process. The shaded area indicates the newly installed process .

10. Future Efforts

The remainder of the life cycle can also be treated in this manner. Since technology changes so quickly, this analysis, including an activity cost breakdown, has to be constantly monitored. The next step, once these models are in place, would be to feed the results into a simulator such as Work Flow Analyzer to do probabilistic analysis on the feedback loops.

	S/W KDSI	% Reuse	DevMyrs to Do Req
SAMPEX	180	40	11
FAST	220	72	9
SWAS	220	87	7
XTE(AS-IS)	250	50	9
ACE(TO-BE)	250	70	1.75

Chart 1 Results of experiment; Only developers' effort considered, no management or publications

11. Conclusion

Together, IDEF and ABC allow large organizations to coordinate their processes and to create a living blueprint for changing and improving business practices. These tools also provide a forum for each individual to identify his or her viewpoint and to comment on these processes from such a perspective. For the business entity and its organization to remain viable, and since the primary cost savings potential is in the process and not the product or product architecture, these types of management methods must be instituted along with such items as product innovation. They in fact provide a means to achieve process innovation. Finally, without these type of tools, large organizations find themselves in the situation of making decisions without the necessary metrics.

12. References

Kaplan, Robert S., *Management Accounting for Advanced Technological Environments*, Articles, August 25, 1989.

Moravec, Robert D. & Yoemans, Micheal S., *Using ABC to Support Business Re-Engineering in the Department of Defense*, Journal of Cost Management, Vol 6, No 2, Summer 1992.

NIST, *Integration Definition for Function Modeling (IDEF0)*, FIPS Publication, Dec 21, 1993.

111192 N95-17607

354325 P-8

MO&DSD ONLINE INFORMATION SERVER AND GLOBAL INFORMATION REPOSITORY ACCESS

Diem Nguyen and Kam Ghaffarian
Loral AeroSys
7375 Executive Place
Seabrook, Maryland 20706

Keith Hogie and William Mackey
Computer Sciences Corporation
7700 Hubble Drive
Lanham, Maryland 20706

ABSTRACT

Often in the past, standards and new technology information have been available only in hardcopy form, with reproduction and mailing costs proving rather significant. In light of NASA's current budget constraints and in the interest of efficient communications, the Mission Operations and Data Systems Directorate (MO&DSD) New Technology and Data Standards Office recognizes the need for an online information server (OLIS). This server would allow

- Dissemination of standards and new technology information throughout the Directorate more quickly and economically
- Online browsing and retrieval of documents that have been published for and by MO&DSD
- Searching for current and past study activities on related topics within NASA before issuing a task

This paper explores a variety of available information servers and searching tools, their current capabilities and limitations, and the application of these tools to MO&DSD. Most importantly, the discussion focuses on the way this concept could be easily applied toward improving dissemination of standards and new technologies and improving documentation processes.

INTRODUCTION

The Mission Operations and Data Systems Directorate (MO&DSD) online information server (OLIS) has been established to share the latest trends, technologies, and standards information among MO&DSD organizations. OLIS offers the same information via multiple retrieval methods, including file transfer protocol (FTP), wide-area information server (WAIS), Gopher, and Mosaic. From their personal computer (PCs), users can electronically retrieve standards information (e.g., Consultative Committee for Space Data Systems recommendations), new technology information (e.g., asynchronous transfer mode), or previous studies and results. This server also provides pointers to other public servers on the Internet.

In support of this initiative, the Systems, Engineering, and Analysis Support technical support group began looking into mechanisms for providing better access to standards and new-technology-related documentation. A standards database on CD-ROM was examined; however, it was found that a local PC-based retrieval mechanism is rather cumbersome and not easily accessible for the more than 4,000 users of the MO&DSD community.

Because Transmission Control Protocol/Internet Protocol (TCP/IP)-based applications are widely available for any platform, the technical support group concentrated on an open systems approach using the Internet as the access mechanism. This approach provides maximum accessibility within Goddard Space Flight Center (GSFC), across NASA, and even to international users.

GOAL

The key issue is giving users timely, reliable, and relevant information. The overall goal of OLIS is to make information available, convenient, and easily accessible to MO&DSD personnel. Users can access and share in minutes information that once took weeks to disseminate. Most significantly, OLIS

focuses on ways to improve awareness of standards, new technologies, and documentation processes. OLIS was established to show what can be done and to encourage other groups to set up similar servers.

INTERNET SERVICES

Many available services require only a basic connection to the Internet, where client applications are used to access information servers. More advanced client/server technologies can access the capabilities of more basic clients and servers (as shown in Figure 1).

Figure 1 shows, on the left, the various types of client programs and, on the right, the various types of servers those programs access. These client/server communications can also occur with both elements on the same system so the same process can be used to access local files, as well remote files.

Also shown are two major types of interfaces. One is a basic textual interface that can be supported on full-screen text terminals, such as the VT100 series. This type of interface can be supported by most computer vendors and can operate over dialup and Telnet connections. The textual interface works very well over low bandwidth connections as it only uses text and does not include any graphics.

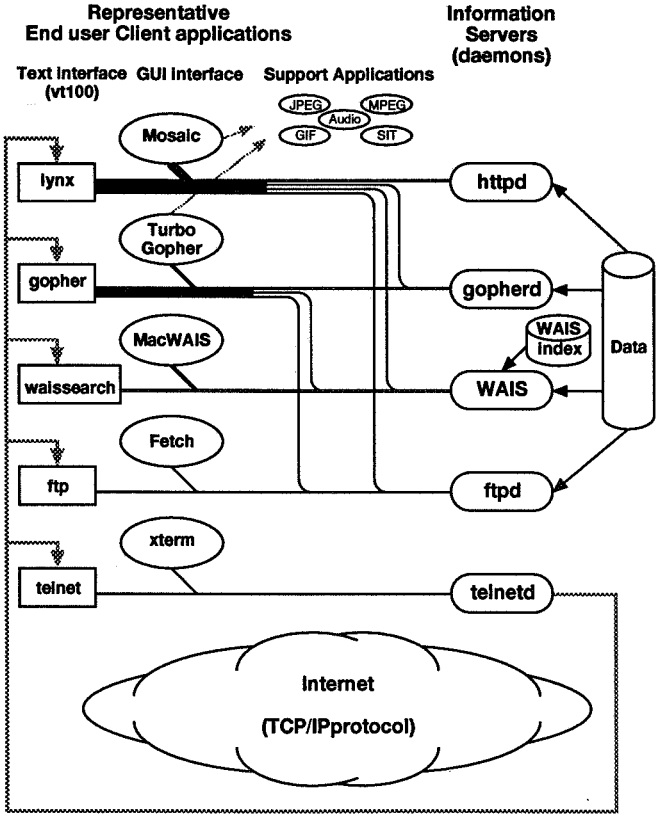


Figure 1. Hierarchy of Client/Server Technologies

The other type of interface is the full graphical user interface (GUI), which normally provides a more user-friendly interface. However, this type of interface requires an end-user computer system with proper windowing capabilities and is not commonly supported over dialup links.

The support applications (shown at the top of Figure 1) are used by Gopher- and Mosaic-type clients. These applications are activated when the client detects an action or file format that cannot be processed by the client itself. These applications are normally activated after the client has transferred a special-format file to the user's disk drive. Support applications include Telnet applications or special file format processors such as graphics interchange format (GIF) or joint photographic expert group (JPEG) image viewers, QuickTime or MPEG movie viewers, audio file players, or word processor packages.

Domain Name Service

Remote systems are normally identified by an Internet format hostname address such as "ddwilson.gsfc.nasa.gov." This name must first be converted into a standard four-number IP address such as 128.183.92.144, which can then be used to communicate with the remote system.

A "directory assistance" service, referred to as Domain Name Service (DNS), is available that automatically looks up IP addresses for a given hostname and vice versa. This automated service is silently invoked every time an Internet hostname is given for connection. The lookup is performed by querying a host called a DNS name server. (The identity of this server should be provided at the time the user's computer is set up for Internet access.) In addition to letting user computers resolve IP addresses automatically, the DNS maintains entries, listing the Internet hostname and IP address of the user's computer. This lets remote systems such as anonymous FTP sites determine if the computer is a registered Internet host.

Telnet

Telnet is the standard TCP/IP remote login protocol. To "Telnet" to another system is to run a Telnet client program that establishes a connection to a Telnet server and then logs on to that system with some user ID. This client/server combination provides a basic, text-only access mechanism but works internationally. Connecting to a system via Telnet normally looks exactly like connecting via a dialup line or direct connection. However, with Telnet, a user can support multiple connections to remote systems simultaneously.

On an X-window system, a local window-based program such as xterm is used to provide a scrollable text window from which a Telnet session is then established. On a Macintosh or PC, a window-based application [such as the public domain program, National Center for Supercomputing Applications (NCSA) Mosaic], Telnet, or a commercial package can be used to provide the client end of the Telnet connection.

The main limitation of Telnet access is that only one system can be accessed per session, and the user must know and use the appropriate commands to examine information there. Some systems support a "guest" login with no password, but most systems require individual user accounts and passwords. A problem with using Telnet for public information access is that it normally requires a user to execute too many commands on the remote system. The combination of user accounts and extensive command capabilities make this is a very poor way to provide a large user community access to many information servers easily and transparently.

File Transfer Protocol

FTP is another information access mechanism that has been in existence since the time of the ARPANET in the mid-1970s. To "FTP" to a system is to run an FTP server, examine file directories, and download or upload files of interest. FTP provides access to a single site at a time and requires explicit commands to disconnect from one site and then connect to another. However, it does provide basic access to anything that can be put into a file. Today, a wide range of information is digitized and stored in files. Users can log on to an FTP server if they have an account and password there. There are also thousands of FTP servers set up for "anonymous ftp" access. In anonymous ftp, the user can log in with a user name of "anonymous" and a password (generally the user's name and/or email address) and then download files or possibly upload files into a special directory set up for that purpose. An anonymous ftp server was set up as the first information server for OLIS.

FTP initially operated in a command line mode (as shown in Figure 2), where the user entered FTP commands to browse file directories and retrieve files.

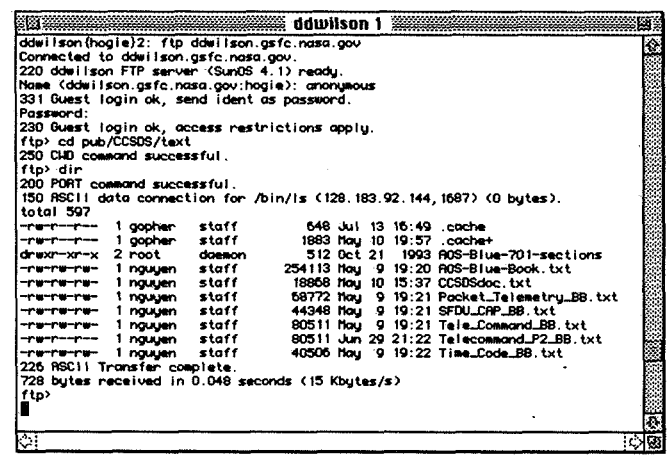


Figure 2. Command Line FTP Interface

More user-friendly, point-and-click, or GUI FTP client programs have been developed in the last few years. These programs eliminate the need for learning FTP commands, but they still provide only lists of filenames to choose from. On a Macintosh, Fetch is one public domain program that implements FTP. It simply asks for the name of the host to which the user wants to be connected and then provides a point-and-click interface to FTP versus the basic command line mode of operation. Users can browse file directories and, if the filename is descriptive enough, can figure out the content and select files for downloading (as shown in Figure 3).

Although FTP provides a very widely used mechanism for access to file repositories, it still provides only simple lists from a single server at a time. Users need to know how to locate other servers and explicitly connect to them. Therefore, FTP does not really provide the sort of easy access desired for wide use. Most FTP sites ask for some sort of password, even

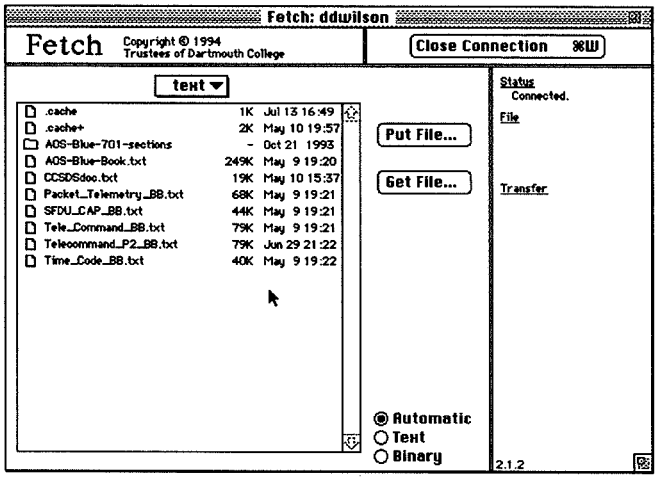


Figure 3 - GUI FTP Interface

if they don't care what it is. The password is a courtesy in the event the FTP site wants to record log-in access. Also, as a security measure, many FTP sites check to ensure that the DNS lists the user's IP address as a registered host.

Wide-Area Information Server

WAIS client/server software began appearing in 1992 and looked very promising for the MO&DSD goals of providing easy user access to documents, as well as full-text search capability on document repositories. A WAIS server was set up on OLIS in 1993. The popularity of WAIS increased rapidly once public domain implementations of the client/server software became stable and available. This was the beginning of the deployment of more advanced clients and servers and associated protocols that allowed users to quickly and easily access more than one server site.

The major feature of WAIS is its full-text search capability. When a document is loaded into a WAIS server, it passes through an indexing application that scans the document and builds a list of all the words in the document and their frequency of occurrence. This information and a pointer back to the original document file are then added to the master indexes on the server. An end user uses a WAIS client program to format a question to be asked of one or more WAIS servers. The question contains keywords of interest and pointers to the servers to be queried. The client program then establishes a temporary connection to a server, asks the question, receives the responses, and then drops that connection and connects to the next server. After querying all servers, the client displays the resulting filenames to the user. The user can then select any file of interest, and that file is transferred to the user's system.

WAIS clients can be set up to easily query multiple servers with minimal user interaction. The user is not involved in the process of connecting to or disconnecting from each server to be polled. Once a question has been established with keywords and servers identified, the question can be saved for later use. The complete search of the same question can then be executed at a later date with a single user action. The results are returned in a single list indicating filenames, together with a score of how frequently the keywords occur in each file. With WAIS, users begin to see the Internet as a large information repository.

An inherent limitation with WAIS is that the indexer supports only ASCII text as input format; i.e., graphical content must be removed and only the text entered into a WAIS server. This presents a problem in that, today, many documents are prepared in desktop publishing packages and contain tables and graphics, as well as text. To get around this problem, a WAIS server can be used to locate documents of interest, and then the fully formatted version can be retrieved separately. Another limitation is that the user is responsible for identifying all information sites to be queried. One of the files at a site can be a list of other WAIS servers that a user can add to the list of sites to be queried. But the user is still responsible for identifying each site to be used.

With Gopher and WWW, pointers to other sites with information of interest can be followed more easily and transparently, as discussed in the following paragraphs.

Gopher

Gopher servers exploded on the Internet during 1993, with thousands in existence today. Gopher is a popular menu-based information system that integrates access to Telnet, FTP, and WAIS in an easy-to-use interface. Gopher also provides easy access to multiple sites. Users move transparently from one server to another with the Gopher client knowing which protocol to use to access each server. Thus, users spend time searching for information rather

than trying to determine how to use each service and navigate the network.

TurboGopher (shown in Figure 4) is a Macintosh version of a Gopher client that allows users to retrieve the same information accessible via a typical VT100 Gopher client. TurboGopher, however, provides the following advantages:

- Point-and-click graphical interface
- Files saved directly to local Macintosh
- User does not have to first log into UNIX host

When an item is selected from a Gopher menu, one of three things happens:

- Lower-level menu on the same system appears
- Connection is initiated to another server and menu actually stored there appears
- Action is initiated, such as a file transfer or Telnet session

The first case is actually a special instance of the second. In the second case, for menus served by another system on the Internet, TurboGopher automatically connects to the new system and sends the proper low-level commands to retrieve the menu being invoked. As the user browses through the menu hierarchy, the program automatically switches from system to system as needed. In the third case, when a user invokes a menu item, some special action may be performed (e.g., automatically initiate downloading a file). This is implemented essentially by FTP-like functionality built into TurboGopher, while a Telnet session is initiated by activating a support application.

Worldwide Web

The WWW project began at the European Particle Physics Laboratory (CERN). WWW seeks to build a distributed hypermedia system in which all information on the Internet can be accessed consistently and easily. Because WWW has the greatest flexibility and most user-friendly interface, the OLIS effort focuses here.

WWW is geared toward hypermedia that allows selected objects to be expanded at any time to provide additional information (i.e., selected objects are links to other objects, such as text, sounds, images, and animation). The basic building blocks of WWW are the HyperText Transfer Protocol (HTTP) and the HyperText Markup Language (HTML). HTTP describes the communication protocol between clients and servers; HTML describes the format of the information pages transferred.

WWW capabilities are comparable to Gopher in that WWW is a client/server information system running on the Internet that provides quick and easy access to a wide range of servers. WWW performs the functions of Gopher but also supports hypertext links that permit the creation of more descriptive information for the end user. WWW provides a full screen of textual and graphical information with individual words, phrases, or icons acting as links (shown in Figure 5). The page-formatting capabilities of HTML can be used to create a wide range of point-and-click user interfaces that operate across the network.

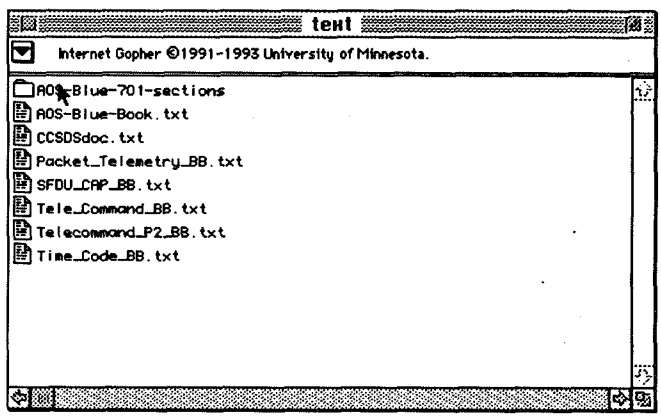


Figure 4 - TurboGopher Interface

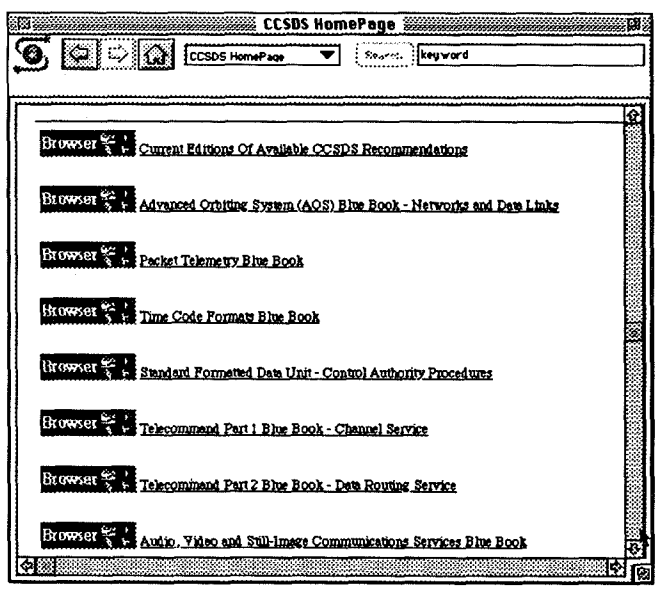


Figure 5. WWW Interface

The WWW model represents everything (document, menu, index, etc.) to the user as a hypertext or hypermedia object. Two navigation operations are available to the user: to follow a link or to send a query to a server. Two powerful features fall out of this hypertext model. One is that almost all other information systems can be represented in terms of WWW documents. The other is that the WWW system has an open but uniform addressing scheme that allows links to be made to any objects on WWW, WAIS, Gopher, FTP, Network File System, or Network News servers. A WWW user can interrogate WAIS indexes and Gopher servers. The hit list returned by a WAIS server (or any other query engine) is treated as a hypertext document with links to the documents found. Gopher menus (or any other hierarchical menu system) are represented as lists of items linked to other objects. The hypertext model also allows the user to put in a hypertext link, when needed, for background information.

NCSA Mosaic

Because WWW supports the functions of other types of clients and servers, this discussion focuses on WWW clients. One of the most popular WWW client applications is the free, public domain software, NCSA Mosaic, which is available for PCs, Macintoshes, and many UNIX workstations. It was developed at the University of Illinois Urbana-Champaign and can be picked up from most major anonymous FTP sites. With Mosaic and the proper support applications, users can explore WWW with full access to multimedia information, including formatted text, graphics, and sound. Through WWW, a user can access Telnet, FTP, WAIS, Gopher, and HTTP systems.

When a user starts Mosaic, the program attempts to connect to a preset known host; the initial host is at the NCSA, but each user can configure their own starting point. Once an initial page of information is downloaded, Mosaic operates in a true client/server fashion. The user sees on the screen a graphical point-and-click hypertext interface (shown in Figure 6). As the user browses through the hypertext tree, Mosaic automatically switches from system to system and protocol to protocol as needed.

Figure 6 shows the Mosaic interface and part of the MO&DSD HomePage. When the user clicks on the SpaceOps 94 hypertext link, Mosaic automatically retrieves information and displays the SpaceOps HomePage (shown in Figure 7).

If a special file type is referenced on a hypertext link, for example GIF format, Mosaic tries to invoke a GIF viewer to display the file after downloading is completed. GIF viewer software must be installed on the local system, and Mosaic must be configured to use it.

Mosaic displays the retrieved information in the large scrolling window, and links are followed by

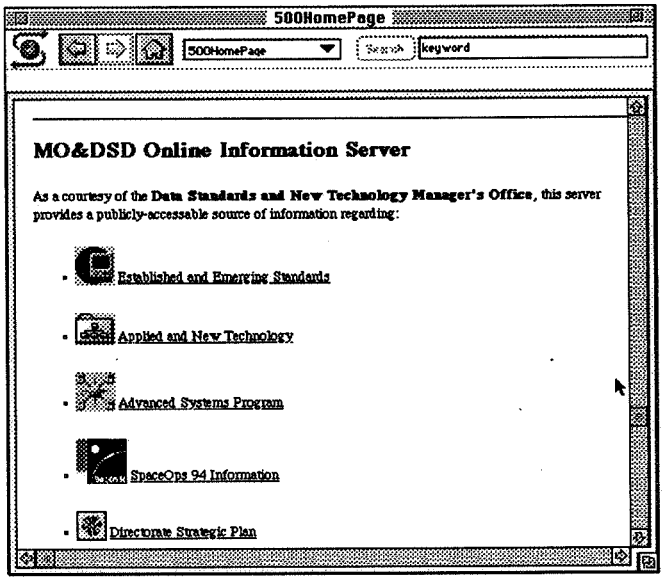


Figure 6. Mosaic Screen

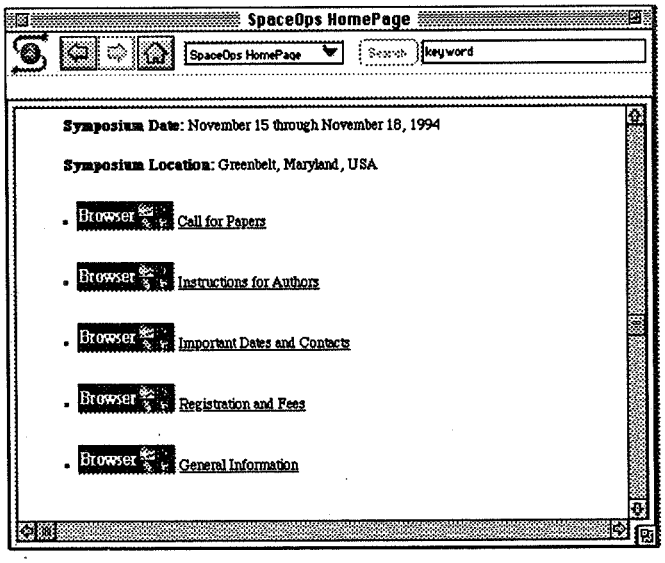


Figure 7. SpaceOps 94 HomePage

clicking on the underlined hypertext items. One of the most important features of the Mosaic client is its ability to save "bookmark" or "hotlist" references to a user's favorite information locations. Thus, the user does not need to know exactly which links were followed to find a particular information site. The user can access these locations again with a single operation using the bookmark or hotlist entry.

DOCUMENT FORMATS

One problem with information servers is the wide variety of possible formats in which data can be available. The MO&DSD server stores all of its documents in ASCII format, as well as in the document's original format. Currently, there are not many bit-mapped images stored on the MO&DSD server. Any images placed there are stored in either GIF or JPEG format.

Plain Text Document

Because ASCII format can deliver textual information to any end user via a wide range of access mechanisms, documents are usually loaded onto information servers in an ASCII format. However, the main problem with this is that all graphics and figures in the original (fully formatted) document are lost. Furthermore, many documents are prepared using desktop publishing packages with paragraph wrap and proportional fonts. When converted to monospace font with individual lines of text, these documents loose their enhanced appearance.

All ASCII documents on the MO&DSD OLIS have been indexed to the WAIS server to facilitate keyword searches. This allows a user to perform keyword searches on located documents of interest, scan the ASCII versions online, and retrieve the original formatted document if desired.

Fully Formatted Document

Original document can be stored in a wide range of formats [e.g., WordPerfect for Windows, Word for Macintosh, Rich Text Format (RTF), or PostScript]. Each of these formats can be used to deliver a fully formatted document. These files are usually converted to a single flat file, with a package such as BinHex, prior to being loaded on the OLIS. When trying to read and process fully formatted documents, the end user must have the proper application software to reconstruct an exact copy of the original document. Eventually, all fully formatted documents will likely be stored on OLIS in the following three formats:

- Plain ASCII text for searching and browsing (document may not look nice but all the text information will be there)
- Single PostScript file containing the whole fully formatted document including graphics
- Original word processor format

MAKING A MACINTOSH OR PC INTERNET CAPABLE

To use any of the Internet services described, a user must first gain access to the Internet. One way to access these services is through a connection to a computer that is a full host on the Internet. However, in this mode, the user needs to log in to that system and use the proper commands to access these services.

The most powerful way to access these services is to have a direct connection to the Internet. This can be done from most UNIX workstations, Macintoshes, or PCs. The connection may be via a local area network (LAN) or via a dialup phone line. The most powerful dialup access involves the use of Serial Line IP (SLIP) or Point-to-Point Protocol (PPP). These protocols operate between software on the user's computer and similar software on a SLIP or PPP server that the user can dial in to. The protocols then implement an IP connection over the dialup line, and the client applications then operate over TCP/IP, just like when the user is directly connected to a LAN. To get reasonable response with SLIP or PPP, a dialup connection of at least 9600 bits per second is required.

For any system to be connected to the Internet, three basic things are required:

- Connectivity to some point on the Internet
- Unique IP address for the user's computer
- Appropriate TCP/IP software and associated application software

A unique IP address for the user's system is provided by the administrator responsible for connectivity. A basic communication program with VT100 terminal emulation and download capabilities is also required on the computer. Some common packages

are Zterm for Macintoshes and ProComm or CrossTalk for PCs.

Macintosh Platform

For Macintoshes, the major item software required is MacTCP to provide TCP/IP support. GSFC has a site license, so this is readily available to any NASA-owned Macintoshes. Other client and support applications are available from Macintosh servers at GSFC or numerous anonymous FTP sites.

PC Platform

For PCs running Windows 3.1, the main software required is the Windows socket interface dynamically loadable library, "winsock.dll." This provides the interface between most applications and TCP/IP on the PC. The client and support applications are available from PC servers at GSFC or numerous anonymous FTP sites.

SUMMARY

With Internet access, modern information retrieval software, and connection to the MO&DSD OLIS, remote users can

- Easily retrieve well-established standards at a local Macintosh or PC
- Perform online browsing and retrieval of documents published for and by MO&DSD
- Search for current and past study activities on related topics within NASA or other government agencies prior to issuing a task
- Improve standards and technology awareness among peers to ensure a design of interoperable systems for cross-support environments
- Access information at thousands of sites worldwide, and make select information accessible by others

Additional potential benefits could be achieved in areas such as:

- Avoiding duplication of effort by being able to quickly and easily scan information of activities being performed across MO&DSD and, eventually, all of NASA
- Faster development of standards documents via immediate access to huge reference libraries and capability for rapid exchange of documents
- Significant reduction in travel costs through easy exchange of information and documentation over the network rather than by attending meetings.

REFERENCES

[1] Frank Hecker, *Personal Internet Access Using SLIP or PPP: How You Use It, How It Works*

[2] Ed Krol, O'Reilly & Associates, Inc., *The Whole Internet Catalog and Users Guide*

[3] Paul Lindner, editor, *Internet Gopher Users Guide* , University of Minnesota

[4] Adam Gaffin and Jorg Heitkotter, *Big Dummy's Guide to the Internet*

ACKNOWLEDGMENTS

The authors would like to acknowledge Dr. William Mackey, Mr. Steve Harris, and Ms. Sydney Buck for helpful comments on a draft version of this paper.

OMIT

ORBITAL MECHANICS PROCESSING IN A DISTRIBUTED COMPUTING ENVIRONMENT

Randolph Nicklas
INTELSAT

Paper Not Available

111193 N95-17608

THE REQUIREMENTS GENERATION SYSTEM: A TOOL FOR MANAGING MISSION REQUIREMENTS

35426 P-7

Sylvia B. Sheppard
NASA Goddard Space Flight Center
Greenbelt, MD 20771

ABSTRACT

Historically, NASA's cost for developing mission requirements has been a significant part of a mission's budget. Large amounts of time have been allocated in mission schedules for the development and review of requirements by the many groups who are associated with a mission. Additionally, tracing requirements from a current document to a parent document has been time-consuming and costly. The Requirements Generation System (RGS) is a computer-supported cooperative-work tool that assists mission developers in the online creation, review, editing, tracing, and approval of mission requirements as well as in the production of requirements documents. This paper describes the RGS and discusses some lessons learned during its development.

INTRODUCTION

One of the most important, time-consuming and expensive tasks for any mission is the development of mission requirements. For example, consider the contractor time expended for development of requirements for the Payload Operations Control Center (POCC) and the Command Management System (CMS) portions of two Small Explorer missions. For the Fast Auroral Snapshot Explorer (FAST) and Submillimeter Wave Astronomy Satellite (SWAS), the contractor person-years expended were 10.7 and 8.0, respectively (Mandl, 6/9/94). (The data do not include civil service time or time expended on requirements for other parts of these missions.) Similar expenditures for the Xray Timing Explorer (XTE) were estimated at between 15 and 18 person-years.

The Requirements Generation System (RGS) was developed to help automate the requirements process. The goal was to reduce mission schedules and costs associated with the creation and use of mission requirements information. We hypothesized that we could meet this goal by:

- increasing communication about all levels of mission requirements among the many individuals and groups of personnel contributing to a mission,
- providing automated assistance for the online development, editing, review, tracing and approval of requirements and the production of related documents and reports, and
- reusing sets of requirements across similar missions.

RGS CAPABILITIES

The RGS uses a distributed system architecture to encourage online work. The RGS was designed for existing desk-top platforms (i.e., Macintoshes and PCs). The sections below present the operations concept.

PRECEDING PAGE BLANK NOT FILMED PAGE 1344 INTENTIONALLY BLANK

Single Mission Database

The RGS uses a single mission database to maximize communication among mission personnel and to facilitate the online management of mission information. The mission-specific database is made available to all mission personnel from the beginning of a mission throughout its life cycle.

Figure 1 shows the mission requirements documents that are produced for a standard mission. The letters above each box list the organization (or person) responsible for producing the document. All levels of requirements and all requirements documents, commentary, and rationale are consolidated in the RGS mission database, thus reducing the time to locate, review and disseminate information. Higher-level requirements, such as those found in the Detailed Mission Requirements (DMR) document, and lower-level requirements, such as those found in the System Requirements Document (SRD), are included. This eliminates the need for traceability across separate documents and/or databases and allows for the production of reports that contain requirements at varying levels of detail. Additional documents (e.g., the Mission Requirements Request) are available online for reference and for the explicit traceability of DMR requirements to requirements in parent documents.

Although working online with the RGS does not eliminate the need for meetings to discuss issues, it can reduce the time needed to agree on a set of requirements. Mission personnel no longer need to wait for the release of a document to review requirements; they can be reviewed, and approved or rejected, individually. Piecemeal review can result in schedule efficiencies.

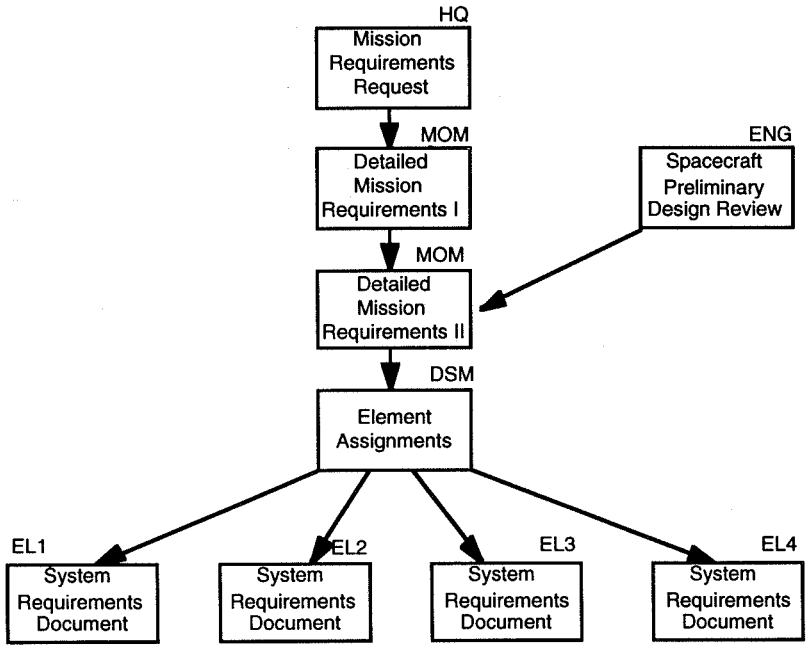


Figure 1. Requirements Document Hierarchy

Online Entry and Editing

The RGS provides a form-based graphical user interface for entering and editing requirements on-line. Requirements may be entered in any order. They are numbered by the requirements developer as they are entered and may be hierarchical. In Figure 2 the first requirement is numbered 4100-1, the second, 4100-1.1, etc. In this example the section of the requirements document (i.e., 4100) is appended to the beginning of the requirement number. This addition is optional.

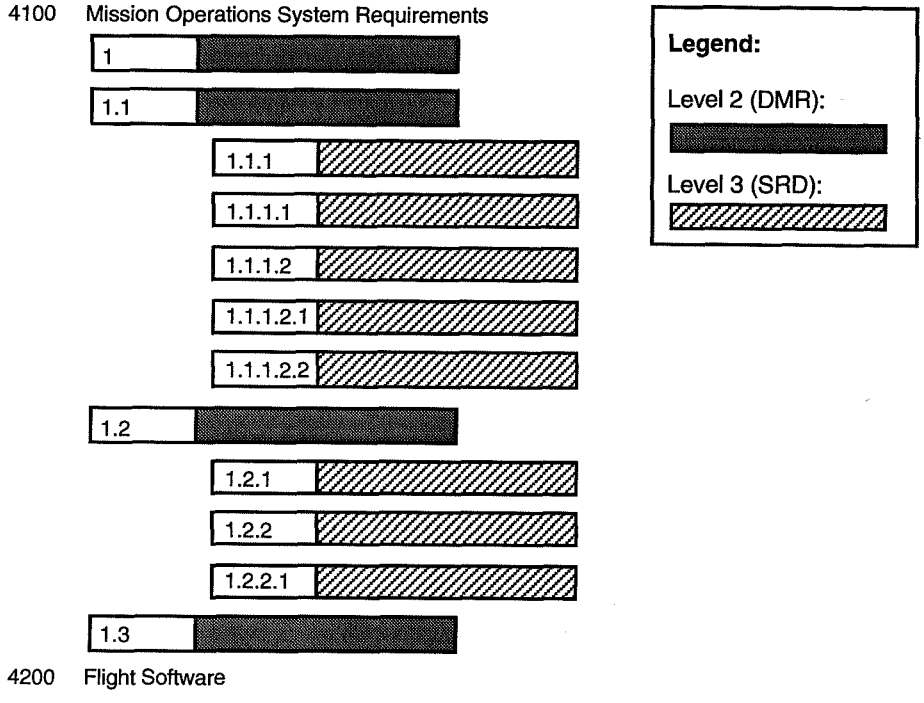


Figure 2 : Sample Requirements Hierarchy

Users may assign a "level" to each requirement. This number associates the requirement with a given degree of detail or a given document type (e.g., the requirements in the Mission Requirements Request might be designated Level 1, the Detailed Mission Requirements document Level 2, and the System Requirements Document Level 3). The requirement level is independent of the hierarchical number assigned to individual requirements. Mission personnel may determine the relationship between the levels and the hierarchy for their mission. Figure 2 shows one possible assignment.

A table of user privileges defines which users may enter and edit which sections and levels of the requirements. For example, the Mission Operations Manager (MOM) might assign the privilege of entering/editing the Level 2 requirements of Section 4000 to one group of requirements developers; Level 3 requirements for Section 4000 might be assigned to a different group of requirements developers.

On-line Approval/ Rejection

The RGS provides on-line, form-based capabilities for appropriate mission personnel to approve or reject a requirement and to attach associated rationale for their decisions. Approval privileges for a specific level of requirement may be assigned to any of the mission personnel. For example, for Level 2 requirements, approval could be assigned to only the MOM; to the MOM and Data System Manager (DSM); to the MOM, DSM and Element Manager (EM); or to some other combination of mission personnel. Different EMs may be assigned approval privileges for different sections of requirements as appropriate.

The RGS annotates each mission requirement in the database with a "status" that describes how far the requirement has progressed toward final approval. A clearly labeled status field distinguishes work-in-progress requirements from mission-approved requirements (Table 1). The instant availability of newly-developed requirements (i.e., draft or pending) provides access to the current thinking on issues and allows for speedier review and response from interested parties. Further, approving requirements individually (as opposed to waiting for the release of a set of requirements in a document) can speed up planning and design. Finally, the overall view of the status of the requirements aids management in their assessment of the progress that has been made at any point in time. Inadequate requirements in a certain mission area can be identified, and measures can be taken to correct any difficulties.

Table 1. Status Classifications for Requirements

Private	A requirement that is work in progress, visible only to the author (or the working group to which the author belongs).
Draft	A requirement that is work in progress, visible to anyone with access to the mission.
Pending	A requirement that has been submitted for approval. This requirement is considered "finished" but not accepted.
In Acceptance	A requirement that has been accepted by one, but not all of the parties responsible for approving the requirement.
Accepted	A requirement that has been accepted by each of the parties responsible for its approval.
Rejected	A requirement that has been rejected by at least one of the parties responsible for its approval.
Accepted with Contingencies	A requirement that has been accepted by each of the parties responsible for its approval, but to which the DSM has responded with exceptions.

On-line or Paper-Based Review and Reporting

Any requirement in the mission database may be reviewed by any mission user who has been granted privileges to access the database. An easy-to-use search mechanism allows users to filter the database and to select reduced sets of requirements for review. The selected requirements may be reviewed on-line or printed. Report contents can be defined by users using a simple selection technique.

Reviewers are also afforded an on-line "notes" capability for attaching commentary to individual requirements. The notes are then available for perusal by all database users.

HARDWARE AND SOFTWARE ENVIRONMENT

The RGS has a client-server architecture. A client-server architecture uses client machine(s) and server machine(s), along with the underlying operating system and interprocess communication systems, to form a composite system that allows the distributed access, management, analysis, and presentation of information.

The RGS supports both PC and Macintosh computers as client machines. Future plans include running on UNIX platforms. A Compaq System Pro/LT comprises the server portion of the RGS hardware configuration. This server houses all the RGS databases, support documentation, and database software.

The RGS server resides on the GSFC Center Network Environment (CNE). GSFC users access the RGS server from their workstations via this network. Local off-site users access the CNE via a T1 line, while off-site users not local to GSFC access the CNE via the Program Support Communications Network (PSCN) Internet.

The RGS was developed using two Commercial Off The Shelf (COTS) software packages, OMNIS 7 and SQL Server. OMNIS 7,

manufactured by Blyth Software, is a graphical user interface package that was used to develop the front-end portion of the RGS. The front-end executes on the client machines and provides the mechanism for users to interface with the RGS database. The front-end is responsible for soliciting queries or directions from the user for purposes of data update, analysis and retrieval and for presenting the results of queries and commands to the user. The front-end may also perform data analysis on the query results returned from the server.

SQL Server, a relational database management system marketed by Microsoft, Inc., was used to develop the RGS databases. The functions of this server component of the RGS custom software are to respond to user queries issued by the client machines and to manage the RGS requirements databases and document library.

DISCUSSION

The primary goal in developing the RGS was to produce a system that improves the development of mission operations. To date the RGS seems to be fulfilling that goal. The system is currently being used for five missions, and there are plans to use it on another six missions. Estimates are that requirements costs will be cut at least 50%, with even larger savings for missions that are similar enough to reuse major portions of the mission databases (Mandl, 1994).

A second reason for developing the RGS was to learn about client-server, computer-supported-cooperative-work (CSCW) systems. This section discusses some lessons learned from development of the RGS.

Lesson 1: Plan on Becoming a Full Service Organization

Computer-supported cooperative-work systems are only useful when there is a critical mass of users. For example, the major hurdle in using an electronic meeting scheduler is that all participants must be able to read and respond or the scheduling activity is hampered (Grudin, 1990). Similarly, deployment of the RGS as a CSCW system would be useless if mission personnel couldn't access requirements online.

Providing physical access to the RGS for all mission personnel required more resources than originally anticipated. Originally, the RGS team had expected to supply the RGS application, the COTS packages (OMNIS and SQL Server), training, a user's guide, and an RGS hot line service for responding to questions. Additionally, we planned to provide for maintenance of the server and the centralized database for each mission. We later determined that we had to become a "full service" organization. Many of the end users, spread across the Center and beyond, did not have the expertise to acquire and install the software to run a client-server system. Client-server software is more difficult to install than software packages that reside on an individual workstation. Generally Macintosh installations generally were done quickly, but PC installations often required extensive analysis. Sometimes it took several hours to get the RGS installed. Conflicts with existing mail and other resident user packages were the rule as opposed to the exception.

Additional chores included wiring offices to get users networked to the CNE, and supplying and installing communications software and Ethernet cards. In the case of off-site contractors who did not access the RGS server through the Internet, we provided expertise in dealing with the telephone company to obtain communication lines.

The extra services were time-intensive and expensive. Had we not had resources to expend for these tasks, the whole project might have failed.

Lesson 2: Employ Users' Groups with Full Representation

Working with users' groups is an integral part of the methodology of the client-server development team. We established an RGS Users' Group at the beginning of the project and met monthly thereafter to determine the requirements for the system. The Users' Group discussed the types of users and the capabilities each would need to do the mission's work. In some cases we used detailed scenarios of the tasks to be done by the individual types of users in order to determine that we correctly understood the requirements.

The RGS Users' Group was very helpful in defining requirement and developing a design. However, one type of user, the Data System Manager, was not represented in the beginning. Users who were present were not able to represent adequately the functions needed by the DSM. For a later release of the RGS we redesigned several features to include those functions. We concluded that an efficient development methodology for CSCW projects absolutely requires the active involvement of every type of user, regardless of their amount or type of use.

Lesson 3: Design for Flexibility

One original goal of the RGS was to design a system specifically tailored to handle the Goddard Mission Operation and Data System Directorate's method of developing and managing mission requirements. The reasoning was that mission personnel were accustomed to a largely paper-based process, and convincing them to adopt an automated system could be done best by making as many of the elements of the automated process as familiar as possible. Other requirements systems that were reviewed did not provide the specific kinds of functions that are needed to satisfy the Mission Operation and Data System Directorate process. To this end the RGS developers and potential users worked together to define the user types and the functions each would perform. The RGS team incorporated those functions into the design.

The first two releases of the RGS were successful because of this approach. However, further discussions, often with the users' groups, highlighted the need for differences in capabilities from mission to mission. Examples include the desire to change the privileges of users and user types, to use the RGS for different types of documents (as opposed to the DMR for which it was originally designed), to tailor the approval processes for individual missions, to create different document structures and formats, and to create traceability to a wider range of parent documents. In short, the original description of RGS capabilities was well-defined and rather rigid. As more users became involved, they requested more flexible capabilities to suit their mission's style of operating.

One approach to making the needed modifications would be to hand-tailor the RGS software for each mission. This approach was rejected because of the inherent software maintenance costs, the problems of managing multiple versions of the same software, and the limitations of what can be changed in software with a short turnaround time.

Instead we chose to design generic capabilities. The latest version of the RGS has a flexible set of functions that can be tailored to a particular mission's need by the mission personnel. Beginning with Release 3, mission personnel may configure the RGS, without assistance from the developers, to allow the definition of any of the following:

- any number of mission-specific user types (e.g., a requirements developer, MOM, DSM, EM, read-only user, and other mission-defined users).
- a mission-specific structure for a requirements document (showing what sections are to be included).

- mission-specific requirement levels, allowing for a greater level of detail beyond the standard three document levels (i.e., MRR, DMR and SRD).
- mission-specific acceptance privileges (determining what approvals are necessary for what sections and levels of requirements).

These flexible capabilities made the RGS a more general purpose tool. We also expect them to reduce the software maintenance required for the RGS.

Lesson 4: Plan to Deal with Changes in Work Flow

In the past, mission personnel developed requirements in a sequential fashion, largely completing and approving higher-level requirements documents before lower-level requirements were defined. Often one group of requirements developers wrote higher-level requirements and another group lower-level ones.

Use of an open database promotes changes to this traditional work flow. Requirements at any level can be entered at any time. Mission personnel can add information as soon as it becomes available. One section of the requirements can be completed at the lowest level before another section is begun at a higher level. Additionally, documents per se become less important. Requirements can now be reviewed and approved individually, or in sections, as opposed to at the "document" level. While these changes can impact the schedule positively, this flexibility may be upsetting to team members who are used to a more structured process. Managers need to be prepared to establish procedures to deal with the changing work arrangements that ensue from automation of this type.

REFERENCES

Grudin, J. (1990). Groupware and cooperative work: Problems and prospects. In B. Laurel (Ed.), *The art of human-computer interface design* (pp. 171-185). Reading, MA: Addison-Wesley.

Mandl, D. (1994, June 9). *Cost (manyrs) to gather requirements for MOC's with and without RGS and SDT methodology.* Informal communication.

ACKNOWLEDGMENTS

The author would like to thank the many people who contributed to the development of the RGS. Mark Stephens is the RGS Project Manager; Lisa Dallas served in that capacity before the birth of the twins. The two software development teams are Linda Cingel, Rachel Campbell, Lou Fenichel and Barbara Wrathall of Computer Based Technology Incorporated and Gary Chatters, Phillip Wolf and Teresa Bleser of Century Computing. Special thanks go to William Guion, Randy Harbaugh, Richard Harris and William Macoughtry for their encouragement and support.

111194 N95-17609

354327 p. 8

AN OPPORTUNITY ANALYSIS SYSTEM FOR SPACE SURVEILLANCE EXPERIMENTS WITH THE MSX

Ramaswamy Sridharan, Gary Duff, Tony Hayes, and Andy Wiseman
MIT Lincoln Laboratory

Abstract - The Mid-Course Space Experiment consists of a set of payloads on a satellite being designed and built under the sponsorship of Ballistic Missile Defense Office. The MSX satellite will conduct a series of measurements on phenomenology of backgrounds, missile targets, plumes and resident space objects (RSOs); and will engage in functional demonstrations in support of detection, acquisition and tracking for ballistic missile defense and space-based space surveillance missions. A complex satellite like the MSX has several constraints imposed on its operation by the sensors, the supporting instrumentation, power resources, data recording capability, communications and the environment in which all these operate. This paper describes the implementation of an opportunity and feasibility analysis system, developed at Lincoln Laboratory, Massachusetts Institute of Technology, specifically to support the experiments of the Principal Investigator for space-based surveillance.

1.0 INTRODUCTION

The Mid-Course Space Experiment consists of a set of payloads on a satellite being designed and built under the sponsorship of Ballistic Missile Defense Office (formerly, Strategic Defense Initiative Office) of the Department of Defense. The major instruments are :

1. A set of sensors being built by Utah State University, called SPIRIT 3, covering the spectral range from 4.2 μ to 20 μ in the long wave infra-red band.
2. A set of sensors operating in the ultraviolet and visible wavelengths (0.1μ - 0.9μ), called UVISI, being built by Johns Hopkins University's Applied Physics Laboratory.
3. A broad-band visible wavelength sensor (0.4μ - 0.9μ), called SBV, being designed and built by Lincoln Laboratory, Massachusetts Institute of Technology.
4. A set of sensors for monitoring and measuring contamination of the mirrors and the space around the MSX.

The satellite bus is being built by JHU/APL who is also acting as the integrator for all the sensors and associated systems. The MSX satellite, shown in Fig. 1, is due for launch in late 94 from the Vandenberg launch complex. It will be in a nearly sun -synchronous orbit with an orbital inclination of 99^0 and an orbital period of 103 minutes.

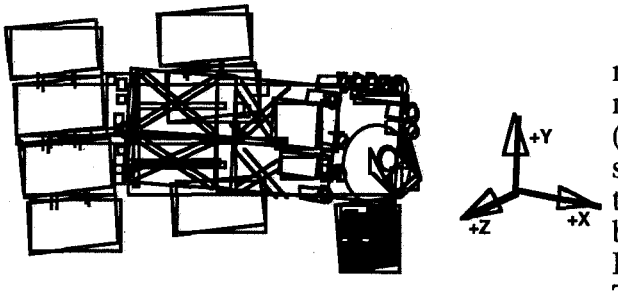


Fig. 1. MSX spacecraft

The MSX satellite will conduct a series of measurements on phenomenology of backgrounds, missile targets, plumes and resident space objects (RSOs); and will also conduct functional demonstrations in support of detection, acquisition and tracking for ballistic missile defense and space-based space surveillance missions. Eight Principal Investigators are associated with the MSX project. The area of interest in this paper is the surveillance of resident space objects from a space-based platform. The Principal Investigator for Space Surveillance is located at Lincoln Laboratory, Massachusetts Institute of Technology. The SBV is the major instrument being used in space-based satellite surveillance experiments.The command and control center for the SBV, called SPOCC, is also at Lincoln Laboratory. All space surveillance experiments are conducted by the Surveillance PI using the resources of SPOCC.

PRECEDING PAGE BLANK NOT FILMED PAGE 1352 INTENTIONALLY BLANK

The conduct of experiments with the MSX has a long planning cycle, similar to NASA scientific satellites. A key aspect of experiment planning is the analysis of the opportunities available for conducting any experiment, taking into account geometric and spacecraft constraints. A software system has been built in SPOCC to support the opportunity analysis for space-based surveillance experiments. We describe, in this paper, the process of computing the opportunities for and analyzing the feasibility of space-based surveillance experiments with the MSX and illustrate it with an example.

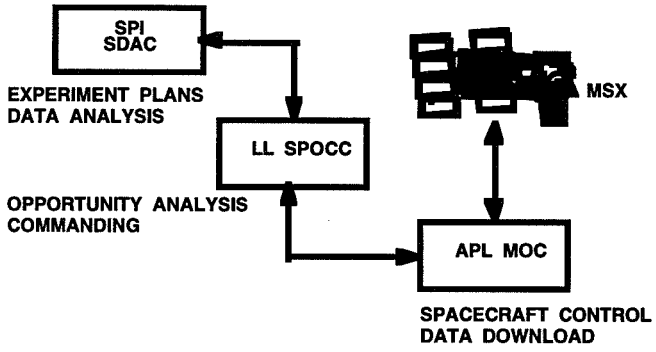


Fig. 2 : Data Flow for Surveillance Experiments

The SBV Processing, Operations and Control Center, located at Lincoln Laboratory, Massachusetts Institute of Technology, generates the necessary commanding for the MSX and its sensors for all space-based space surveillance experiments designed by the PI for Surveillance. SPOCC also converts and calibrates the returned science data from the SBV before turning them over to the SPI's Surveillance Data Analysis Center. The data flow is illustrated in Fig. 2. JHU/APL's Mission Operations Center is in overall charge of the spacecraft.

2.0. MSX AND ITS INSTRUMENTS

It is necessary to have a working knowledge of the MSX spacecraft, its sensors and their interaction to understand the functioning of the Opportunity Analysis System.

Figure 1 shows the body reference axes defined for the MSX spacecraft. All major sensors on the MSX have their fields of view substantially co-aligned along the +X-axis.

The MSX will be launched into a near-sun-synchronous, 99 deg. inclination orbit with an orbital period of 103 minutes. The satellite will have shadow periods as long as a third of the orbit due to the initial value of the right ascension of the ascending node. It carries a set of Nickel-Hydrogen batteries for powering the spacecraft operations during eclipse. The batteries are recharged by the solar panels.

The MSX carries two redundant tape recorders for high bandwidth data recording. The tape recorders are operated singly (or in parallel for critical data). Each unit is capable of recording 36 minutes of data at 25 Mb/s or 180 minutes of data at 5 Mb/s.

The SPIRIT 3 infrared sensor has a dewar containing solid hydrogen to cool the focal planes to 10^0K. The lifetime of the sensor is critically affected by the rate of dissipation of the Hydrogen. This sensor writes out its data almost entirely to the tape recorder.There is a set of ultra-violet and visible wavelength imagers and spectrometers on board, collectively called the UVISI. These instruments also use the tape recorder for storage of experiment data.

The SBV is the third major sensor on board the MSX. This sensor is comprised of a 6-inch aperture off-axis rejection telescope, a camera with 4 CCD chips with a total field-of-view of ~6^0x 1.4^0, a Signal Processor for data compression and an Experiment Controller. The Experiment Controller controls SBV operations and has a large data buffer to store science data processed by the Signal Processor. Raw science data can be written out to the tape recorder.

The MSX supplies power, data handling, telemetry, commanding and pointing capability for all the sensors on board. Expected pointing accuracy is of the order of 0.1^0 on board around all axes. The attitude processor data can be further processed in the ground-based Attitude Processing Center to yield a pointing/attitude knowledge of a few arcseconds.

The MSX weighs ~6000 lbs. on the ground and is due to be launched on a Delta 2 launch system in Nov. 94. The launch will be from the Vandenberg Air Force Station.

3.0. SPOCC SUPPORT OF SURVEILLANCE EXPERIMENTS

SPOCC, as mentioned earlier, is the mission planning node for all experiments of the Principal Investigator for Surveillance.

The major tasks of the mission planning system in SPOCC are:

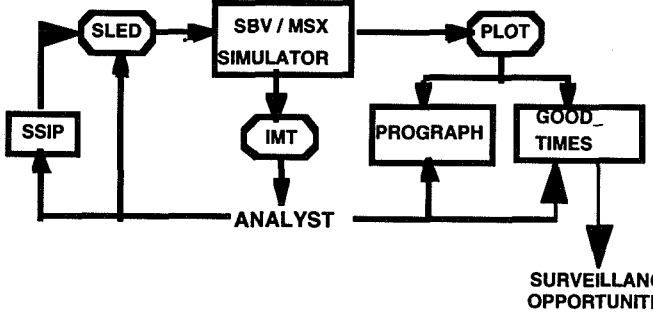
1) to permit a study of the opportunities available for an experiment; and
2) to generate the necessary commands to the sensors and the spacecraft to execute the experiment.

This report concentrates on the opportunity analysis.The major question answered by opportunity analysis system is:

When can an experiment be conducted?

The answer to this seemingly simple question is complicated by the following requirements:

1. Opportunities have to be computed for a month, six weeks before the start of the month, due to the long planning cycle for the MSX.
2. A feasible opportunity implies that an experiment, as defined, can be conducted within the available time and without violating constraints on the spacecraft or the sensors.
3. Further, the spacecraft resources consumed (both renewable and non-renewable) by the experiment must be within limits allocated to the experiment.



SLED : SBV LANGUAGE FOR EXPERIMENT DESIGN
IMT : INSTANTIATED MISSION TIMELINE
SSIP : SPACE SURV. INTERFACE PROCESSOR

Fig. 3 : Opportunity Analysis Software System

A software system has been built in SPOCC to conduct opportunity analysis for surveillance experiments. Figure 3 captures the essential components of the System.

This system is invoked by a file of commands in a high level interface language called Surveillance Language for Experiment Design (Ref. 1). The SLED code can be written by a user, which is the predominant mode for most experiments involving the collection of data on a single resident space object (RSO). SLED code can also be generated automatically by the Space Surveillance Interface Processor (SSIP), which is the mode for multi-RSO experiments and for experiments which have to be conducted with short notice (called Quick Reaction Events).

The components of the Opportunity Analysis System are described below.

3.1. SLED

The Surveillance (or SBV) Language for Experiment Design is a structured high-level language for describing space-based surveillance experiments with the SBV; and to a limited extent, with the SPIRIT 3 and the UVISI sensors. Principal characteristics of SLED are:

1. The language has a precise syntactic and logical structure.
2. The language permits description of a space surveillance experiment independent of detailed timing information.
3. The syntactic and logical structure is expandable to adapt to new requirements.

The fundamental requirement in the design of the SLED is to free the experimenter from the details of timing and control of the MSX and instead let him/her concentrate on the objectives and the logical design of the experiment.

3.2. The Simulator

The Simulator is the heart of the Opportunity Analysis System in SPOCC.

The Simulator parses and compiles the SLED code into a detailed event timeline which models the temporal flow of the experiment as a set of timed events for the sensors and the spacecraft. Each event implies a state change for the MSX and/or its instruments. The cost of each event is also accumulated by the Simulator. A block diagram of the Simulator functions with inputs and outputs is shown in Fig. 4.

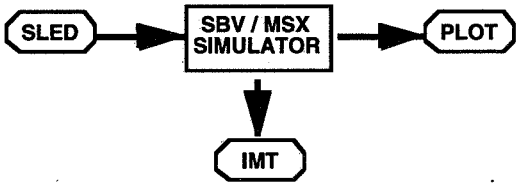


Fig. 4 : Functions of the Simulator

3.2.1. The Parser

All SLED code is parsed by the front end of the Simulator. The functions of the Parser are :

1. Check the SLED code for syntactic and logical consistency.
2. Check the implied modes of operation of the sensors for logical consistency.
3. Create ordered tables of the modes of operation of the sensors, and where applicable, their components.
4. Create ordered tables of the mode of operations of the MSX.

The tables built by the Parser are used by the entire Mission Planning System.

3.2.2. Spacecraft and Sensor Modelling

The Simulator has to create all the necessary events to render the MSX ready to collect data – including all turn-on/off of components and the attitude maneuvers necessary to re-orient the satellite before, during and after the experiment. The Simulator also creates events for commanding the sensors to collect, process and store the experiment data. Further, the Simulator concatenates all these events into an ordered timed set for further processing by the rest of the mission planning system.

3.2.3. Geometrical Modelling

The instruments on the MSX, and the MSX itself, impose several geometrical constraints on the pointing and orientation of the spacecraft. These constraints are divided into hard (potential

for damage) and soft (high resource usage). Significant constraints are summarized below.

Control of cryogen depletion on the SPIRIT 3 instrument is required to prolong its useful life. Hence the thermal input into the telescope axis from the sun and the earth must be kept low. This results in the following pointing constraints (see Fig. 1 for body reference axes):

1) The X-axis (which is the common telescope axis) should be kept away ($> 30^0$) from the sun direction(hard) and $> 63^0$ from the nadir (hard) .
2) The +Y-axis, which defines the open or exposed side of the dewar containing the cryogen, should be kept $> 90^0$ away from the sun (soft).
3) The –Y-axis which defines the convex side of the SPIRIT 3 sunshade, should be kept $< 90^0$ from the nadir (soft).

Other major pointing constraints are :

1) The UVISI sensors require that the +X-axis be not pointed near the sun(hard) or at the solar specular point on the earth(hard) when they are on.
2) The SBV telescope field of view cannot be pointed at the sun for more than 15 minutes (hard). The SBV should be pointed at least 25^0 away from the sun for good data (soft).
3) The –X axis of the spacecraft cannot be pointed at the sun directly for fear of heating the battery (soft).

The Simulator models all the angles relevant to these geometrical constraints during a data collection. The precise values for the constraints are yet to be refined. The MOCARH, referred to earlier, will be the formal document for operational constraints.

The Simulator also propagates the orbit of the MSX and of any RSOs requested. Geometrical visibility of the RSOs and solar illumination of both the RSOs and the MSX are computed. Further various relevant phase and aspect angles are calculated. Finally, visibility from a set of ground-based downlink contact stations is also computed.

3.2.4. Resource Usage Constraints

The Simulator has a detailed model for the power usage on board and the power generated by the solar panels. Knowing the initial state of the battery, the depth of discharge is computed.

The tape recorder on the MSX and the data memory in the SBV are finite resources for recording science data. The Simulator monitors their usage and either terminates the experiment, in the case of the tape recorder, or requires a downlink contact, in the case of the SBV memory, when no more data can be written out.

The SPIRIT 3 sensor has a finite quantity (~900 liters) of solid hydrogen for cooling its focal planes. A cryogen depletion model has been developed by the instrument manufacturers that predicts the quantity of hydrogen lost as a function of thermal input from the earth and the sun. The Simulator uses this model to compute cryogen depletion while simulating an experiment.

The Simulator has a thermal model for key parts of the spacecraft, viz., the SPIRIT 3 baffle, the tape recorder heads and the battery. The baffle temperature affects the sensitivity of the SPIRIT 3 sensor significantly. The other components have been identified by their manufacturers as being prone to damage due to large temperature excursions. Hence, the temperature rise of these components during an experiment is estimated by the Simulator.

3.3. PROGRAPH Display System

The Simulator creates a number of output products. Of relevance to the Opportunity Analysis, however, is the following.

The Simulator writes out into a file all the resource usage and geometrical computations during the data collection event simulated. The PROGRAPH processor displays all of these variables in graphical form on a display. This enables the user to visualize the experiment cost and modify the SLED code appropriately to reduce the cost if necessary.

PROGRAPH is implemented with a commercial software package called PVWAVE. All variables are plotted on the against elapsed time during the data collection. The user can select any graph(s) to be expanded and displayed.

Visual analysis is aided by the following capabilities of PROGRAPH:

1) Display of a selected graph.
2) Display of selected variables in a graph.
3) Re-scaling of x and y axes on the graph (time elapsed during the data collection event is always the x-axis in the graph).

Generally, the analyst uses the X-axis constraints on the MSX and the power usage graphs as key indicators of the feasibility of an experiment.

3.4. GOOD_TIMES Process

The final step in the Opportunity Analysis process is to examine the values of the various parameters displayed by PROGRAPH and pick intervals of time when the experiment can be conducted while observing all constraints and not exceeding allocated costs.

The input data to PROGRAPH can be automatically analyzed by a process called GOOD_TIMES. Apart from the PROGRAPH data, a task file drives the GOOD_TIMES processor. The task file specifies the range of values permitted for each parameter. When invoked, the GOOD_TIMES process examines the entire PROGRAPH data and finds time intervals that satisfy all the constraints in the task file. The output is captured in a Surveillance Opportunities File which is the major data product produced by the Opportunity Analysis System and sent to the Mission Operations Center at JHU/APL.

4.0. AN EXAMPLE

A geosynchronous surveillance experiment will be taken as an example here. The requirement, as set by the Surveillance PI's experiment plan, is to survey any part of the geosynchronous belt for 3 consecutive hours using the SBV and its on-board signal processor . The geosynchronous belt is quite heavily populated with resident space objects. A space-based optical sensor like the SBV has the ability to efficiently survey and collect data on all the RSOs in the belt, unlike a ground-based sensor, which is restricted in coverage by geographic location and inhibited in its operation by daylight and clouds. Hence a geosynchronous surveillance experiment is key to demonstrating the utility of space-based surveillance.

In the present example, the search strategy chosen was to point at a location in right ascension in the geostationary belt and vary the declination in steps between $+3.5^0$ and -3.5^0. Fig. 5 depicts the search strategy.

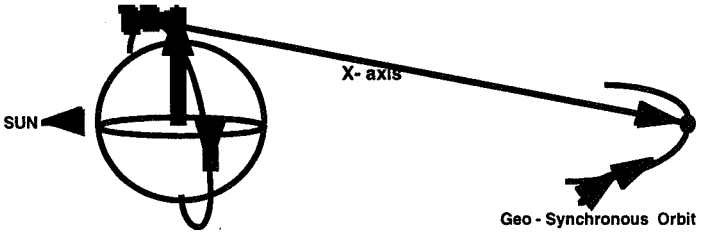


Fig. 5. Geosynchronous Search

Stare at fixed point in Geosynchronous Belt

The MSX is due for launch in late '94. However, for the purpose of this study, the launch date was chosen to be Oct 93. Orbital elements were specified by the MSX

program.

Two optional roll laws for the MSX are used in this example:
1) the –Y axis is pointed as close as possible to the nadir (–Y-TO-EARTH)
and 2) the –Y axis is pointed as close as possible to the sun (– Y-TO-SUN).

Roll law refers to the rotation of the MSX about its common pointing or +X - axis. The first roll law minimizes the thermal input into the SPIRIT 3 telescope from the earth because the convex side of the earth(sun)shade (its bottom) faces the earth all the time as the MSX orbits the earth. Thus the cryogen is conserved. The second roll law enables the solar panel axis (the Z-axis) to be as close to perpendicular to the sun as possible because the MSX orbit is near-polar and near-normal to the earth-sun line. Thus the solar panels can be rotated about the Z-axis for maximum solar illumination and power generation. These are the type of soft constraints that an analyst examines to assess the resource usage of the experiment. The effects of the roll laws on the cost of the experiment are illustrated below.

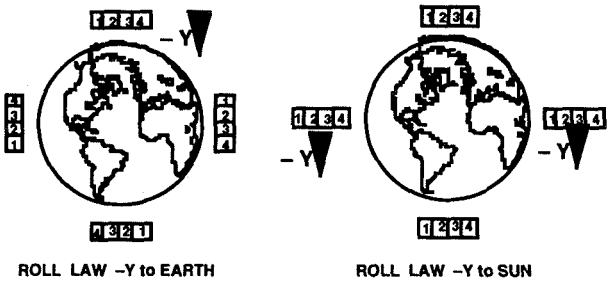


Fig. 6 : Roll Laws and SBV Focal Planes

Fig. 6 shows the orientation of the SBV focal plane, which is extended along the Z-axis, under the two roll laws. The focal plane consists of 4 CCDs, each with 420x420 pixels, butted against each other. The individual field-of-view is 1.4^0 x 1.4^0. The total FOV is ~6.6^0x1.4^0 because of distortion effects of the off-axis reimaging optical system. The figure shows that with the –Y-to - earth roll law, the focal plane is rolled as the MSX circumnavigates the earth in its orbit with the –y - axis maintained pointing at earth center. Such an orientation reduces the thermal input from the earth into the SPIRIT 3 aperture (+X - axis) thus helping to keep it cold. The focal plane, on the other hand, stays invariant in space under the –Y - to -sun roll law. Such an orientation allows the solar panels to be rotated about the Z-axis for maximum power production, but at the price of greater thermal input from the sun into the +X-axis with the consequence of higher cryogen depletion.

Figure 7 shows the estimated depth-of-discharge of the battery as a result of the experiment being conducted for 24 hours with the two roll-laws. There is a periodic component in battery depth of discharge that arises from the orbital period of the satellite - recall that the MSX is in earth shadow for part of the orbit. There is a secular component, clearly evident in the graph for the roll law "– Y-TO-EARTH" that is due to the inadequate recharging of the battery in the illuminated part of the orbit. A requirement is that the battery be not depleted by more than 40% routinely with 60% as an extreme limit. It is evident that the experiment has to be cut short at ~28000 seconds with the first roll law. However, when the "–Y-TO-SUN" roll law is used, the solar panels can be rotated for maximum power production and hence the depth-of-discharge does not have a secular component. Therefore, the experiment can be continued indefinitely from a power perspective.

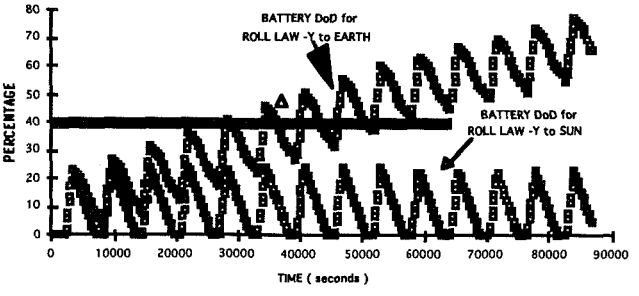


Fig. 7 : Battery Depth of Discharge

The gain in power, and in the battery depth-of-discharge, however comes at a price. The temperature of the baffle of the SPIRIT 3 telescope (inside the sunshade shown in Fig. 1) is affected by the thermal input into the aperture; and, further, the cryogen depletion is related to the thermal input and the baffle temperature. Also, the baffle temperature directly affects the noise

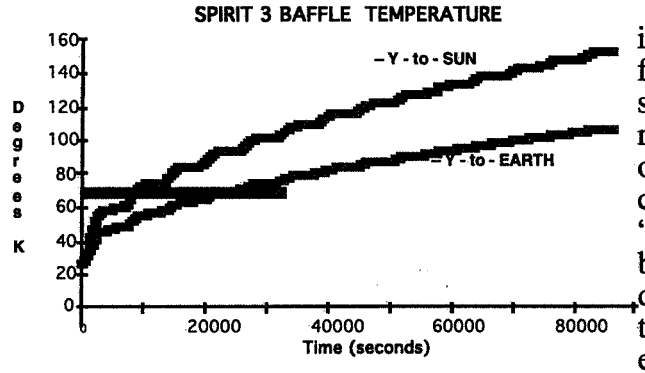


Fig. 8 : Baffle Temperature

input into, and hence the quality of, the data from the focal plane of the SPIRIT 3. Fig. 9 shows that, while conserving battery power, the roll law "-Y-TO-SUN" causes a more rapid rise of the baffle temperature, and consequently, cryogen depletion. battery power, the roll law "–Y-TO-SUN" causes a more rapid rise of the baffle temperature, and consequently, cryogen depletion. Also, the baffle cools very slowly and thus the data quality for any subsequent experiment using the SPIRIT 3 is degraded for a longer time than if the roll law "–Y-TO-EARTH" were used.

The orbit of the MSX is not quite sun-synchronous. It precesses with respect to the sun slowly. Hence, the power balance between the solar panels and the battery changes over a period of time. Figure 9 illustrates the effect of the time of instantiation of an experiment on the power balance. A 24 hour long geosynchronous experiment conducted in July 94 depletes the battery more rapidly than if it were conducted in Jan. 95. The difference is entirely due to the fact that the periods the MSX is in earth shadow are much shorter on the latter date.

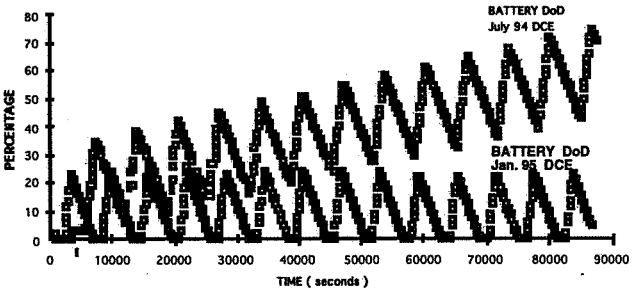


Fig. 9 : Effect of Time of Instantiation

5.0 SUMMARY

A successful Opportunity Analysis System has been developed in SPOCC to facilitate the scheduling of the Surveillance Principal Investigator's experiments on the MSX. The system uses knowledge of relevant geometries and spacecraft and instruments constraints to model the cost of conducting an experiment. The system has been tested extensively and fully supports the long experiment planning process associated with the MSX.

BIBLIOGRAPHY

1. John D. Mill, et al: "The Midcourse Space Experiment: Introduction to the Spacecraft, Instruments and Scientific Objectives", JSR (accepted for publication).
2. R. Sridharan, et al: "An Opportunity Analysis System for Space Surveillance Experiments with the MSX", Technical Report No. 1011 (to be published).

111195 N95-17610

Matrix Evaluation of Science Objectives

354329 p.8

Randii R. Wessen

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California U.S.A.

Introduction

The most fundamental objective of all robotic planetary spacecraft is to return science data. To accomplish this, a spacecraft is fabricated and built, software is planned and coded, and a ground system is designed and implemented. However, the quantitative analysis required to determine how the collection of science data drives ground system capabilities has received very little attention.

This paper defines a process by which science objectives can be quantitatively evaluated. By applying it to the Cassini Mission to Saturn, this paper further illustrates the power of this technique. The results show which science objectives drive specific ground system capabilities. In addition, this process can assist system engineers and scientists in the selection of the science payload during pre-project mission planning; ground system designers during ground system development and implementation; and operations personnel during mission operations.

1. Approach

The basic approach has both the science community and the ground system define a set of matrices. The science matrices define the main objectives of the mission, who will collect them and when. The ground system matrices define the characteristics that drive ground capabilities and an estimate of when each service can be provided. Together, the set of matrices represents a powerful analytic tool.

To begin, the first matrix created (and the most fundamental) is the matrix that explicitly establishes which science objectives can be met by each investigation. This matrix known as the "Science Objectives vs. Investigation" matrix, ensures that the objectives of the missions can be met by the selected investigations.

Once the "Science Objectives vs. Investigation" matrix is completed, a second matrix, which establishes the times during the mission (i.e., epoch) where each objective is captured is created. This matrix identifies the importance of each epoch based on the acquisition of science objectives. Epochs are determined either by orbital events (e.g., bow shock crossing, satellite closest approach, etc.) or by investigation characteristics (e.g., the time when the target body fills the narrow angle camera field-of-view).

Next, the science community creates a matrix which defines "types of observations" the spacecraft must perform to obtain the desired science. The observation type only represent activity that is external to the science instruments. It is assumed that instrument internal commands can always be sent to the spacecraft when two-way communications has been established.

The last matrix generated by science defines which ground system resources are needed for each observation types. This matrix, known as the "Operations Characteristics vs. Observation Type" matrix, allows the science community to, independently from the Ground System (GS), evaluate which ground resources are needed by their investigation.

During the development of these matrices, the GS defines its own tables. The first of these defines the mission operation characteristics (i.e., those characteristics that drive mission ops cost) and their associated dynamic range.

Next the GS generates the "Operations Characteristics vs. Orbital Segment" matrix. This matrix is the GS's best estimate of how its ground resources will be used during the course of the mission. It show what level of resources are needed for each segment of the mission. Once generated, the observation types (based on the GS's characteristics) are compared to this table. The results show which science objectives are in jeopardy by the current allocation of GS resources.

By identifying conflicts early, the GS and science community can negotiate how to reallocate resources to design a ground system that is within budget, consistent with mission plans and responsive to the needs of the science community.

2. 1 Science Matrices: Science Objectives vs. Investigation

The first set of matrices captures the mission's science objectives. These objectives usually fall into one of four categories: atmospheres, magnetospheres, rings and satellites. In some cases, categories may need to be added, removed or modified. In the Cassini example, the addition of a Titan category is required. In each category there are approximately five to ten explicit science objectives.

This set of matrices have one matrix for each category. Each matrix shows which objectives are captured by which investigation (see fig. 1 "Cassini Titan Science Objectives"). During pre-project development, the proposed generic instrument payload (i.e., imagers, spectrometers, radiometers, mass spectrometers, magnetometers, etc.) are evaluated against their corresponding science objectives. This ensures that the proposed instrument payload captures all the science that the spacecraft is designed for, confirms that no proposed investigation is redundant with another and that no investigation exceeds the scope of the mission.

During development the selected payload is again evaluated against the science objectives. This confirms that between pre-project design and project start (and the selection of investigations) the desired set of science objectives are indeed captured by the spacecraft's payload. Once evaluated, these matrices are placed under project change control to ensure that the contributions from each investigation are explicitly stated and that their requirements do not continue to grow.

2.2 Science Matrices: Science Objectives vs. Orbital Segment

Once the science objective matrices have been developed, the times in the mission when the science objectives are acquired needs to be established. For a "swingby" mission, like Voyager, the encounter period may be divided into segments and geometric events (e.g., approach, far encounter, near encounter, planet closest approach (C/A), satellite C/A, post encounter). For an orbiter mission which studies temporal variations of a target for many years, orbital segments are created by the identification of geometric events. As an example, the Cassini mission starts with Saturn Orbit Insertion (SOI) and then has its associated geometric events:

1. atmospheric (e.g., atmosphere occultations, phase angle, etc.)
2. magnetospheric (e.g., bow shock crossings, satellite wake crossings, etc.)
3. ring (e.g., ring plane crossing, ring occultations, etc.)
4. satellite events (e.g., Titan encounters, targeted icy satellite encounters, nontargeted icy satellite encounters)

SCIENCE MATRICES

CASSINI TITAN SCIENCE OBJECTIVES

	CAPS	ISS	MAG	RPWS	UVIS	VIMS
ABUNDANCE	●				●	●
CHEMISTRY	●	●			●	●
CIRCULATION		●		●	●	●
MAGNETOSPHERE	●		●	●	●	

Fig. 1: This marix shows which investigations capture each science objective.

CASSINI
SCIENCE OBJECTIVES vs. ORBITAL SEGMENT

Sci Obj \ Orb Seg	Probe	F&P	Occult	Sat
TITAN				
ABUNDANCE	1	N	N	N
CHEMISTRY	1	N	N	N
CIRCULATION	1	N	N	N
MAGNETOSPHERE	2	N	N	N

1 - Major Observation Period
2 - Minor Observation Period
N - Not Applicable

Fig. 2: This marix identifies the importance of each epoch in the orbit based on science objectives.

CASSINI
SCIENCE OBJECTIVES vs. OBSERVATION TYPE

Science Objective	Prime	Obs Type	Comment
TITAN			
Atmospheric	CAPS	Roll	D/L FP&W
Abundances	UVIS	Mosaic	Auroral Scan
	VIMS	Mosaic	4 x 4
Chemistry	CAPS	Roll	D/L FP&W
	ISS	Mosaic	3 Filter

Fig. 3: This marix defines activities that the spacecraft must perform to obtain the desired science.

CASSINI
OPS CHARACTERISTICS vs. OBSERVATION TYPE

Ops Charact \ Obs Type	Limbtrack	Mosaic	Roll
Adaptability	Low	Low	Lo
Dev. Time/Execute Time	3	1	1
Concurrent Activities	2	1	1
Repetitiveness of Sequence	Unique	Blocks	Bl
Simultation Effort	Most	None	N

Fig.4: This marix allows the science community to independently evaluate Ground System resources.

Once segments are defined from the geometric events, a matrix of science objectives vs. orbital segments is developed (see fig. 2 "Cassini Science Objectives vs. Orbital Segments"). It is important to note that the sum of the segments defines the entire encounter or orbital tour. If it does not, then the addition of "place holders" may be necessary. "Saturn Orbital Ops" is an example of a Cassini orbital tour place holder. This place holder is needed because some high priority observations are bound to orbital characteristics and not just particular geometric events. These high priority events dictate that "Saturn Orbital Ops" be divided into high activity and low activity segments. Only high activity periods contain high priority events. The low activity segments are for the remainder of the orbital tour

An example of an observation which requires a high activity period is a stellar ring occultation. This important observation is tied to both a geometric event and orbits with relatively high inclinations. For Cassini, these orbits occur early and late in the orbital tour. A low activity period may contain periodic fields, particles & wave measurements. These measurements are critical to the understanding of the magnetosphere but may be done anywhere in the orbit. The spacing of individual observations do not matter as long as complete coverage of the orbit is obtained.

2.3 Science Matrices: Validation of Orbital Segments

The "Science Objectives vs. Orbital Segment" matrix is used to determine the times in the mission when the science objectives are achieved. A "1", "2" or "N" is placed in each cell of the matrix to identify the degree in which the objective was captured during the particular orbital segment. A "1" indicates that the objective was met during the particular orbital segment. A "2" indicates that some portion of the objective was met; and an "N" indicates that the objective could not be obtained at this particular time.

Once the entire matrix is finished, all cells with an "N" are shaded for readability. This matrix can now be used to validate that the set of orbital segments is complete. The validation process is first performed on the rows (i.e., science objectives). Each row must have at least one "1" or a "2" in it. If it does not, then the objective is not captured with the current set of orbital segments. This implies that either the objective should be removed or a new orbital segment (which would capture the objective) be added.

Next, the columns are checked for internal consistency. At least one "1" or "2" should be in every column. If it does not, then the column (i.e., orbital segment) is unnecessary and should be removed from the matrix (In this case, some columns do not contain a "1" or a "2" because this figure is only a part of the complete matrix). It is desired for simplicity that the final matrix have the least number of columns. The end result is a table that explicitly defines when in the mission specific science objectives are obtained.

2.4 Science Matrices: Define Observation Types

Science investigators next define observation types. An observation type is an activity needed by an investigation in order to capture a scientific objective. The investigator only needs to define those types of activities that impact ground system resources. Any activity that is performed internal to the instrument does not need to be considered as it only drive the investigation's resources.

The observation types are used to ensure that the GS has the correct resources in place as determined by the investigators. An example of an observation type is a

"mosaic". The shuttering of a single image, a UV atmospheric occultation observation and a mass spectrometer sample of the atmosphere (by orienting the spacecraft into the ram direction) all fall under the same observation type (i.e., 1 x 1 Mosaic). In each case, the investigation needs to orient its field of view in only one specific direction.

Observation types are determined by creating a table of science objectives, investigation that provide "notable contributions" (a.k.a prime investigations) and then defining the proposed observation type (see fig. 3 "Cassini Science Objectives vs. Observation Type"). The first Titan science objective, "Atmospheric Abundances", lists the investigations that were identified as prime in the "Science Objectives vs. Investigation" matrix (see fig.1). For each investigation in a particular science objective, an observation type is identified.

While identifying observation types, it is important to remember that the number of types be kept to a minimum. This is driven by the fact that the larger the number of types, the more resources have to be spent by the GS to capture them. Thus, if Titan spiral radiometry scans and Saturn limbtrack maneuvers can both be performed by the same spacecraft routine (i.e., "maneuver" observation type), than a cost savings will be realized.

Once all the objectives have been assigned an observation type, a summary of the different types is compiled. In this case, Cassini has six basic observation types:

1. Articulation - Mechanical Motion of Cassini Plasma Spectrometer, Cosmic Dust Analyzer & Magnetic Imaging Instrument
2. Langmuir Probe Operations - Radio & Plasma Wave Science Experiment
3. Maneuver - RADAR Radiometry & Radio Science Limb-tracks
4. Mosaics (m x n) -
    a. 1 x 1 (e.g., Imaging, Integration or Stare)
    b. 1 x m (i.e., Scan)
    c. n x m (i.e., Mosaic)
5. Roll - Spacecraft Roll at 0.26 deg/s for Fields, Particles & Waves
6. Sounder Mode Operations - Radio & Plasma Wave Science Experiment

This list contains all activities that the GS has complete or partial responsibility for in order for the investigations to achieve their science objectives. In addition, this list begins to define the fundamental activities that could be built into the ground system prior to the orbital tour. With good system engineering, these activities should only require changes to their parameters in order to be used during the mission.

2.5 Ground System Matrices: Operations Characteristics vs. Dynamic Range

The GS, in turn must define which characteristics during operations drive its resources. For each characteristic a range of values are defined to establish its dynamic range. As an example, the repetitiveness of a sequence directly drives the amount of resources (i.e., dollars) that must be utilized to develop command loads. The range extends from none, where each sequence is used only once (i.e., unique); to high, where each sequence is used many times. Obviously the more frequently a sequence can be used, the greater the cost savings during operations.

For the Cassini mission, operational characteristics fall into five areas; sequencing, spacecraft, navigation, systems and real-time operations. In each area, characteristics which drive operation costs and their associated dynamic ranges are identified. It is important to note that each mission has its own unique cost drivers. As such,

operational characteristic tables must be generated for each mission.

2.6 Ground System Matrices: Operations Characteristics vs. Orbital Segment

Once the GS establishes its operations characteristics, an "Operations Characteristics vs. Orbital Segment" matrix is produced. This matrix allows the GS to scope where in the mission specific resources are necessary based on the relative importance of each orbital segment. The level of resources placed in each cell are done based on the mission plan and in accordance with the available GS resources. The final matrix represents the GS's best estimate of when specific capabilities must be in place in order to achieve the objectives of the mission.

It must be mentioned that in actuality resources can not be added and subtracted as frequently as indicated by the change of orbital segments. Personnel must be trained in advance of their need date and must remain at their task for at least a number of months. An employee can not be hired for a task for five days only to be removed for the next three weeks. However, the allocation of ground resources does identify the ebb and flow of resources and thus help determine the level of effort that must be applied at different times in the mission.

2.7 Science Matrices: Operation Characteristics vs. Observation Type

With the generation of the GS's operation characteristics, the science representatives (i.e., Project Scientist, Principal Investigators, Experiment Representatives, Investigation Scientists, Science Coordinators, etc.,) produce the ops characteristics vs. observation type matrix (see fig. 4 "Cassini Operation Characteristics vs. Observation Type"). This matrix, endorsed by the science community (independent from the ground system), establishes what resources are needed by the investigations in order to capture a specific type of activity. It is this matrix that will be used against the GS's estimate of the availability and allocation of its resources.

3.0 Application

As an example of the application of these matrices, Cassini RADAR scans will be analyzed. First find which objectives require RADAR scans. To do this, look at fig. 5, "Cassini Science Objectives vs. Observation Type". Determine the objective(s) for which RADAR is the prime investigation and the observation type is "scans". For this particular case, RADAR scans are only needed at Titan to determine the "State/Composition of Surface".

With the science objective known, use the "Cassini Science Objectives vs. Orbital Segments" (see fig. 6) to determine when the particular objective may be acquired. The table indicates (by the presence of "1s" or "2s") that scans are only needed during the "Probe" and "Titan" orbital segments. When we apply the fact that RADAR will not be used during the probe mission, then we realize that the GS only has to provide the capability for RADAR scans during Titan swingbys

Next return to the "Cassini Ops Characteristics vs. Observation Type" matrix (see fig. 7). From this matrix remove the RADAR scan column and compare to the he "Titan" column from the "Cassini Ops Characteristics vs. Orbital Segment " matrix (see fig. 8)". For ease of review, the orbital segments not needed for RADAR scans have been shaded gray.

The requirements of the RADAR scan is then compared with the capability provided by the GS. For this example, areas in the RADAR column which require more capability then the ground has provided were shaded gray. In this

RADAR SCAN EXAMPLE

CASSINI
SCIENCE OBJECTIVES vs. OBSERVATION TYPE

Science Objective	Prime	Obs Type	Comment
TITAN			
Atmos. Circulation & Physics	RSS UVIS VIMS	Limbtrack Movie Mosaic	2-Frequencies Feature Track Feature Track
State/Comp. of Surface; Interior	RADAR RSS	Scan Limbtrack	Radiometry X- and Ka-B
Upper Atmos. Relation	CAPS INMS	Articulation Integration	Ram Direc

Fig. 5: First find which science objectives require RADAR scans. In this case, only "State/Comp. of Surface" of Titan.

CASSINI
SCIENCE OBJECTIVES vs. ORBITAL SEGMENT

Sci Obj \ Orb Seg	Probe	F&P	Occult	Ti
TITAN				
ABUNDANCE	1	N	N	N
CHEMISTRY	1	N	N	N
CIRCULATION	1	N	N	N
STATE/COMP SURF	1	N	N	1
MAGNETOSPHERE	2	N	N	N
SATURN				

1 - Major Observation Period
2 - Minor Observation Period
N - Not Applicable

Fig. 6: Titan surface composition measured during Probe and Titan segments. However, during the probe mission, the main antenna will be used for data relay not RADAR. Thus, RADAR scans only needed during Titan passes.

CASSINI
OPS CHARACTERISTICS vs. OBSERVATION TYPE

Ops Characteristics \ Observation Type	Articu-lation	Mosaics	RADAR scans	Sounder
Adaptability	Low	Low	Low	Low
Dev. Time/Execute Time	2	2	3	2
Concurrency	2	2	1	2
Repetitiveness of Sequence	Blocks	Blocks	Unique	Blocks
Simultation Effort	None	None	All	None

Fig. 7: Investigators, independent from the GS, generate the ground capability needed for each observation type.

CASSINI
OPS CHARACTERISTICS vs. ORBITAL SEGMENT

Ops Charact \ Orb Seg.	Probe	Occult	Titan
Adaptability	Low	Low	Low
Dev. Time/Execute Time	3	2	2
Concurrency	2	1	1
Repetitiveness of Sequence	Blocks	Blocks	Blocks
Simultation Effort	Most	None	None

RADAR scans
Low
3
1
Unique
All

Fig. 8: Compares GS capability with the science requirements needed to capture science objectives. Identifies which activities need to be simplified, which GS capabilities needs to be reallocated, or which activities may be at risk.

example three areas (i.e., development time/execute time, repetitiveness of sequence and simulation effort) are in conflict. If we look at the "Simulation Effort" row on this table, we see that the GS does not plan to simulate RADAR sequences. However, from a science point of view, all RADAR sequences must be simulated. This apparent discrepancy results in one of the following:

1. GS reallocates resources to simulate all RADAR scans, or
2. The RADAR Team uses its own resources to simulate scans prior to submitting their sequences to the GS, or
3. Nothing is changed and the projects excepts the greater risk of science data loss during RADAR scans

4.0 Conclusion

The use of these matrices by the science community and the project's ground system allows both groups to understand what and when types of observations can be performed. The results make the science community sensitive to the limits of the ground resources and thus, reduce the amount of "creeping" science requirements. In turn, the GS will be more responsive to the needs of the investigators in order to return the primary science objectives of the mission.

Once the matrices have been developed and analyzed, potential misallocation of resources will become evident. The areas where investigator's requirements are greater than the available resources will drive the GS and science community to one of three possibilities:

1. Reallocate GS capability to meet the observation, or
2. Decrease the observation type's complexity by transferring the responsibility to the investigator, or
3. Leave resources as is and accept the greater risk of data loss

The approach stated in this paper may be applied during advanced mission planning in order to select a spacecraft's science payload; during ground system design to ensure the ground system's compatibility with the investigations; and during operations to quantify where ground resources need to be applied to return the quality of science data demanded by a first rate planetary exploration program.

5.0 References

1. Cassini Project Policies & Requirements Document; JPL Internal Document; PD 699-004 Rev. B; 1992 September

2. Cassini Ground System Architecture Review; JPL Internal Document; Volume III; 1993 April 8; "Framework for the New Ground System Design"; R. B. Morris; pages 526-527

3. "OCMP Table and OCBYMP Matrices", IOM 380-92-0-004/JD, J. H. Duxbury, 1993 June 3

4. Cassini Tour Cost Sensitivity Working Group Final Report; JPL Internal Document; 1993 September 24

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Systems Engineering

6. Systems Operations

* *Presented in Poster Session*

111196

N95-17611

354330

p. 6

A New Systems Engineering Approach to Streamlined Science and Mission Operations for the Far Ultraviolet Spectroscopic Explorer (FUSE)

Madeline J. Butler
Systems Engineering Office
Mission Operations and Data Systems Directorate

Dr. George Sonneborn
Laboratory for Astronomy and Solar Physics
Space Sciences Directorate

Dorothy C. Perkins
Missions Operations Division
Mission Operations and Data Systems Directorate

NASA/Goddard Space Flight Center

ABSTRACT

The Mission Operations and Data Systems Directorate (MO&DSD, Code 500), the Space Sciences Directorate (Code 600), and the Flight Projects Directorate (Code 400) have developed a new approach to combine the science and mission operations for the FUSE mission. FUSE, the last of the Delta-class Explorer missions, will obtain high resolution far ultraviolet spectra (910 - 1220Å) of stellar and extragalactic sources to study the evolution of galaxies and conditions in the early universe. FUSE will be launched in 2000 into a 24-hour highly eccentric orbit. Science operations will be conducted in real time for 16-18 hours per day, in a manner similar to the operations performed today for the International Ultraviolet Explorer.

In a radical departure from previous missions, the operations concept combines spacecraft and science operations and data processing functions in a single facility to be housed in the Laboratory for Astronomy and Solar Physics (Code 680). A small mission operations team will provide the spacecraft control, telescope operations and data handling functions in a facility designated as the Science and Mission Operations Center (SMOC). This approach will utilize the Transportable Payload Operations Control Center (TPOCC) architecture for both spacecraft and instrument commanding. Other concepts of integrated operations being developed by the Code 500 Renaissance Project will also be employed for the FUSE SMOC. The primary objective of this approach is to reduce development and mission operations costs.

The operations concept, integration of mission and science operations, and extensive use of existing hardware and software tools will decrease both development and operations costs extensively. This paper describes the FUSE operations concept, discusses the systems engineering approach used for its development, and the software, hardware and management tools that will make its implementation feasible.

PRECEDING PAGE BLANK NOT FILMED

1370 INTENTIONALLY BLANK

MISSION DESCRIPTION

The FUSE science program will address fundamental problems in such diverse areas as composition and properties of interstellar gas and dust, stellar explosions and mass loss, galactic dynamics, active galactic nuclei, and planetary magnetospheres. Many of these problems require long exposures (15-50 hours) of faint objects. Among the most important and demanding FUSE science is the study of trace species in interstellar and intergalactic gas using absorption spectroscopy of faint, distant sources, such as active galactic nuclei and quasars. The deuterium-to-hydrogen (D/H) abundance ratio is a critical parameter for understanding the Big Bang and the chemical evolution of the universe. FUSE will address this problem by measuring the D/H ratio in a wide range of astrophysical conditions representing different evolutionary histories, degree of stellar development, and chemical mixing. These problems require high spectral resolution and instrument sensitivity in the wavelength range 910 to 1220 Angstroms. The mission design lifetime is three years to provide sufficient observation time to meet the science exposure time requirements; the mission goal is five years.

The FUSE Principal Investigator at the Johns Hopkins University(JHU), Baltimore, Maryland, leads a university-government team which will build and test the FUSE instrument. The instrument team includes JHU, University of Colorado at Boulder, University of California at Berkeley, and the GSFC Engineering Directorate (Code 700). Canada and France are partners in the FUSE mission and are providing critical elements of the instrument to JHU. FUSE operations are the responsibility of the GSFC Laboratory for Astronomy and Solar Physics.

Since the start of the Phase B there have been major changes in the FUSE mission concept in order to meet programmatic requirements while maintaining scientific performance. The result is a new, innovative normal-incidence optical design, a dedicated spacecraft to be built at GSFC, a Delta II launch, and a 24-hour highly-eccentric, geosynchronous orbit (600 km perigee, 71000 km apogee) which provides about 18 continuous hours of unocculted, low radiation background science time per day (see Figure 1). This occurs when the spacecraft is at altitudes greater than about 30,000 to 40,000 kilometers.

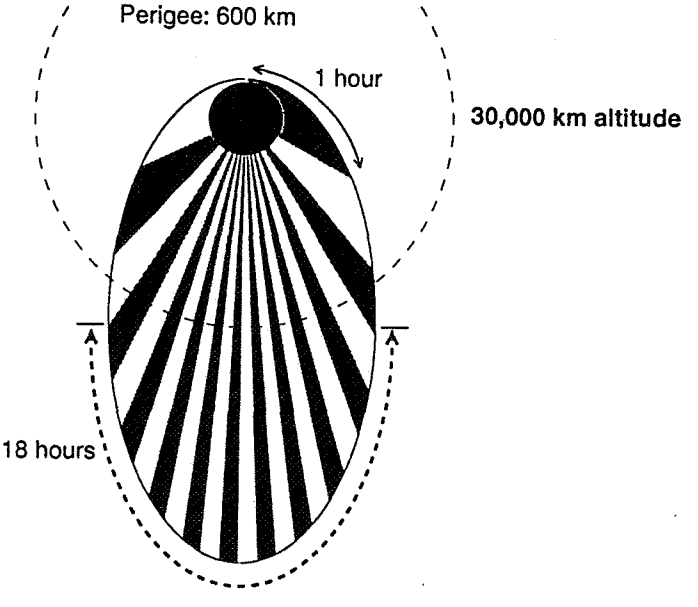


Figure 1

FUSE Orbit Geometry

FUSE OPERATIONS CONCEPT

Through a series of trade studies involving the instrument, spacecraft, and ground system, a new operations concept was developed. In addition to lowering total mission costs, low-cost mission operations was a major objective of this process.

The principal characteristics of the FUSE operations concept are summarized as follows:

- Science operations take place 18 hours per day, and only when the spacecraft is at altitudes above 30,000 km.
- Spacecraft telemetry rate is 64 kbps, using one of two omni-directional antennas.
- Two ground stations can provide coverage of 19-20 hours per day; only a 10m dish is required to provide adequate link margin (3db). This coverage requires one station in the Northern Hemisphere and one in the Southern. Wallops Island, VA, and Canberra, Australia, were used for analysis purposes. This wide geographical separation is required due to the high eccentricity of the orbit.
- Spacecraft and instrument command and control is generated and executed in real time.
- All ground system functions are located in the Science and Mission Operations Center (see Figure 2).
  - Data capture and level zero processing
  - Spacecraft and instrument command and control
  - Science/mission planning and scheduling
  - Orbit determination (flight dynamics function)
  - Trend and performance analysis (with support of instrument and spacecraft teams)
  - Science data processing, calibration, distribution, archiving
  - Science program management
  - Guest Observer proposal support
- A number of functions and processes are automated in order to reduce the work complement of the ground operators.
  - Spacecraft and instrument health and safety functions in flight software
  - Health and safety monitoring in ground system
  - Data capture and data retransmission
  - Level Zero processing
- Constraints imposed to simplify operations
  - No operations will take place during the lower portion of the orbit when FUSE traverses the trapped electron and proton belts.
  - No spacecraft maneuvers outside periods of direct contact.
  - No science operations required during shadow periods.

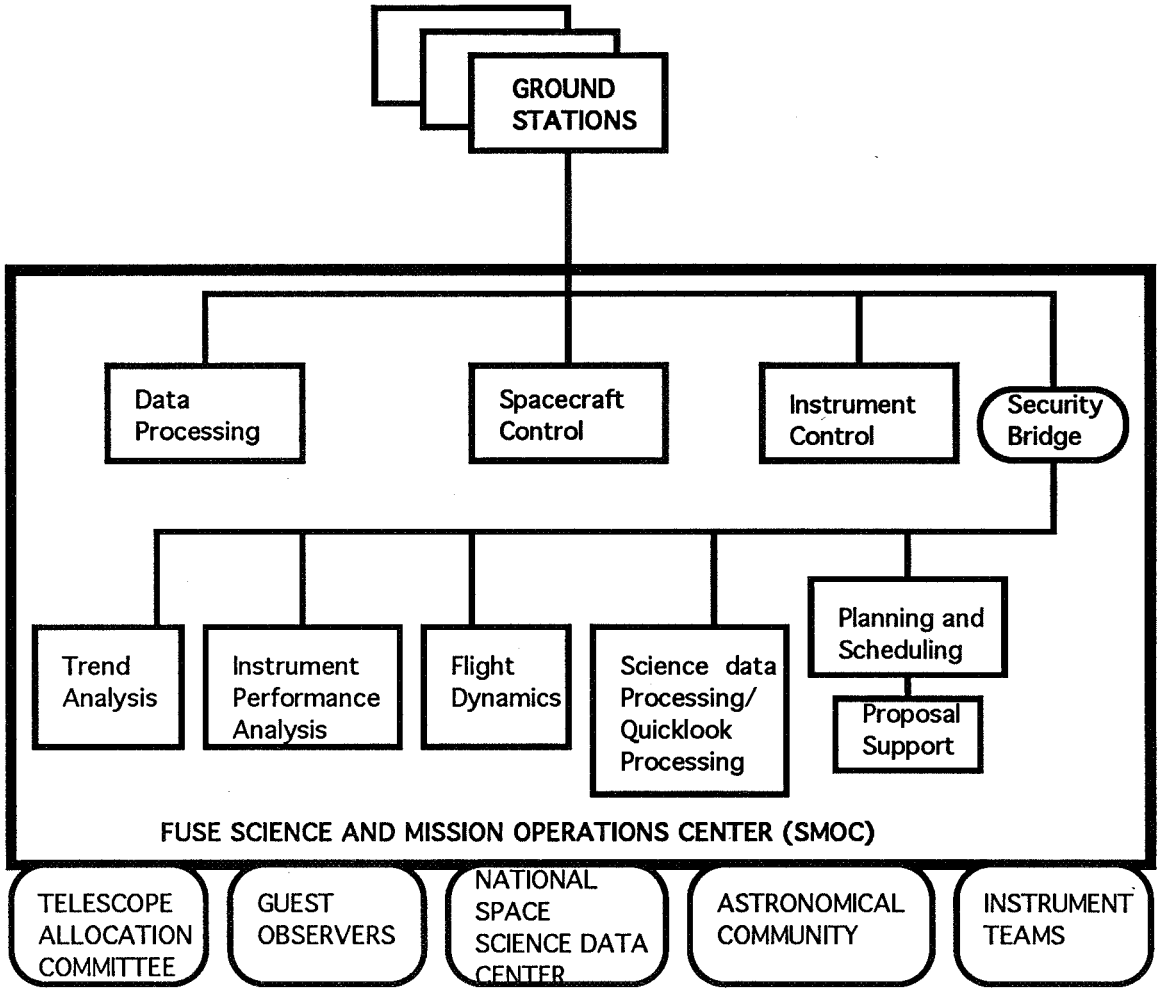


Figure 2
FUSE Science and Mission Operations Center

Real time requirements include the ability to execute interactive, real-time control of the instrument and spacecraft. The daily real-time contact must occur when the spacecraft is at altitudes above 30,000 km to use the low radiation portion of the orbit. Health and safety, housekeeping and science data will be monitored autonomously in real-time to avoid possible data loss due to anomalies or improper operations. The target and guide star identification and acquisition process is interactive with the ground. Adjustments to the instrument fine alignment are required every two hours or less; commanding the spacecraft and instrument will occur on a frequent basis.

FUSE Guest Observers and the Instrument Team will have electronic connections to the SMOC for proposal submission, observation planning and execution, and science data evaluation and retrieval. Level zero data will be available to scientists shortly after receipt in the SMOC and processed science data within 2 days. The mission data will be archived at the National Space Science Data Center (NSSDC).

FUSE DEFINITION PROCESS

Ground and flight segment engineers and scientists have been heavily involved throughout the FUSE definition process. This breadth of involvement has allowed mission-wide and life-cycle trade studies to be conducted and informed design decisions made based on study results. A team consisting of the Project Scientist, the Spacecraft Manager, Instrument System and Operations Engineers, Ground Data Systems Engineers, Operations Scientist, Mission Operations Manager, and Flight Dynamics Systems Engineer was established to create a flight and ground system concept that meets the mission requirements. The revised concept was reviewed periodically by the Principal Investigator and the FUSE Project. The ground system concept described above was ultimately one result of this process. Since operational considerations were an integral part of the mission and life cycle trade studies, the operations and ground system concept developed continuously with the rest of the mission concept.

The orbit selection was the first major trade study. The 24-hour orbit described above provides several significant advantages over a low-earth orbit for the FUSE mission. This orbit decision minimizes the mission lifetime for the mission's science program, lowers mission complexity (operations, spacecraft, and instrument), maximizes time outside the radiation belts for low background science by providing 2 to 3 times the observing efficiency of LEO, provides a 100X improvement in target visibility, radically simplifies science and mission scheduling, provides continuous observing periods for long observations, and simplifies faint-object target acquisition.

These factors, combined with the utilization of real-time control of the instrument and spacecraft, reduce the number of functions and subsystems in the flight and ground segments. Such simplifications lower system complexity, lower development and testing costs, lower software maintenance costs, and reduce operations support staff. Specifically, a high-rate data downlink, command management system, planning and scheduling system, and automated target and guide star acquisition system were either deleted from the FUSE mission concept or significantly simplified. Many other design trades were made to optimize the mission design concept and reduce cost.

The fact that FUSE is operated in real time, has one instrument, one common destination for all data, and a low rate for data downlink made possible the consolidation of the spacecraft and science operations. A single Mission Operations Team will be trained to perform both science and spacecraft operations. The current concept plans to staff each operational shift with two console operators (one for the spacecraft, one for the instrument) and one resident astronomer. The operations staff will be assisted by autonomous functions in the ground system for health and safety monitoring, data capture and retransmission, and Level Zero processing. These are very significant, because by automating these activities the operations staff will have significant portions of their time freed for more intellectually challenging and critical work, such as calibration, planning and scheduling, preparing for the next science observation, and of course spacecraft and instrument commanding.

The SMOC will be a joint development by Codes 600 and 500 and the FUSE instrument team. The objective is to reduce documentation (e.g., a single requirements document for the FUSE ground system), reduce and simplify interfaces, and reduce staffing and facility requirements. The SMOC will be one of the missions to be implemented utilizing services and products of the Code 500 Renaissance system.

FUSE DEFINITION METHODOLOGY

The FUSE Project has adopted a Functional Analysis methodology to achieve an end-to-end systems approach to mission concept development. FUSE is the first project at GSFC to employ this process across the entire mission starting in the definition phase. Functional analysis is the process of identifying, describing and relating the functions a system must perform in order to fulfill its goals and objectives. The primary analysis tool is the functional flow block diagram. These diagrams show the network of actions that lead to the completion of a function. It is part of the end to end flow of level one activities that leads to the system definition. The process starts with requirements, followed by functional analysis, and ends with system definition.

The mission phases that were defined to be studied were development, prelaunch, launch, mission operations, and end of mission. Each mission phase was further broken into sub-phases, and sub-phases into functions. This allows the systems engineers to evaluate the functional relationships and dependencies across subsystems and between flight and ground segments. In addition, the interdependencies at the function level can be traced to higher levels. The functional flow diagrams will be used to allocate and validate requirements at the subsystem level.

This process helped to identify major areas of the mission for further study, such as the critical topics for the FUSE End-to-End Data System Study. The main objectives of this study are: to characterize the space to ground link, the command and data handling memory sizing and the on board bus traffic; and to establish a baseline of the data rates and identify "tall poles" in the instrument, observatory and ground data systems. Among the important outcomes of this type of study are the identification of missing requirements, the identification of the requirements that drive the complexity of the system in terms of data and the validation of the operations concept. In parallel, the FUSE Operations Concept Document was written. Through this process the mission operations concept is "engineered" like the other components of the mission.

CONCLUSION

Because of its need to dramatically reduce costs, and its success in achieving this goal, the FUSE mission is a pathfinder for NASA's low-cost mission operations objectives. A tightly integrated team and consideration of end-to-end operations issues in all phases of the FUSE mission contributed substantially to the development of this approach.

N95-17612

111197

354534

p. 5

Risk Reduction Methodologies and Technologies for the Earth Observing System (EOS) Operations Center (EOC)

Richard K. Hudson and Nelson V. Pingitore
Loral AeroSys
1616A McCormick Drive
Landover, Maryland 20785

ABSTRACT

This paper will discuss proposed Flight Operations methodologies and technologies for the Earth Observing System (EOS) Operations Center (EOC), to reduce risks associated with the operation of complex multi-instrument spacecraft in a multi-spacecraft environment. The EOC goals are to obtain 100% science data capture and maintain 100% spacecraft health, for each EOS spacecraft. Operations risks to the spacecraft and data loss due to operator command error, mission degradation due to mis-identification of an anomalous trend in component performance or mis-management of resources, and total mission loss due to improper subsystem configuration or mis-identification of an anomalous condition. This paper discusses automation of routine Flight Operations Team (FOT) responsibilities, Expert systems for real-time non-nominal condition decision support, and Telemetry analysis systems for in-depth playback data analysis and trending.

INTRODUCTION

The Flight Operations Segment (FOS) of the EOS Core System (ECS) is currently in early stages of the design process. The Preliminary Design Review (PDR) is scheduled for December 1994, and the concepts discussed in this paper will be refined as we progress with the development cycle.

The FOS will provide the command and control system for EOS instruments and spacecraft. The EOC will be located at NASA Goddard Space Flight Center (GSFC) to generate commands to the instruments and spacecraft of NASA within the International Earth Observing System (IEOS) as well as monitoring the health and performance of these flight elements.

International partners flying instruments on NASA EOS spacecraft will be able to provide these functions for their instrument from their own center. Principal Investigators (PIs) and facility instrument teams will participate in monitoring their instruments and in resolving instrument anomalies from their home institutions through use of an Instrument Support Terminal (IST) Toolkit, a special set of software that will be run on a local computer workstation.

AM-1 will be the first mission to be supported by the EOC FOT, it is scheduled for a June 1998 launch. Other missions currently scheduled to be supported from the EOC include: AERO-1 (9/00); PM-1 (12/00); ALT-1 (6/02); CHEM-1 (12/02), AM-2 (6/03); AERO-2 (9/03); PM-1 (12/05); AERO-3 (9/06); ALT-2 (6/07); CHEM-2 (12/07); AM-3 (6/08); AERO-4 (9/09); PM-3 (12/10); ALT-3 (6/12); AERO-5 (9/12); and

CHEM-3 (12/12). At full capacity the EOC FOT will be required to support up to 7 missions (five on-orbit, one in pre-launch stage, and one mission in a "decommissioning" stage).

Systems will be in place well before the launch of the first EOS satellite to provide the full functionality required to support it. The command and control functions will be brought on-line and fully tested with simulated EOS data in operational scenarios.

After the EOS missions are on-orbit and providing high volumes of data, the EOC will continue to evolve and add capabilities in response to new requirements and lessons learned through its use. This evolution will be in the form of planned EOC upgrades. Continued prototyping will occur, and development of EOC will be actively sought. This continuing evolution will enable the EOC to incorporate advances in data system technologies, as well as adapt to changing user requirements.

The FOT shall provide mission operations support with technical directives from the NASA Mission Operations Manager (MOM) and the EOS Project Scientist. Coordinated mission planning, scheduling, and commanding operations shall be performed by the FOT in accordance with the MOM's policy guidelines and directives. Instrument science planning and scheduling operations, including conflict resolution, shall be performed under the general high level direction and guidance of the Project Scientist. The FOT shall perform operations necessary at the EOC to ensure that the ECS FOS achieves the functional and performance requirement of the ECS specification. These functions include the following services: operation planning and scheduling; command management; commanding; telemetry processing; observatory and instrument monitoring and analysis; data management; element management; and user interface services.

With increased complexity of space and ground systems, and interactive science operations there will be an increase in the level of complexity of onboard resources and constraint management. Current tools for assessing the state of the system require that FOT members mentally convert alphanumeric data into a mental model and reason about the model. Automated logic checking is performed at the parameter level, leaving subsytem and system level assessment as a human task. Because of this FOT effectiveness is an issue that promises to grow in the future.

The ECS FOS will supply tools that reduce FOT sensory requirements. This will be accomplished through the development of automated routine FOT responsibilities. A Decision Support System, and the EOC Telemetry Analysis system will be the main tools used to automate these responsibilities. The goal of these systems is to aid in complex parameter checking, and system level reasoning checks for the FOT. These tools will also support real-time resource and constraint management. In the EOC these tools must effectively manage multiple payload sets in a dynamic resource allocation environment.

Automation of expected versus actual state analysis process will greatly aid the FOT. FOT productivity gains will also be achieved by visualization tools for assessing system status. These

visualization tools will support graphical representations of system level data with the capability of rapidly expanding the displayed information down to the parameter level. The promise of improved visualization techniques is that the FOT will be able to monitor systems by exception, rather than through surveillance. Each of these methods has the promise of improving FOT efficiencies and reducing mission critical risks.

AUTOMATION OF ROUTINE FOT RESPONSIBILITIES

In traditional control centers significant labor is expended for routine operations, such as monitoring subsystem displays, and supporting poorly engineered interfaces for the negotiation of external services (e.g., communications, flight dynamics). An ECS FOS goal is to automate and standardize these interfaces. This will result in increased operational efficiency, lower system life-cycle costs, and reduced operational risk.

The FOT will depend heavily on the accuracy and quality of spacecraft manufacturer and FOS documentation and information. Deficiencies in either spacecraft or FOS documentation or ground system test results will increase the level of mission risk, reduced mission effectiveness, and result in higher life-cycle costs. These deficiencies will lessen the FOTs ability to provide accurate responses to anomalies.

The amount of information required for operating a spacecraft is staggering. In a traditional control center this data is stored on paper in an ad hoc manner. This information is usually not organized in a way that allows quick access by the FOT for real time operations. Frequently this data is not kept on paper at all, but is retained in the minds of experienced FOT members.

We have proposed an extensive on-line technical information database for the missions controlled from the EOC. This system would allow for rapid information access through keywords, such as: subsystem name, and telemetry or command mnemonics. The information stored within this database will be integrated to the system level. The database will also serve as a repository of system specification, drawings, simulation and test data, historical data, operations procedures, and contingency plans.

Traditional control center operations involve a large number of alphanumeric displays monitored routinely by subsystem specialists with relatively little automatic checking of data, except for simple limit checks. The EOC operations concepts for FOT real-time monitoring calls for most monitoring to be performed by exception when specified rules are violated, when telemetry does not match predictive models, or when telemetry behavior is similar to previously known anomalous patterns.

The EOC telemetry displays will be at a system level, integrating pictures and hierarchical diagrams of spacecraft subsystems with detailed presentation of subsystem data. Both "idiot lights" and dense information displays will be provided and further integrated with analytic tools that provide data exploration capabilities.

With the insertion of more powerful

workstations and operating systems, complex analytic tasks will be performed in real time on a non interfering basis. This will result in lessening the traditional distinction between on-line and off-line systems. As an example, upon the detection of a major system event or anomaly, the FOS software might formulate a list of information relevant to the problem at hand and automatically provide pop-up windows displaying and organizing this information along with appropriate recommendations.

EOC DECISION SUPPORT SYSTEM (DSS)

The EOC FOT will utilize the proposed DSS to provide long-term analysis support and reduce mission critical risk factors.

The EOC DSS should encapsulate knowledge from previous missions. Spacecraft knowledge is often lost from mission to mission. In the life of a single long duration mission key operations knowledge may be lost due to personnel attrition. The EOC DSS is envisioned to disseminate this knowledge across the missions supported by the EOC.

The FOS DSS will have the following characteristics:
a. Access to comprehensive detailed spacecraft and operations knowledge to provide a systems perspective.
b. A library of extensive tools that are readily adaptable to a number of problem-solving activities.
c. A robust pattern-matching capability for matching experience to new problems.

The DSS will provide the following functions:

a. Support in-depth spacecraft and ground system long-term analysis. This will use the DSS integrated knowledge base subsystem in a graphical nature.
b. Assist in identifying and resolving space and ground system problems in a proactive manner.
c. Assist in developing plans for correcting current and future problems through consistent application of domain knowledge.

EOC TELEMETRY ANALYSIS SYSTEM

An often neglected, but important aspect of risk reduction in FOT activities is off-line analysis. In traditional control centers this is principally supported through relatively primitive trending and data analysis tools. These primarily include paper listings of trend data that are analyzed by subsystem engineers to determine potential degradation of components.

Careful and continuous analyses of data can improve the lifetime of a spacecraft and reduce risks associated with catastrophic failures. The ECS FOS software will contain on-line and off-line support capabilities' tools that provide rapid analysis of real-time and post-pass spacecraft behavior, and more direct user management of spacecraft activities.

Heritage and lessons learned from NASA, and NOAA missions will be utilized in developing the EOC Telemetry Analysis System. Specifically these missions include Landsat, EP/EUVE, GOES, and HST.

The EOC Telemetry Analysis System,

includes increased use of workstation-oriented interactive data analysis and visualization tools to support both spacecraft and subsystem analysis. It will scan for anomaly signatures derived from component histories. For example, changes in battery charge/discharge ratios provide early warnings of battery failures. Such signatures would be stored in an operations knowledge base and used to predict component failures.

Proposed specifics of the EOC Telemetry Analysis System include:

a. Support of automatic searches for interesting data and couplings: Such as, problems caused by couplings between attitude and power/thermal subsystems. Unusual iterations would be automatically detected and presented for further analysis in our proposed system.

b. Comparison to recent trends and manufacturer's specifications: Sudden changes in trends or specification would be immediately presented for more detail analyses.

c. Graphical presentation of knowledge in discipline-relevant formats: This includes the presentation of analyses in understandable formats, overlaid on spacecraft diagrams or in multi-dimensional presentation that facilitate data understanding and exploration.

CONCLUSIONS

The EOS FOT will benefit by having the proposed FOS architecture and tools. These benefits include: improved system performance since FOS is more responsive to user needs; reduced risk of spacecraft and/or data loss; lowered operations costs; reduced time and cost of supporting new missions by enhancing the design of the existing ECS FOS software; reduced operator training time and providing for retention of experienced operations personnel knowledge; increased job satisfaction among FOT personnel by automating performance of routine actions; provide for the capability to insert new technologies/products into EOC faster; and provide a more transparent, less obstructive interface between science users and the instrument and science data they use.

BIBLIOGRAPHY

Maurice Assaraf, *Lessons Learned from the Transportable Payload Operations Control Center (TPOCC)*, NASA GSFC SEAS Central Engineering Board presentation, February 17, 1993. Goddard Space Flight Center, Greenbelt, MD.

Allan Jaworski, Gardiner Hall III, and David Zoch *CC2005 An Architecture for Future Mission Operations Control Centers*, Loral AeroSys, Seabrook, MD, October 1992.

Paul Ondrus, and Michael Fatig, *Operations Technology Working Group presentation*, NASA Goddard Space Flight Center, Greenbelt, MD, December 1991.

James R. Wertz and Wiley J. Larson (editors), Space Mission Analysis and Design, Kluwer Academic Publishers, Norwell, MA, 1991.

111198 N95-17613

354336 p. 8

EDOS Operations Concept and Development Approach

G. Knoble, C. Garman, and G. Alcott, Goddard Space Flight Center
and
C. Ramchandani and J. Silvers, Computer Sciences Corporation

Abstract

The Earth Observing System (EOS) Data and Operations System (EDOS) is being developed by the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center (GSFC) for the capture, level zero processing, distribution, and backup archiving of high speed telemetry data received from EOS spacecraft. All data received will conform to the Consultative Committee for Space Data Standards (CCSDS) recommendations. The major EDOS goals are to:

- Minimize EOS program costs to implement and operate EDOS
- Respond effectively to EOS growth requirements
- Maintain compatibility with existing and enhanced versions of NASA institutional systems required to support EOS spacecraft.

In order to meet these goals, the following objectives have been defined for EDOS:

- Standardize EDOS interfaces to maximize utility for future requirements
- Emphasize life-cycle cost (LCC) considerations (rather than procurement costs) in making design decisions and meeting reliability, maintainability, availability (RMA) and upgradability requirements
- Implement data-driven operations to the maximum extent possible to minimize staffing requirements and to maximize system responsiveness
- Provide a system capable of simultaneously supporting multiple spacecraft, each in different phases of their life-cycles
- Provide for technology insertion features to accommodate growth and future LCC reductions during the operations phase
- Provide a system that is sufficiently robust to accommodate incremental performance upgrades while supporting operations.

Operations concept working group meetings were facilitated to help develop the EDOS operations concept. This provided a cohesive concept that met with approval of responsible personnel from the start. This approach not only speeded up the development process by reducing review cycles, it also provided a medium for generating good ideas that were immediately molded into feasible concepts. The operations concept was then used as a basis for the EDOS specification. When it was felt that concept elements did not support detailed requirements, the facilitator process was used to resolve discrepancies or to add new concept elements to support the specification. This method provided an ongoing revisal of the operations concept and prevented large revisions at the end of the requirement analysis phase of system development.

1.0 Introduction

EDOS operations supports end-to-end data delivery for EOS spacecraft. The operations concept describes the strategic, tactical, execution and post-execution phases for EOS Ground System (EGS) elements, and describes the role of EDOS in each phase. In support of these phases, the concept describes EDOS operations in relation to current and future GSFC Mission Operations and Data System Directorate (MO&DSD) institutional systems and EOS systems. These include the Tracking and Data Relay Satellite System (TDRSS) Ground

PRECEDING PAGE BLANK NOT FILMED

1382

Terminals (TGTs), the Network Control Center (NCC), EOS Communications (Ecom), as well as EOS Core System (ECS) facilities, including the EOS Operations Center (EOC), Distributed Active Archive Centers (DAACs), and other EGS elements.

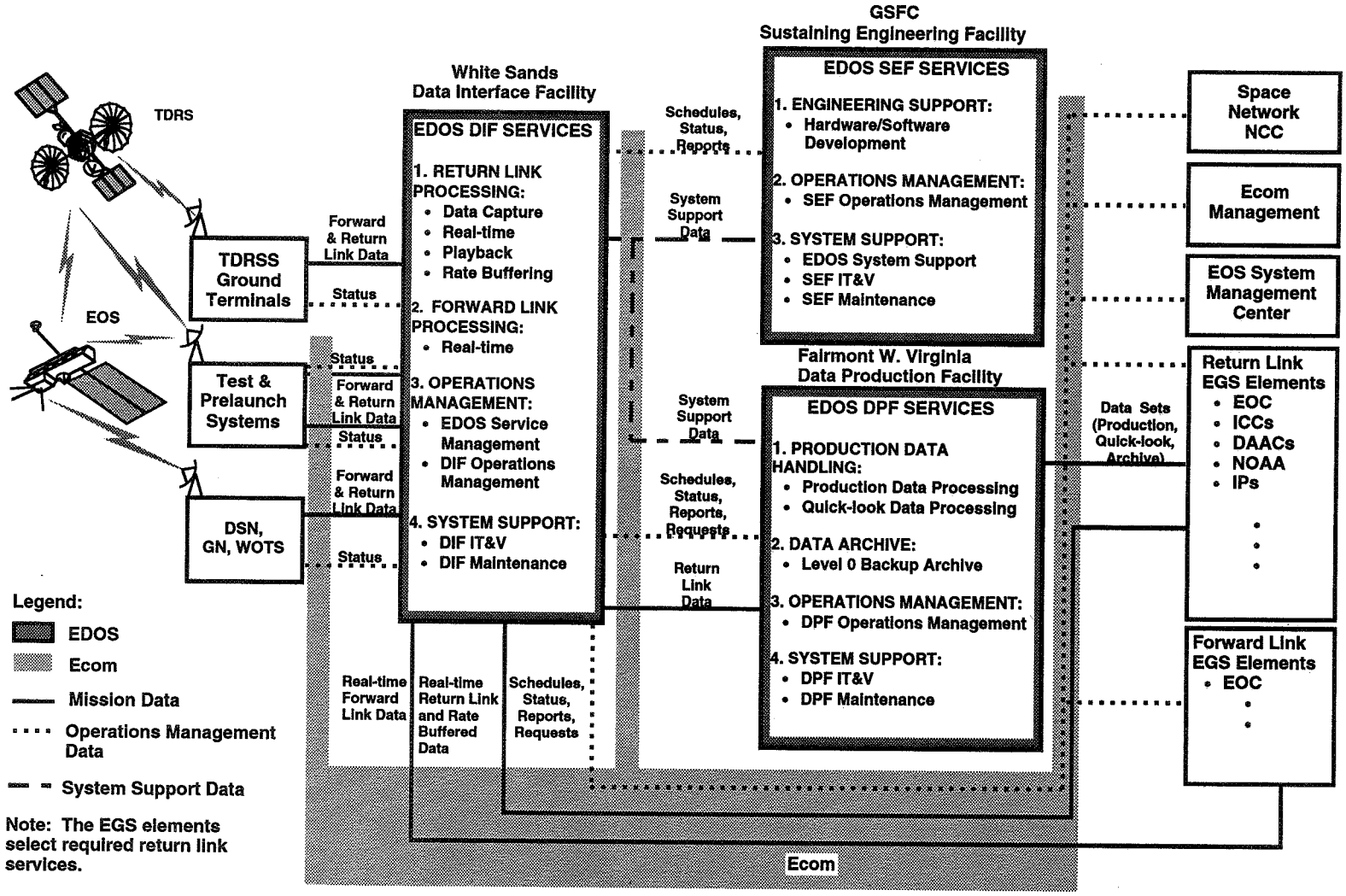
The approach used for developing an operations concept is almost as important as the concept itself. In order to be an effective concept, it must be well thought out and in agreement with the interested parties (systems engineers, interface organizations, and management). The approach must also allow change. This includes a discussion of the development of alternative concepts, and the tradeoff and other engineering analyses performed in selecting and developing the baseline operations concept. The significance of the operations concept in the development of the detailed EDOS functional and performance specification and interface requirements is described as the "proof of concept" of the development method.

2.0 EDOS Operations Concept

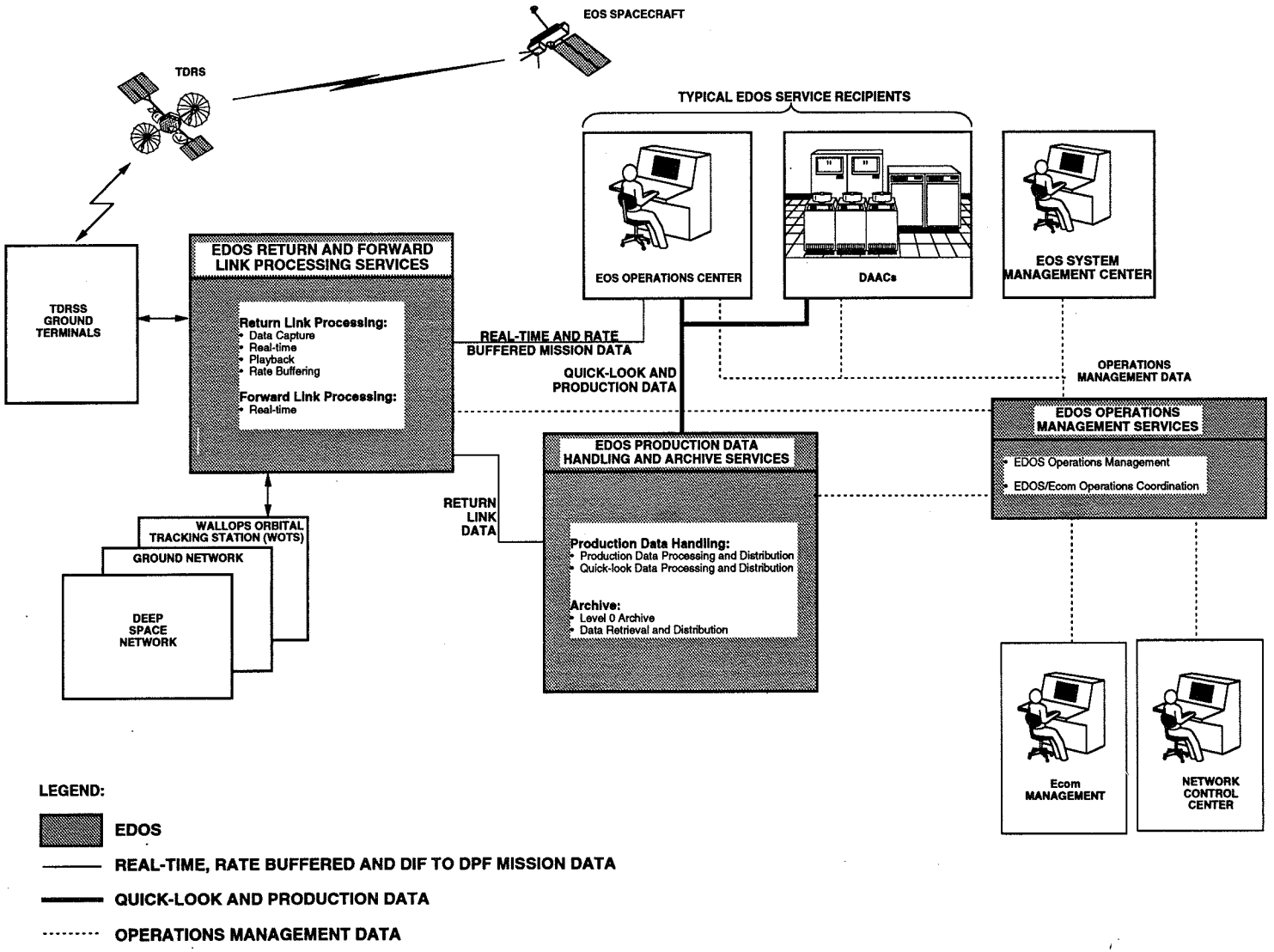
EDOS is the EOS data handling and delivery system maintained and operated by the MO&DSD. The development and implementation is being managed by the Information Processing Division (IPD), Code 560, of the MO&DSD at the GSFC. EDOS provides capabilities for handling data for EOS spacecraft that adhere to recommendations established by the CCSDS. Specifically, EDOS provides capabilities for return link data capture, data handling, data distribution, backup archival data storage, and forward link data handling. EDOS supports ground to ground data communications for data delivery using a set of approved protocols. Reliance of EDOS on these space/ground and ground to ground standards facilitates mission interoperability and will result in lower life-cycle costs for NASA. EDOS supports all levels of MO&DSD and EOS end-to-end testing in preparation for EOS spacecraft launch readiness, by utilizing the operational system without interrupting ongoing operations. Data delivery is provided by the SN, EDOS, and Ecom. SN provides space/ground data communications. The SN consists of the Tracking and Data Relay Satellite (TDRS) constellation, the TGTs, and the NCC. The TGTs include the White Sands Ground Terminal (WSGT) and the Second TDRSS Ground Terminal (STGT). Space/ground data communications for emergency operations are provided by the Ground Network (GN), Wallops Orbital Tracking Station (WOTS), and the Deep Space Network (DSN). Ecom includes the wide area network and the Ecom Management capability, which provide ground to ground data communications support for the SN, EDOS, and EGS elements. EGS elements include the EOS Operations Center (EOC), the Distributed Active Archive Centers (DAACs), or other associated data handling facilities, such as the National Oceanic and Atmospheric Administration (NOAA).

There are three EDOS facilities. The Data Interface Facility (DIF) is located at the White Sands Complex (WSC) near Las Cruces, New Mexico. The Data Production Facility (DPF) is located in Fairmont, West Virginia. The Sustaining Engineering Facility (SEF) is located in the Data Operations Facility (DOF), Building 28 at GSFC in Greenbelt, Maryland.

The capabilities that EDOS provides are grouped into categories of services. These services are allocated to the three EDOS facilities. EDOS services include the data delivery services outlined in the previous section and the services that support EDOS operations. The service categories are designated as return link processing, forward link processing, operations management, production data handling, data archive, system support, and engineering support. The DIF provides return and forward link processing services. The DIF also provides operations management services for DIF processing services and for the centralized EDOS operations management. The DPF provides production data handling, data archive, and DPF operations management services. Return link services are provided according to mission-specific requirements. The SEF provides sustaining engineering services, the EDOS system support coordination services and operations monitoring. System support services are provided at



EDOS Services and Interfaces



Representative EDOS Services

each of the three facilities to support the operations at the respective facility.

2.1 Return and Forward Link Processing Operations

The DIF return and forward link processing services provide for the receipt, capture, processing, and transfer of digital data that conform to applicable CCSDS communication services recommendations. EDOS acts as an interface between the EGS and the SN. Return link processing removes communications artifacts and provides computer ready data sets to the EGS. Telecommand link and physical layer services are provided for forward link data received via Ecom from the EOC and delivered to the EOS spacecraft via the TDRSS. Data capture is provided for return link data received from spacecraft via the TDRSS. Return link services include real-time and rate buffered Path and VCDU services. Return link data can be delivered to any appropriate EGS destination. All data handling services, return and forward link, include data quality assurance and accounting.

The DIF processing services are highly automated data-driven services using management information provided by the DIF operations management service. The management information represents service requirements for data processing and defines the parameters the DIF will use to process and deliver data. The DIF incorporates built-in test capabilities in support of on-line operations.

The DIF provides the following capabilities for the processing and delivery of mission data:

Data Capture. All return link data, including fill data, are captured and stored for 30 days after receipt by EDOS for use in recovery processing.

Return Link Real-time Processing. Real-time processing receives and processes all return link data, and delivers CCSDS Service Data Units (SDUs) (e.g., Virtual Channel Data Units (VCDUs), CCSDS packets) to EGS elements with minimized processing delay through EDOS, as required.

Playback Processing. Playback processing restores "as recorded order" to spacecraft tape recorded data received by the frame synchronization function in reverse order. Playback data received in forward order are processed and stored as received. Transfer of playback data commences after the completion of the TDRSS Service Session (TSS).

Rate Buffering. Rate buffering is the process in which data from an EOS spacecraft, transmitted to the ground during a TSS, are completely received by EDOS at one data rate and transmitted to destinations at negotiated reduced data rates.

Forward Link Real-time Processing. The DIF provides the capability to process forward link data in support of CCSDS Telecommand services.

2.2 Production Data Processing Operations

The DPF provides production data handling services for return link mission data received from the DIF. Production data handling services annotate and remove, when possible, communications artifacts and data anomalies due to spacecraft operations. These services include production data processing and quick-look data processing.

Production Data Processing. Production data processing of return link CCSDS packet data is the process in which packets from one or more TSSs are sorted by applications process identifier (APID), forward ordered by packet sequence count and time, and quality-checked. A production data set (PDS) consists of production data processed packets, quality and accounting summary information. Production data sets have redundant and previously processed packets deleted, and may be delimited by time interval, number of packets, number of octets of data, or TSS boundary.

Quick-look Data Processing. Quick-look data processing is similar to production data

processing except redundant packets are not removed and the content of a quick-look data set (QDS) is limited to either all packets received for a single APID during one TSS or all packets in one TSS in which the quick-look flag is set in the packet secondary header. Quick-look data processing may be performed on up to five percent of return link data received over a 24-hour period. Quick-look data processing demands in excess of five percent will be detected and the EOS System Management Center (SMC) will be notified about possible degradation in EDOS support. The packets contained in a QDS are included in production data processing. Specific operational requirements for quick-look data processing will be contained in the Operations Agreement (OA) document between the EGS element and EDOS.

2.3 Data Archive Operations

The DPF data archive service provides a long-term storage capability as a Level 0 data backup to the DAACs. The PDSs created by EDOS are stored for the life of EOS plus 3 years. Retrieval of archived data is expected to occur infrequently. Retrieved PDSs together with quality and accounting information are delivered to the requesting DAAC as Archive Data Sets (ADSs). The data archive service can recover from lost or damaged PDSs by receiving and storing DAAC to EDOS Data Sets (DEDSs) from a DAAC.

2.4 Operations Management

EDOS operations management services provide the management capability for all EDOS resources and services. These services provide highly automated system monitoring and control capabilities and manage the operation of EDOS services.

The DIF and DPF operations management (OM) capabilities monitor and control the systems that implement the services of the respective facility. These management capabilities receive, consolidate, and analyze system performance data as well as respond to service requests received by the EDOS service management (SM) capability. The DIF and DPF OM capabilities transfer service status information to the EDOS SM capability for service reporting.

2.5 System Support Operations

System support services are provided at all three EDOS facilities. These services include the capabilities for integration, test, and verification (IT&V), fault isolation support, and maintenance support for the processing services at each facility.

The EDOS IT&V capability provides tools to support EDOS and external testing. Maintenance support capabilities at each facility provide tools for managing the maintenance of systems at the respective facility. The EDOS IT&V and maintenance activities are coordinated by the system support service at the SEF.

2.6 Sustaining Engineering Operations

The EDOS sustaining engineering capability provides an environment for the development of system enhancements, trouble-shooting and hardware and software updates to the operational system. The environment supports tracking of the operational system performance and maintenance history, and the development and evaluation of system changes and the evaluation of new technologies and requirements.

3.0 Operations Scenarios

The EDOS operations concept includes several operations scenarios to clarify system and interface functional interactions. A typical scenario describes real-time return link operations during a TSS.

3.1 Real-time Return Link Data Processing Scenario

a. TGT transfers Channel Access Data Units (CADUs) from each TDRSS service channel to the designated DIF TGT ports. Data capture recognizes data are present and starts storing CADUs, including fill CADUs. (The following steps apply to each TDRSS service channel)

b. VCDU service. The return link processing (RLP) service frame synchronizer recognizes CADU frame sync pattern, performing bit inversion and CADU reversal as required. The frame sync is stripped off, status data is collected and sent to the DIF OM, and the VCDU is passed to the Reed Solomon (R-S) decoder. The R-S decoder decodes the applicable portion of the VCDU (header and/or entire VCDU),and strips off the R-S code. The RLP deletes fill VCDUs, generates an EDOS Service Header (ESH), collects status data for the ESH and sends status data to the DIF OM. Time and date of CADU receipt by the DIF is added to the ESH and the ESH is appended to the VCDU, creating a VCDU EDOS Data Unit (EDU). Services for the VCDU are determined in the RLP by checking the service requirements for the VCDU-ID [spacecraft ID (SCID) and virtual channel ID (VCID) located in the VCDU header]. Command Link Control Words (CLCWs) are extracted from VCDUs and transferred in real-time with the source VCDU ESH to the EOC. VCDU EDUs not requiring Path service are stored. VCDU EDUs requiring real-time service are transferred to the requesting EGS elements. VCDU EDUs requiring Path service are sent to the Path service processor.

c. Path service. VCDU EDUs are disassembled: packets are extracted and reassembled. Packet fragments with headers are filled out with fill data. The source VCDU ESH is retained, packet quality and accounting data are added to the ESH and the ESH is appended to each related packet, creating packet EDUs. Packet EDUs are then stored. Packet EDUs requiring real-time service are concurrently transferred to the requesting EGS elements.

d. VCDU EDUs and packet EDUs requiring TSS post-operations services are stored in a manner that facilitates rapid access, in order to start transferring multiple EDU files to destinations within 5 minutes. Stored files are identified for the type and priority of post-TSS processing needed: quick-look data processing, playback processing, rate buffering, and production data processing).

e. The DIF OM collects service processing status data from each of the DIF processing services during processing activities. During a TSS, the EDOS SM collects these data from the DIF OM and also Ecom's service status data, compiles the data into a customer operations data accounting (CODA) Report, and sends the report to the EOC, nominally every 5 seconds, during the TSS. SN performance messages are received from the NCC and used at the DIF along with other status data and SN schedule data by the operators for fault isolation. The DIF OM also does quantitative and quality determination for DIF operators and for TSS summary reporting. The EDOS SM also receives TGT performance data via the NCC. The EDOS SM operator compares TGT performance parameters with the RLP status data for fault isolation.

4.0 Operations Concept Development Approach

Traditionally, the responsibility for drafting an operations concept for a new system lies with one or two knowledgeable people who have had some experience in the past with such documentation and who have participated in high level requirements meetings and discussions with the system project personnel. The concept is drafted and distributed for review. After several draft revisions, the concept eventually gets honed into an acceptable product. At best this method is a compromise of ideas (concept features) of how the system should operate. At worst, the concept may be lacking in support of key requirements. This could be caused by reviewers misinterpreting the concept or the writers misinterpreting the reviewers' intentions in their comments. There is more chance for this to happen if a new system is different or more complex than existing systems. Reviewers may not be persistent enough in their reviews to ensure compliance with their change requests. The traditional method was initially tried in developing the EDOS operations concept. After several unsuccessful attempts to satisfy reviewers, a facilitator approach to the development was tried.

The EDOS Project formed an operations concept working group (OCWG) consisting of EDOS systems engineering team (SET) members. The OCWG was composed of government and contractor project support personnel who had participated in Phase B studies and were responsible for the requirements analyses for EDOS. The OCWG met regularly and representatives of systems with EDOS interfaces were invited to participate in the concept discussions. Each member was allowed to express his or her ideas and critique the other members' ideas. Members shared facilitating of the meetings. This avoided over dependence of any one person and also avoided the "leader" instinct of some of the members. It also increased the homogeneity of the meetings. Agendas were followed at each meeting. A member was delegated to write the minutes (including concepts developed). These minutes were reviewed in detail at the next meeting prior to proceeding with new business/concepts. This gave the members an opportunity to correct or improve the concept as recorded and reach further agreement. An important feature of this method is that a consensus was reached among the responsible project personnel before a draft document was started. This meant that the critical part of the concept development was basically finished before documentation began. Another feature was that each member's technical knowledge and familiarity with the system requirements were enhanced during the process. This was important during the next phase of system development which was the requirement analyses for the EDOS specification. During this phase, the operations concept was used to understand what requirements were needed for the specification. If the concept was found lacking, the facilitator method was used to develop new or improved concept features. Since this method had been used previously and by the same personnel, it was easy to re-institute the process.

111199 N95-17614

354337 P-7

CONCURRENT ENGINEERING: SPACECRAFT AND MISSION OPERATIONS SYSTEM DESIGN

J. A. Landshof*, R. J. Harvey*, and M. H. Marshall**

The Johns Hopkins University
Applied Physics Laboratory
Laurel, Maryland 20723-6099

Abstract

Despite our awareness of the mission design process, spacecraft historically have been designed and developed by one team and then turned over as a system to the Mission Operations organization to operate on-orbit. By applying concurrent engineering techniques and envisioning operability as an essential characteristic of spacecraft design, tradeoffs can be made in the overall mission design to minimize mission lifetime cost. Lessons learned from previous spacecraft missions will be described, as well as the implementation of concurrent mission operations and spacecraft engineering for the Near Earth Asteroid Rendezvous (NEAR) program.

Introduction

The traditional approach of system development (requirement definition, specification development, preliminary and detailed design, fabrication, and test) is a long, cumbersome, and frequently costly process. Current system engineering techniques for system development such as concurrent engineering and rapid prototyping can be much faster, and, consequently, cheaper. There may be increased risk in this approach, however, the benefits generally outweigh these risks. In cost and schedule constrained programs such as Discovery programs, higher risk must be tolerated to achieve the goals of faster, better, and cheaper.

Concurrent engineering is defined here as the simultaneous development of two or more interacting systems from the earliest stages of the system life cycle through the design and development process. System engineering includes in part the allocation of system requirements to subsystems, and when two or more subsystems' requirements overlap, or when a system-level requirement could be handled by two or more subsystems, concurrent engineering techniques can be used to arrive at an optimal solution. This paper will describe what is meant by concurrent engineering as it applies to the development of space systems, focussing on the concurrent design and development of a spacecraft and the mission operations system that will be used to operate it on orbit. The benefits and costs of concurrent engineering in this application will be discussed, and concurrent engineering methods will be presented. Then, specific examples of lessons learned from past space system development programs at the Johns Hopkins University Applied Physics Laboratory (JHU/APL, or APL) will be presented, along with a work-in-progress snapshot of concurrent engineering in practice on the Near Earth Asteroid Rendezvous (NEAR) mission .

Concurrent Engineering of Space Systems

A space system includes a spacecraft and the systems with which the spacecraft will be operated once it is in space (the mission operations system, or MOS). At the very start of a mission, requirements are allocated between the spacecraft and the MOS (e.g., existing MOS infrastructure may require a certain frequency for uplink and/or downlink, requiring the spacecraft telemetry system to be built in compliance thereof), ideally by a mission system engineer. After these top-level allocations are made, requirements in both systems are further allocated to subsystems within each, by the cognizant system engineers. Even when requirements are allocated along clear lines, simple decisions in one system can have

* Member, Senior Professional Staff
** Member, Principal Professional Staff

great affects on the other. A mission system engineer is crucial to resolve conflicts, and to make decisions as to what requirements should be done where.

As the spacecraft and the MOS are being designed, constant communication between the two development activities is crucial in order to end with a space system that works well as a whole. Therefore, the communications between spacecraft subsystem design efforts and MOS design efforts must include design decisions as they are being made. Communication must occur at the lowest possible level, between individual engineers responsible for subsystem design if possible. In some cases, relatively minor changes in spacecraft or instrument design can significantly save in operations costs. For example, thermal and power robustness may eliminate the need for complex analysis of every maneuver sequence, saving time and money in the development of sequence uploads.

A mission level system engineer should be designated at the start of a program by the program office, with the capability and responsibility to perform requirement tradeoffs at a high level. Too frequently, all flexibility and operability is pushed onto the ground system and mission operations functions to save development costs in the spacecraft. This is often the correct approach (complexity versus reliability tradeoffs in the spacecraft can be prohibitive), however, in the current budget environment, this is not always the optimal approach.

Benefits

There are many benefits to designing major elements of space systems concurrently. Concurrent engineering allows optimal systems solutions across disciplinary boundaries, with the added bonus of often doing so in less time at a lower cost. One inevitable outcome is the education of engineers about each other's systems. In the case of spacecraft subsystems and mission operations, the better mission operations understands the spacecraft, the more safe, efficient, and reliable mission operations is going to be. The better trained and educated the mission operations team is the better they will be able to respond quickly and correctly to solve any anomaly that might arise on the spacecraft.

In the development arena, concurrent engineering can allow for more flexible response to changes in requirements. If a requirements change is forced on the spacecraft late in the design cycle, it is often very costly to modify flight designs. If the MOS is able to respond to the requirements change, costly delays in the spacecraft development program are often avoided, albeit at some potential expense to the MOS development effort.

Finally, if a spacecraft is designed from the outset with operability in mind, fewer people may be required to operate it. Since personnel are usually the driver for mission operations post-launch costs, lowering the number of personnel required to operate a spacecraft can dramatically reduce mission operations', and thus the overall program's, costs.

Costs

Concurrent engineering does not come without costs. There is often an increase in the time required for communications between development groups. This is especially true early in the program, during conceptual and preliminary design phases when teams may be small and design time precious. During the system development period, subsystem teams can not just build their box in isolation. They must continue to work with other elements as designs are solidified, to ensure a working system at the end.

Finally, perhaps the most critical time consuming effort is in convincing all team members that concurrent engineering is a worthwhile effort. Concurrent engineering runs counter to traditional subsystem development processes. Often, concurrent engineering can seem to overstep 'turf,' when for instance a mission operations person requests changes in the command system design. A strong mission systems engineer can smooth the turf battles, but it is time consuming. Once everyone realizes that the true end product is the space system, not a subsystem, these concerns tend to go away.

Methods

Two methods are currently being used at APL by the mission operations organization in conjunction with spacecraft development programs to enable the concurrent engineering process for space system development. These methods are the identification of a spacecraft specialist in the early prelaunch phase, and the development of the spacecraft ground system ICD.

The Spacecraft Specialist

The spacecraft specialist is responsible for providing the bridge between mission operations and the spacecraft development team. This person should ideally have both a spacecraft hardware and an operations background.

During the initial phases of the mission, the spacecraft specialist job is to work with the spacecraft system engineer, and subsystem designers, to ensure operability is a consideration in all design phases. It was found on previous programs that just asking subsystem designers to think about operations did not work -- someone was needed, paid for by mission operations, whose job was to look over the shoulders and comment on designs as they evolved.

Early on, as mentioned above, some subsystem designers felt that mission operations was intruding into their territory. As time went on, though, almost all came to understand and appreciate, and in some cases even demand, the perspective brought to the table by the spacecraft specialist. Critical to this success, however, is the credibility of the spacecraft specialist.

The Spacecraft/Ground System ICD

One of the primary products of the spacecraft specialist in the early program phases is the Spacecraft/Ground System Interface Control Document (ICD). This document captures the interface between the ground, both the GSS and the MOS, and the spacecraft, and should be completed before the spacecraft Critical Design Review (CDR). For each spacecraft system, and subsystem, the ICD defines commands, telemetry, and operating rules, as they are known at that point in the program. With this document in hand, the ground system development team can proceed to build the command and telemetry processing system, and the spacecraftdevelopment team can proceed with the development, integration and test of their subsystems. Communication between the teams is still required, though; the ICD is the beginning of the process, not the end.

Like most documents, the Spacecraft/Ground Systems ICD is most useful during its development, not by its use. Requiring that both mission operations and the spacecraft subsystem personnel think about command formats, telemetry, and operating rules very early, in order to develop the ICD, is the very essence of concurrent engineering.

Space System Development: Past Experience

Over the years, the Applied Physics Laboratory has built over fifty spacecraft. Virtually all of these were one-of-a-kind spacecraft built for a specific research purpose. With this history comes an institutional way of doing business. Programs have tended in the past to be very focused on the spacecraft. As current missions have required more of a mission focus, the institutional ways of doing business are changing. On previous missions, there were a number of areas where concurrent engineering might have reduced the cost of mission operations development and implementation, helping to reduce overall mission costs. Areas of spacecraft design where mission operations' input early on might have proven beneficial include spacecraft commanding, telemetry, onboard memory management, onboard data processing, and the testing and testability of some subsystems. Examples are given below of specific lessons learned on recent APL space system development programs. In some cases these examples refer to the standardization of designs throughout the spacecraft, while others refer to particular design change recommendations to make operations more efficient.

Commanding

Mission operations' sole connection to the spacecraft after launch is through the command and telemetry links. The only path for mission operations to affect anything on the spacecraft is via commands from the ground. In the early days of space, spacecraft were launched with fixed timelines of activities; no changes from the ground could be made. Now, of course, spacecraft are built to respond to ground commands to carry out activities. The development of the commands to be sent to the spacecraft is, in fact, the primary focus of mission operations today. Therefore, designing the command interface to the spacecraft with operability offers perhaps the best opportunities for a more easily operable spacecraft, which in turn can reduce the size of mission operations considerably. One particular area of interest is in the types and formats of the commands themselves.

A standard command format being mandated throughout the spacecraft would enable the mission operations team to develop a standard mechanism for the automated generation of commands. Hard-coded workarounds in flight software that require special command types not only escalate the cost of development, but reduce the speed of an automated command generation process. A standard command format should be applied to serial data commands, which might include an "opcode" at the start of the data to indicate the command type or functionality. Mode change commands should not be of the type where each bit addresses some particular function; to change a single element with such a system, each bit must be respecified to its current state. This is a nightmare for mission operations! As an example, one program had a command design where four bits of a serial data command represented the enabling and disabling of four different data formatters. Every time one particular formatter was to be enabled, the previous state of the others had to be known. If the wrong state had been assumed, the command may have inadvertently disabled one formatter that should have remained enabled. This could have caused something as critical as communication of spacecraft housekeeping data suddenly being lost when science data was enabled for on-board recording. If the function of controlling each formatter had been made a separate "opcode," each formatter could have been controlled individually without having known each other's previously commanded state. The creation of the command loads would have been easier, the checking of those commands loads more reliable, and mission operations workload reduced significantly.

Standard command formats also may reduce mission operations development costs by making spacecraft state determination and tracking easier. Lower fidelity models of the onboard processor, its memory, and its state would still provide all necessary functions, but require less design, development, and maintenance, thereby reducing costs. Automated command generation schemes also can reduce personnel requirements.

Telemetry

To assist in the area of spacecraft control and performance assessment, every command, whether executed in real-time or delayed, must have telemetry which allows for the verification of proper execution or rejection. For serial data commands some means of verification are required (at a minimum the data should be reflected back into telemetry). This is essential in determining that a command was not only correctly received by the command system and transmitted from the command system to an onboard subsystem, but was in fact properly executed by the intended subsystem.

Tracking what the spacecraft has done since the last contact with the ground is very important for mission operations. To support this requirement, the spacecraft should have a command history buffer. The size of this buffer should be changeable by uplink command. Stored commands and commands from macros should be logged, but not necessarily data loads. Downlink of the buffer may be by ground command, to conserve downlink bandwidth. This history buffer capability allows for the assessment that a command was rejected for reasons other than not being properly transmitted from the command system. This allows

mission operations to assess spacecraft health more easily and quickly, an important factor especially on low earth orbiters with short ground contact durations.

Memory Management

Other means of standardization include the uplinking and downlinking of on-board processor's memory locations or specifically, the use of data structures. Data structures allow the loading of a processor's memory without knowing the exact locations, which could change should the processor's code be re-linked. The functionality of the processor's software should allow for the uploading and downloading of these data structures by a specific ID number. On a recent mission this capabiltity was not built into the flight software, requiring the downlinking all of the data structures at once as opposed to each data structure individually by ID. This created the requirement for additional ground software that would search through the entire downlink and find the particular one of interest.

Onboard Data Processing

On spacecraft where housekeeping data is not continuously recorded, there should be a capability for a buffer which allows the routine periodic sampling and storage of critical parameters. Most likely, throughout the life of a spacecraft's mission, different parameters will vary in their criticality. Therefore, the capability should exist for allowing ground commands to change which parameters are sampled and their periodicity. The buffer obviously must have a particular size limitation, so in cases where data will be lost because it cannot be downlinked for long periods of time, it would be highly desirable from a mission operations assessment perspective to be able to download this data to the on-board recorder for later retrieval. On a previous spacecraft a similar type buffer was limited to the sampling of certain unchangeable parameters and its capacity allowed for up to 5 orbits of sampling at a rate of one sample per 200 seconds. The rate was changeable; however, as the sampling rate was increased, the amount of time between required downlinks was reduced. In these cases, it would have been advantageous to have the capability of transferring it to the on-board recorder. Also, as the mission progressed, certain parameters which were "hard-coded" became invalid. In these cases it would have been beneficial to replace those with other critical parameters.

Such a capability would give mission operations insight into spacecraft state between contacts and allow performance assessment and trending of critical parameters as they vary throughout a mission.

Testing

Also in the area of performance assessment, for any processor or recorder (either solid state or tape), there should be a method of loading a standard data test pattern in each processor or on each tape such that it may be downlinked through telemetry and run through a bit-by-bit comparison to a ground image of the same pattern to certify memory validity and periodically measure bit error rates.

Summary of Lessons Learned

In all of these cases, if the Mission Operations Team was involved in the specification of spacecraft design requirements, the overall mission operations cost would have been reduced through both a lowering of system development costs and an increase in efficiency in the performance of mission planning, control, and assessment tasks.

Use of Concurrent Engineering on the NEAR Mission

The Near Earth Asteroid Rendezvous (NEAR) program was officially turned on in December of 1993. Prior to that, a small study team had been working on the conceptual design of the mission and the spacecraft. In August of 1993, mission operations was asked to provide input as to spacecraft design considerations for the NEAR mission which would enhance operability. Below, the input provided for spacecraft design features are listed, and the current status of each is described. Following that, other activities highlighting the use of concurrent engineering on NEAR are described.

It must be strongly emphasized that the NEAR space system is still being designed; as of the writing of this paper (July 1994) both the spacecraft and the mission operations system are in the design and develop-

ment stages. What follows is a snapshot of work-in-progress; by the time of the SpaceOps '94 symposium, (November 1994) the spacecraft will have passed its critical design review, and the presentation for this paper will update the following material.

Mission Operations Inputs for NEAR Spacecraft Design

The following items (numbered) were listed by mission operations in August of 1993 as design considerations for the NEAR spacecraft, and can be seen to come from the experiences described above. They are listed in no particular order:

1. "Spacecraft and RF system must have power/thermal margin to transmit continuously for 8-hour contact. If a contact is delayed, actual transmission time may be longer"

Current Status: The NEAR spacecraft can transmit continuously during all mission phases.

2. "The spacecraft will have a data summary area in the command and data handling (C&DH)system computer. The 'Data Summary' requirements include:

– At least 5 data points for each important telemetry parameter (high, low, average, time of high, time of low)

– A variable length 'Anomaly Data' area where data triggered/ written by the autonomy system is stored."

Current Status: The NEAR spacecraft C&DH software requirements specification includes all of the above requirements.

3. "The NEAR Solid state recorder (SSR) memory must be non-volatile."

Current Status: The NEAR spacecraft solid state recorder memory is volatile - shutting off the power causes the data to be lost. However, the power should never have to be turned off, so this is not a seen as a critical issue by mission operations. The SSR is a purchased component, with an existing design, and designing a new recorder would have been cost and schedule prohibitive.

4. "Realtime telemetry must be available to the SSR and telemetry system simultaneously."

Current Status: The NEAR spacecraft can both record and downlink housekeeping ('realtime') data simultaneously. This feature can be used to prevent the loss of spacecraft housekeeping data in the event of a transmission error.

5. ."The SSR must have random access capability. Downlink of selected time periods of selected parameters is required."

Current Status: The NEAR solid state recorder has some capability for random access, but not by time and parameter. The ground will have to model data recording functions in order to know what particular SSR memory addresses to downlink for particular data. The onboard data rates of all instruments and subsystems are controlled by ground command, so this is not seen as a problem.

The following items concern the onboard spacecraft telemetry processing and anomaly detection and correction processes, collectively known as autonomy

6. "Autonomy rules should include chaining (i. e. if A is true, then check if B is true, then take some action) and arithmetic (i.e. allow the multiplication of a voltage and current telemetry parameters to check on power consumption."

Current Status: The onboard autonomy does not allow for arithmetical functions on telemetry, but it does allow for limited logical checks (ANDs and ORs of particular telemetry values). Mission operations and the flight software team are still negotiating this requirement.

7. "Autonomy should have access to SSR (to support onboard trending in case of fault detection)."

Current Status: This has not been designed into the system.

8. "Autonomy should be able to write data and conclusions to an'Anomaly Data' area of the 'Data Summary'."

Current Status: The onboard processor will capture that information which caused a

particular autonomy rule to be triggered. The data will be stored in a known location, and can be downlinked.

9. "To minimize commanding, a data region accessible to commands is necessary."

Current Status: The intent here was to reduce the amount of commanding required by allowing mission operations to uplink changes in data for previously transmitted commands. As the design matured, mission operations and the software team agreed on a scheme utilizing onboard sets of commands, called macros, invoked by a smaller set of uplinked commands. All macros are uploadable, changeable, etc., and can be used over and over again. A great deal of preparation will go into the design of the macros to ensure their reusability.

Other NEAR Concurrent Engineering Activities

Significant interaction between the spacecraft and mission operations systems design efforts is occurring in the areas of flight and ground software. Regular meetings are held, conducted by the flight software system engineer, the MOS software lead, the mission operations manager, and the cognizant technical leads for specific subsystems under discussion each meeting. Requirements are negotiated, specifications reviewed, and implementation issues aired and resolved among all the parties. Additionally, the mission operations manager has been asked to be on the review panel of the flight software preliminary design review.

Mission Operations and the spacecraft design team are working together shoulder to shoulder, in many other areas. Load management schemes, maneuver algorithm design, etc. are all being worked on together to make sure the final space system design is a good one. All teams seem to recognize the importance of strong spacecraft/operations interaction at this important stage of the NEAR mission.

Conclusions

Concurrent engineering is a technique which can work to provide a better space system, in less time, while substantially reducing total program costs. Inherent advantages of teams working together are gained, at the cost of a little more communication and flexibility. Based on our belief in the benefits of concurrent engineerirg and lessons learned from previous space missions, the NEAR mission operations team is taking an aggressive (but tactful!) approach to concurrent engineering of the spacecraft and the mission operations system. Lessons learned from past space systems development programs have given the NEAR project team a leg up, and we are using those lessons to our advantage. The NEAR project is using concurrent engineering as the basis for the system design, and both the spacecraft design team and mission operations are profiting from the close working relationship. The payoff to date is evident; we are confident that future payoffs of this approach will enable NEAR post-launch costs to be constrained to an optimal level.

Author Index

MM - Mission Management, OP - Operations, DM - Data Management, SE - Systems Engineering, SD - Systems Development, * *Presented in Poster Session*

Author Index

MM - Mission Management, OP - Operations, DM - Data Management, SE - Systems Engineering, SD - Systems Development, * *Presented in Poster Session*

Author Index

MM - Mission Management, OP - Operations, DM - Data Management, SE - Systems Engineering, SD - Systems Development, * *Presented in Poster Session*

Author Index

MM - Mission Management, OP - Operations, DM - Data Management, SE - Systems Engineering, SD - Systems Development, * *Presented in Poster Session*

Author Index

MM - Mission Management, OP - Operations, DM - Data Management, SE - Systems Engineering, SD - Systems Development, * *Presented in Poster Session*

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1994	Conference Publication

4. TITLE AND SUBTITLE
Third International Symposium on Space Mission Operations and Ground Data Systems

5. FUNDING NUMBERS
500

6. AUTHOR(S)
James L. Rash, Editor

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Goddard Space Flight Center
Greenbelt, Maryland 20771

8. PERFORMING ORGANIZATION REPORT NUMBER
94B00135

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, D.C. 20546–0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
CP-3281
Part 2

11. SUPPLEMENTARY NOTES
This Proceedings is composed of two volumes.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category 17
Report available from the NASA Center for AeroSpace Information, 800 Elkridge Landing Road, Linthicum Heights, MD 21090; (301) 621-0390.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

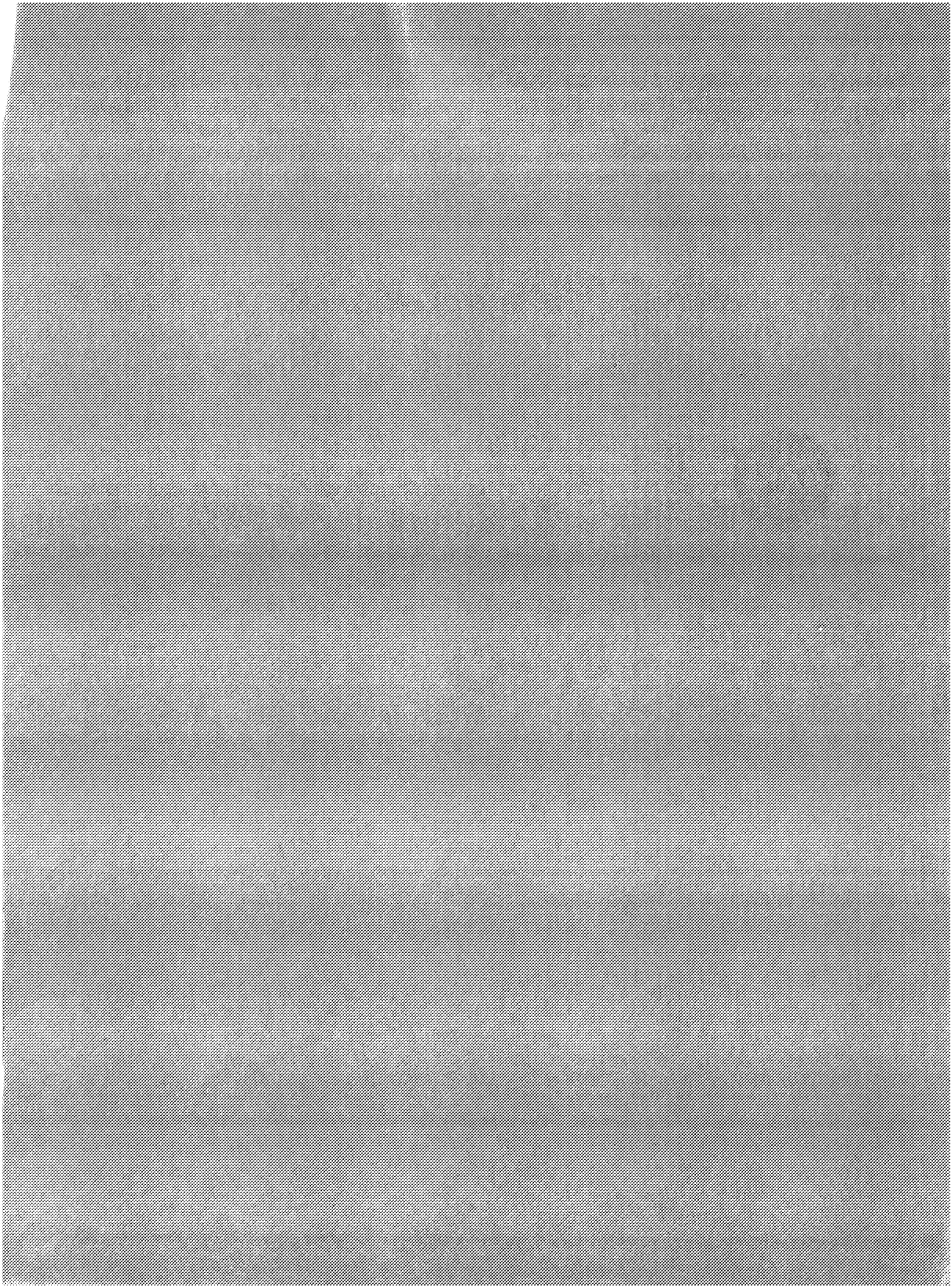
Under the theme of "Opportunities in Ground Data Systems for High Efficiency Operations of Space Missions," the SpaceOps '94 symposium included presentations of more than 150 technical papers spanning five topic areas: Mission Management, Operations, Data Management, System Development, and Systems Engineering.

As stated in the executive summary of the symposium proceedings, the papers "focus on improvements in the efficiency, effectiveness, productivity, and quality of data acquisition, ground systems, and mission operations. New technology, techniques, methods, and human systems are discussed. Accomplishments are also reported in the application of information systems to improve data retrieval, reporting, and archiving; the management of human factors; the use of telescience and teleoperations; and the design and implementation of logistics support for mission operations."

14. SUBJECT TERMS	15. NUMBER OF PAGES
Data Handling, Telemetry Processing, Mission Planning, Orbit Determination, Standards, Modeling, Communications Networks, Communications Systems, Ground Based Data Acquisition Systems, Spacecraft Communications, Spacecraft Command, Spacecraft Tracking, Spacecraft Control (Communications), Expert Systems	743
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

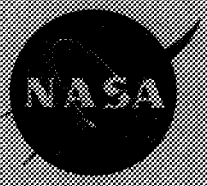
PRECEDING PAGE BLANK NOT FILMED



National Aeronautics and
Space Administration
Code JTT
Washington, D.C.
20546-0001

Official Business
Penalty for Private Use, $300

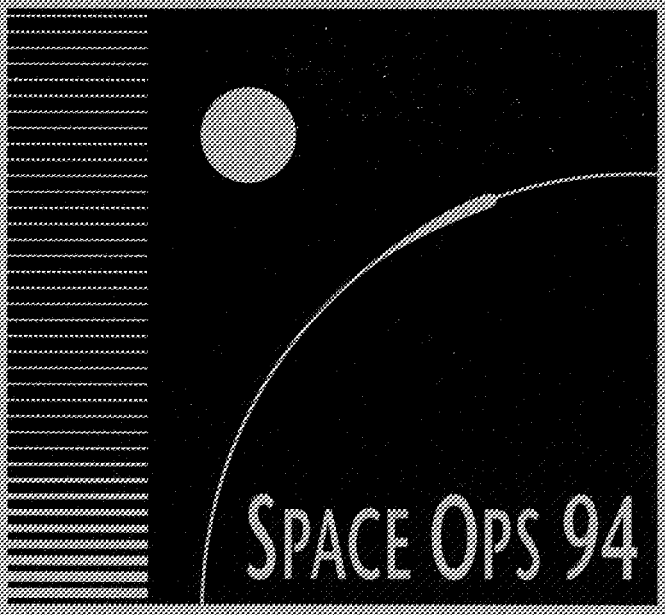
SPECIAL FOURTH-CLASS RATE
POSTAGE & FEES PAID
NASA
PERMIT No. G27



POSTMASTER: If Undeliverable (Section 158,
Postal Manual) Do Not Return

NASA Conference Publication 3281

Third International Symposium on Space Mission Operations and Ground Data Systems Part 2



Proceedings of a conference held at
Greenbelt Marriott Hotel
Greenbelt, Maryland, USA
November 15-18, 1994

(NASA-CP-3281-Pt-2) THIRD INTERNATIONAL SYMPOSIUM ON SPACE MISSION OPERATIONS AND GROUND DATA SYSTEMS, PART 2 (NASA. Goddard Space Flight Center) 700 p

N95-17531
--THRU--
N95-17614
Unclas

H1/17 0031936