PROCEEDINGS
of
1986 CONFERENCE
on
ARTIFICIAL INTELLIGENCE APPLICATIONS

MAY 15, 1986

(NASA-TM-101873) PROCEEDINGS OF 1986 CONFERENCE ON ARTIFICIAL INTELLIGENCE APPLICATIONS (NASA) 144 p

N90-70235

Unclas

00/63 0233130

NASA GODDARD SPACE FLIGHT CENTER
GREENBELT, MARYLAND

PROCEEDINGS

of

1986 CONFERENCE

on

ARTIFICIAL INTELLIGENCE APPLICATIONS

MAY 15, 1986

NASA GODDARD SPACE FLIGHT CENTER
GREENBELT, MARYLAND

SPONSORED BY CODE 514, SPACECRAFT CONTROL PROGRAMS BRANCH
AND BENDIX FIELD ENGINEERING CORPORATION

Message from the Co-Chairmen

The conference co-chairmen take pleasure in welcoming the attendees of the 1986 Conference on AI Applications. We hope that this conference will succeed in bringing a few current expert system applications to your attention, with the expectation that such exposure will help "demystify" the software products labeled "expert systems".

The papers presented at this conference focus on AI applications related to space science. Some of today's papers pertain to mission operations support and to the support of planning for spacecraft command and science operations. Other papers discuss applications areas of general interest to expert system implementers at the Goddard Space Flight Center. We hope that you will find these papers informative and interesting.

We also have a few demonstrations of practical AI applications and a demonstration of the new NOVEX FORTH chip. We encourage you to take advantage of the opportunity to view these implementations.

The co-chairmen of this conference would like to express their appreciation to Patricia Lightfoot of the NASA Goddard Spacecraft Control Programs Branch and to Jerry Barsky of the Bendix Field Engineering Corporation for making this event possible. We would also like to thank some co-workers for assisting with conference preparations for the past few months, especially Phil Marino, Ellen Stolarik, Ron Littlefield, David McLean, and Carolyn Dent.

David Beyer
Software and Engineering Services,
Bendix Field Engineering Corporation,
Lanham, Maryland

William Macoughtry
Spacecraft Control Programs Branch,
NASA Goddard Space Flight Center,
Greenbelt, Maryland

TABLE OF CONTENTS

Message from the Co-Chairmen.....ii

Papers for Presentation

Keynote Address.....1
Patricia Lightfoot

The Development of an Intelligent User Interface for NASA's Scientific Databases.....8 51
William Campbell and Larry Roelofs

AI Goes FORTH.....20 52
William Dress

The Expert Project Management System.....28 53
Barry Silverman

An Expert System for Ground Support of the Hubble Space Telescope.....43 54
Don Rosenthal

An Expert System Prototype for Aiding in the Development of Software Functional Requirements for NASA Goddard's Command Management System: A Case Study and Lessons Learned.....55 55
Jay Liebowitz

The Design and Application of a Transportable Inference Engine (TIE1).....76 56
David McLean

A Prototype Expert System in OPS5 for Data Error Detection.....91 57
James Rash

Multiperspective Analysis and Testing of Expert Systems.....106 58
Terry Bollinger

Papers of General Interest

Simple Methods of Exploiting the Underlying Structure of Rule-Based Systems.....116 59
James Hendler

DIOGENES - Expert System for Extraction of Data System Requirements.....122 510
Robert Hobbs and T. Patrick Gorman

Omit
TD
P. 8

AI Challenges for Spacecraft Control Programs

Patricia Lightfoot

Spacecraft Control Programs Branch
NASA Goddard Space Flight Center
Greenbelt, Maryland

Welcome to the 1986 Conference on Artificial Intelligence at the Goddard Space Flight Center. This conference is a step towards uniting the research in artificial intelligence with the development of systems to be used for support of future NASA missions. The goal of this conference is to share experiences about the development of artificial intelligence applications, to promote the utilization of artificial intelligence technology in spacecraft and experiment command and control systems and mission planning systems, and to provide a better understanding of the plans of others for the use of AI technology.

The Spacecraft Control Programs Branch is responsible for the development and operation of the GSFC data system components which provide mission support for planning and command of science and spacecraft operations. This support is provided to project experiment operations facilities and spacecraft controllers and is operated to coordinate planned command operation with the real-time support provided by the control centers. The Branch's facility, The Command Management Facility, serves as the focal point for receipt of experimenter inputs and coordination of experimenter requirements with spacecraft requirements. The facility also provides for distribution of planned operational timelines to various experimenters.

Historically, command and control functions have been implemented as computerized systems with operational procedures under human control. The computerized functions have been those that could be defined and specified in an unambiguous manner. Furthermore, these functions had to be definable months or years in advance of their actual use. Examples of command and control functions that are typically computerized are algorithm computations, limited problem identification, input/output data verification, data storage/retrieval, data formatting, and limited problem resolution.

Most command and control problems are handled by procedures and people because of the difficulties in defining all possible courses of action and in devising solutions. As used here, a problem is any issue, an unplanned or unanticipated happening, a conflict in plans, or an unusual occurrence. A problem can also be caused by a good stimuli, such as an opportunity to observe a new celestial phenomena.

The "best" solution to a problem is not always a static solution. Sometimes, the "best" solution changes and is based on the results obtained from using previous "best" solutions to the problem. In such cases, the decision maker generally must (1) consult lists of predefined options, colleagues, experts, or the written records of previous problem occurrences, (2) analyze the available data, and (3) develop and implement a new solution. This process can be time consuming, may not consider all possible solutions, and may not result in the "best" solution. In fact, at times the process may be so complex that no action is taken. For science operations, this could mean a lost opportunity for experimentation with a short lived phenomenon, such as a solar flare. An incorrect decision made under the pressures of time can also result in the loss of data or the loss of the experiment.

Expert systems technology can provide the capability to build and operate more efficient and effective command and control systems, especially for performing functions that require the analysis and solution of complex problems. The requirements for command and control systems to provide this type of decision support can be derived from the operational scenario of the mission to be supported. These requirements include decision support for experimenters and the flight operations team. The requirements for decision support for planning science operations are expanding as future missions will not require that the experiment control facilities be located at Goddard; the experimenters will have more control capabilities at their home institutions which are located across the country and in foreign countries.

At best, the present methods for analysis of spacecraft and experiment command and control problems are unwieldy for a distributed user community. Data are subject to all of the hazards of long distance communication (between people and between computers): data can be filtered so that all relevant data are not available to the users, users can be overloaded with data so that the time required for

individual user analysis is unacceptable, data can be garbled so as to make analysis impossible, and data can arrive too late to help analyze and correct problems. Communication hazards also make coordinated problem analysis difficult; the synthesized problem analysis that occurs in face-to-face meetings is blocked by the communication barriers.

At its worst, the application of present command and control methods for a distributed user community can result in problem analysis with no input from the users. A potential expert system implementation for coordinated problem analysis includes a user's knowledge base containing each user's problem analysis methods, preferred courses of action, and problem impact assessments. Such an expert system implementation could (1) allow the users to update the knowledge base, (2) facilitate problem analysis by using all available data, and (3) provide timely coordination of the problem analysis effort. This implementation could allow the users to participate (via the knowledge base) in the analysis process, reduce communication barriers, and promote coordinated problem analysis. This suggested AI implementation may not be feasible now, but the potential for using expert systems technology assistance in solving this problem exists.

Problem analysis and resolution is difficult in today's environment of centralized operations. It is difficult to obtain optimal solutions for easy problems that have previously occurred, while the solution for a complex problem is even more difficult to optimize. The requests of some experimenters may have to be sacrificed to honor the higher priority requests of other experimenters, and the requirements of some or all of the science users may be sacrificed to ensure spacecraft health and safety. In addition, decision makers may have to make non-optimal solutions because of pressures of time, complexity of coordination, priority of mission objectives, spacecraft health, or priority of experiments. Communication barriers impact the ability to devise a solution if the user community is distributed. A distributed user environment makes the give and take of negotiated solutions almost impossible to accomplish in a reasonable time.

Typically, command and control systems have provided some degree of operator control via a man-machine interface. A man-machine interface generally provides for a somewhat flexible system that is responsive to the needs of the experimenters. Operator control can occur at selected

nodes on the path of command request data that goes from the Principal Investigators to the inflight experiment. Historically, these command and control functions have been implemented to be under human control in order to maximize the capability to respond to a wide variety of external stimuli, stimuli which cannot be defined in advance of their occurrence or which are definable only in terms of a range of values.

A man-machine interface allows system operators (whether experimenters or mission operations personnel) to exercise actions that are correct for most situations. These decisions are typically made by an expert human operator who, through extensive experience with such stimuli, can usually make the correct decision. However, a man-machine interface can also be prone to human error. Some of the weak points of this type of command and control system are listed below.

1. An inexperienced operator is more likely to make a mistake in reacting to the stimuli which normally occur in the course of a mission, but it takes a long time to develop into an experienced operator.

2. The "expert" operator may make an error in the analysis of the stimuli and select the wrong action.

3. The "expert" may not be available at some particular time.

4. It is difficult to capture "why" an expert operator makes a decision so that the logic can be applied to future occurrences.

5. Routine operator decision-making, which could be automated, is not automated because no record is kept to show that the same decision is acceptable each time a specific stimuli occurs.

Artificial intelligence offers the possibility of capturing expertise so it can be used repetitively, and of learning from past experience.

The current method of developing command and control systems may require four or more years of development time. The structured system development methodology, which provides detailed system documentation and which has periodic reviews of the development process, has reduced the dependence on continuity of personnel throughout the

development cycle and from mission to mission. The structured system development process has progressed over the years, but it still some weak points.

1. Highly skilled analysts and programmers are required to translate the system requirements to design and to code.

2. The originator of a requirement has difficulty verifying that the implementation is correct.

3. System modifications must be made by programmers.

4. Programming code is a barrier between the user and the system.

5. Mountains of documentation can be a hindrance to its usability. There are two learning curves: first, to use the documents, then, to use the system.

6. Implementation of new requirements and system modifications in response to changing operational needs are time consuming.

Some goals of the Spacecraft Control Programs Branch are to provide more efficient and effective spacecraft and experiment operations, to improve man-machine interactions, and to facilitate the system development process. Expert systems technology offers some possible solutions to facilitate achieving these goals.

One aspect of achieving these goals is to develop a science and spacecraft operations environment that supports the concept of a distributed user community, which allows the users to possess a high degree of operations flexibility, and which includes each user's requirements as part of any decision that impacts the user. A system to provide this should:

1. Reduce the time required for users to learn the system's operational characteristics.

2. Provide planning methods which reduce the potential for problems.

3. Select the data that is required to analyze and resolve a problem.

4. Rapidly propose workable solutions, with trade-offs, to support near-real-time operations.

5. Provide the rationale for the selected solution to an operational problem.

6. Allow expert decisions to be made by non-experts.

7. Optimize the decision using the user's criteria for acceptable solutions.

Another aspect of achieving these goals involves supporting the development and operation of a system which has a high turnover of user's, as will be the case in the Space Station era. Attainment of this goal can be aided if there is:

1. Tighter coupling of requirements to system implementation to provide better requirements traceability.

2. A natural language implementation (rather than a programmers language) which allows user's to understand the built system.

3. A synthesized system implementation which allows a user to build a customized version of the system or to rapidly modify the system to accommodate new requirements.

4. A method of knowledge transfer that allows tomorrow's users to benefit from the lessons learned today and to avoid rebuilding the wheel and repeating mistakes.

5. A transportable system implementation that is not dependent on specific vendor hardware and that can be executed on a "cheap" hardware configuration as well as an on expensive configuration.

6. An upgradable system implementation which can be executed on future state-of-the-art hardware.

The challenges of using expert systems technology to attain these goals are many. The field is rapidly expanding and moving towards practical applications. Identification of candidate mission support applications is crucial to the successful transition from the academic world.

While there is great enthusiasm for the use of expert systems technology, there are many precautions to be observed. The selection of an appropriate application should consider the potential payoff in order to avoid the development of resistance to the acceptance of this new technology. Systems developed using expert systems technology do not behave as traditionally implemented systems, so new techniques for documenting the design of expert systems are required. Methods of verification which call for "execute every line of code" are not applicable. Lastly, new methods of managing the development of expert systems must be developed since to date most implementations have been in a prototype environment.

There is a relatively small group of individuals who are experienced in building expert systems. For mission support application system development, more people will have to be trained in the use of this technology. Knowledge engineers and expert system developers are required.

Today, several speakers will discuss a variety of experiences with expert systems.

S1-82
233131
12P.

The Development Of An Intelligent User Interface For NASA'S Scientific Databases

William J. Campbell Code 634
NASA/Goddard Space Flight Center
National Space Science Data Center
Greenbelt, Md. 20771

Larry H. Roelofs
Computer Technology Associates Inc.
McLean, Va. 22102

ABSTRACT

The National Space Science Data Center (NSSDC) has initiated an Intelligent Data Management (IDM) research effort which has as one of its components, the development of an Intelligent User Interface (IUI). The intent of the IUI effort is to develop a friendly and intelligent user interface service that is based on expert systems and natural language processing technologies. This paper presents the design concepts, development approach and evaluation of performance of a prototype Intelligent User Interface Subsystem (IUIS) supporting an operational database.

1.0 INTRODUCTION

In the past decade, operations and research projects that support a major portion of NASA's overall mission have experienced a dramatic increase in the volume of generated data and resultant information that is unparalleled in the agency's history. The effect of such an increase is that most of the science and engineering disciplines are suffering from an information glut, which has occurred, not only because of the amount, but also because of the type of data being collected (generally spatial and most often continuous in nature such as images, maps, two and three dimensional drawings and figures).

This information glut is growing nonlinearly, and is expected to continue to grow in this fashion for the foreseeable future. Consequently, it is becoming physically and intellectually impossible to identify, select, and access the most suitable information specifically applicable to the various engineering and research projects of interest. For example, in the earth sciences such vast amounts of data are now being collected and/or are available (e.g., satellite images) that it now exceeds the ability of all the professionals in the field to process, manage and study it. In addition, the number of professionals in the application disciplines is not expected to increase significantly enough to resolve this data problem in the foreseeable future. Thus, the dilemma arises that the amount and complexity of information has exceeded and will continue to exceed, using present information systems, the ability of all the scientists and engineers to understand and take advantage of this information.

Based on the scope, expected growth and dominance of this problem, it is anticipated that the future ability of NASA to function and perform meaningful space and earth related research will be significantly affected by its ability to manage and use its collected information to derive knowledge. Consequently, it is envisioned that dramatically different approaches to data management will need to be taken if earth and space related operations and scientific investigations are ever to take full advantage of the information and data being collected and stored.

Considering the trends of present computer science technologies it would appear that an approach to data management that employs Artificial Intelligence (AI) technologies coupled in a distributed environment with powerful super micro computer based workstations, offers a reasonable solution to resolving many present and future data management problems.

2.0 BACKGROUND

In October of 1984 Goddard Space Flight Center's National Space Science Data Center (NSSDC) initiated a research effort to develop a new generation of data management technologies to support the needs of NASA's future operational, engineer-

ing and scientific programs through the beginning of the twenty-first century. This research effort is the Intelligent Data Management (IDM) Project. The Project's long term research goals are as follows:

- to develop very powerful database management systems using advanced generation technologies,
- to develop intelligent value-added services that will enable users to interact with the most complex database systems with minimal understanding of the systems architecture, stored data, or query language,
- to allow automatic data ingest and maintenance with minimal user guidance and interaction,
- to manage symbolic (characters and letters) and spatial data in the same database systems.

The IDM Project has been organized to address, simultaneously, those areas of data management that could potentially be improved by the applications of artificial intelligence technologies including:

- User Interfacing,
- System management and control,
- Spatial database management,
- Automatic data ingest and system maintenance,
- Advanced database systems using AI design concepts (i.e. dynamic database systems).

A top level diagram of the IDM concept that includes each of the above areas, except dynamic database systems, is shown in Figure 1.

After careful consideration of the operation of database systems, it was determined that the development of an intelligent user interface offered the best chance for near term success while at the same time, providing technical direction for most of the other services in the overall concept. It was envisioned that a database enhanced with a functioning IUI Subsystem (IUIS) would enable scientific and supporting (nondatabase) technical users to access and use the stored information with little or no under-

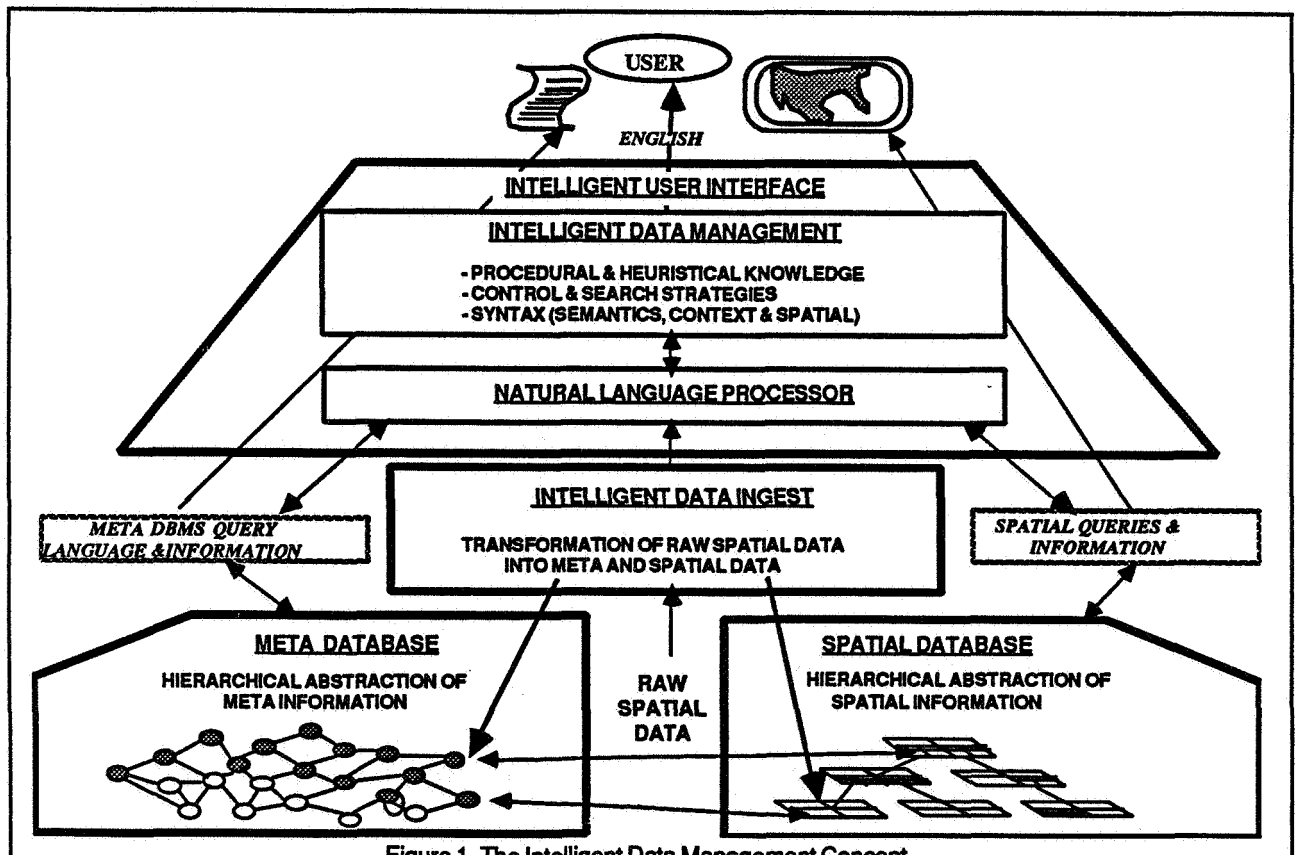


Figure 1. The Intelligent Data Management Concept

standing of the database's architecture, the actual data content or the system's query language.

This paper presents the results of a research and development effort related to user interfacing, called the Intelligent User Interface (IUI) task.

It is believed that the development and implementation of a prototype IUI subsystem will not only demonstrate the concept of applying artificial intelligence to data management, but will also enable NASA to assess the long term applicability of AI technologies to its future data management needs.

3.0 INTELLIGENT USER INTERFACE CONCEPT

The IUI, as envisioned, is a system that will "serve as an intermediary between a database and a user" who has little or no knowledge of the database's architecture, language, or content.[1]

Such an interface would allow the user to operate a database (which means locating, identifying and selecting specific information) in the context of a user's particular knowledge domain, and to do so using a communications medium that is most suitable to the user, such as the user's natural language (i.e., English). The advantages of such a concept are that:

- It can facilitate understanding by specifying the contents and meanings of a collection of data as well as the relation between objects within the data,[2]
- It will be able to support approximate reasoning to infer conclusions that are not explicitly stated by the user, such as an imprecise or fuzzy query which can be stated without mentioning file names or table joins,
- Fuzzy concepts can be expressed by using a fuzzy query,
- It will be able to handle information demands for which the database is used routinely, rapidly and efficiently without any understanding by the user,
- It will be able to communicate with the user in plain English text,
- It will serve as the semantic basis for understanding the user's data needs in conjunction with the natural language query system,
- It will provide a logical representation of the database architecture and information stored in the database to the casual user,
- It can be used to facilitate the understanding and identification of database information from the database operational view to the user related views by the use of conceptual graphs that translate between the database's meaning and the user's meaning of data and sets of data,[3]
- It will provide a physical and logical link between information contained in the metaknowledge database and the spatial database in the overall IDM concept.

To support such a design concept, two ideas need to be developed. First, the creation of multiple views of the database that are both functionally appropriate, from the perspective of a nonexpert database user, while at the same time operationally necessary, from the database's perspective, for supporting the physical storage and management of the data. Second, the creation of a means of moving from one type of view to another using conceptual graphs[4], based on logical inference statements (i.e., rules).

A singularly important feature of such a concept is its role as a mediator between the user and the database and its ability to deal with the duality of the views (or contexts) that must be contended with in order to support both the integrity of the database with its data and the user's understanding of the data and information needs. Using such a concept the IUIS would be able to deal with a user based on his syntax and understanding of the objects in the database, and at the same time understand the database's viewpoint, based on the physical and operational organization of the data and the relationships between objects within the data.

Based on our present understanding of database operations, as well as our reading of the supporting literature, it would appear that there are at least three types of views required for a functionally complete IUIS. These views are:

- Architectural view,
- Multiple application views,
- Operational view (the database design model which can be relational, network, etc.).

The last view, or database operational view, exists

because of the fact that the database exists. And consequently, this view is important because it is this view that all other views must translate to, in order to get information from the database.

Based on the premise that the design of the IUIS requires an awareness of all of the above types of views, one can postulate a model that considers such views collectively linked by intelligent processes which translate between the views. Such a model is shown in Figure 2., and, as one can observe, the primary function of the IUIS is to translate between the various database views and the user in a fashion that best supports the user's needs. An important observation about the model is that the operational view must be included as part of the model even though the view also exists outside the model as the database management system.

3.1 The Intelligent User Interface & Database Views

The multiple view concept upon which the IUIS is based is an enhancement of an earlier database design concept involving what was alluded to as an external level of a database's architecture. This idea of the external level of design is presented very clearly in a popular text book by C. J. Date which describes a database architecture as being divided into three levels: internal, external, and conceptual.[5] The internal level is the one concerned with the way data are stored and, is the closest to physical storage; the external level is the one closest to the user's understanding of the data; and, the conceptual level is a "level of indirection" between the other two levels.

In the context of the IDM concept the intelligent user interface serves as a conceptual level process that is able to provide powerful indirection by being able to deal with the user using his own syntax and logical processes while at the same time understanding the database's design, data content and query language.

The IUIS concept model shown in Figure 2. is based on two types of views to support the conceptual level: an architectural view and multiple user application views. However, because of the broad definitions that can be given to the two types of views, it is necessary as part of the design formulation to specifically limit their scope so as to avoid the uncertainties and ambiguities that can occur when considering database concepts.

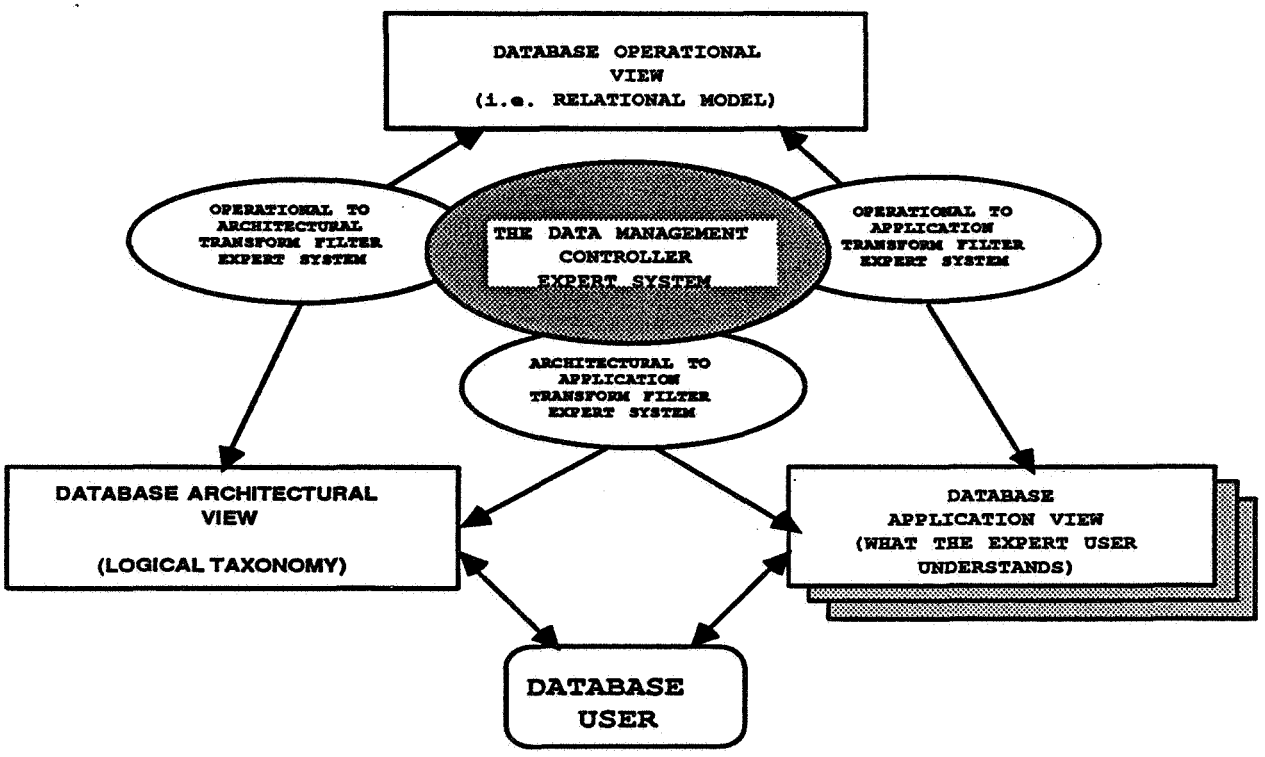


Figure 2. IUI Concept Model

In the context of the IUIS concept, the architectural view is defined to be a logical taxonomy of the database's organization that includes the relationship of the objects within the database that relate to the general knowledge and information that an expert user might have about the database. This view is organized to minimize processes that a user would apply when trying to locate a particular object, or the relationship between objects.

The architectural view is similar to some existing database design concepts but with two major differences. The first difference is that the objects and the language which are used to describe the objects in the architectural view are based on the user's language and understanding of the information of interest. The second difference is that this view may contain objects which do not exist as real data objects in the actual database. Such objects are known as virtual objects (objects that result from the clustering of real data objects).

The second view, the application view, is defined as being that view which is very domain specific and includes all the knowledge, facts and metaknowledge that would be known to the expert user working in a particular technical area for which the database has been designed to support. The application view will include three different types of knowledge and facts:

- Heuristic knowledge related to obtaining particular information for a specific application.
- Procedural knowledge that is used by an expert user to obtain information from the database for a specific application.
- Virtual objects that are known by the expert user as part of a specific application.

It is to be expected that there will be different user application views for different areas of database expertise. Because of this, it will be particularly important to formulate an IUIS design that will be capable of easily representing and modeling an "expert" user's knowledge of the meaning of the collected data. Such information will enable the expert system to include functional and semantic interrelationships among database entities, domain specific attributes, as well as content descriptions of objects.

The effect of the multiple view approach on the development of the IUI model is that, when it is integrated with a database there will be two basic schemas of the database: one which is a conceptual schema and one which is the database architectural schema (i.e., operational view).[6] Both are necessary for a complete description of the database.

3.2 Conceptual Graphs

The use of the conceptual graph in the intelligent user interface process is presented in Figure 3. To better understand the application of the conceptual graph to the database problem, the following

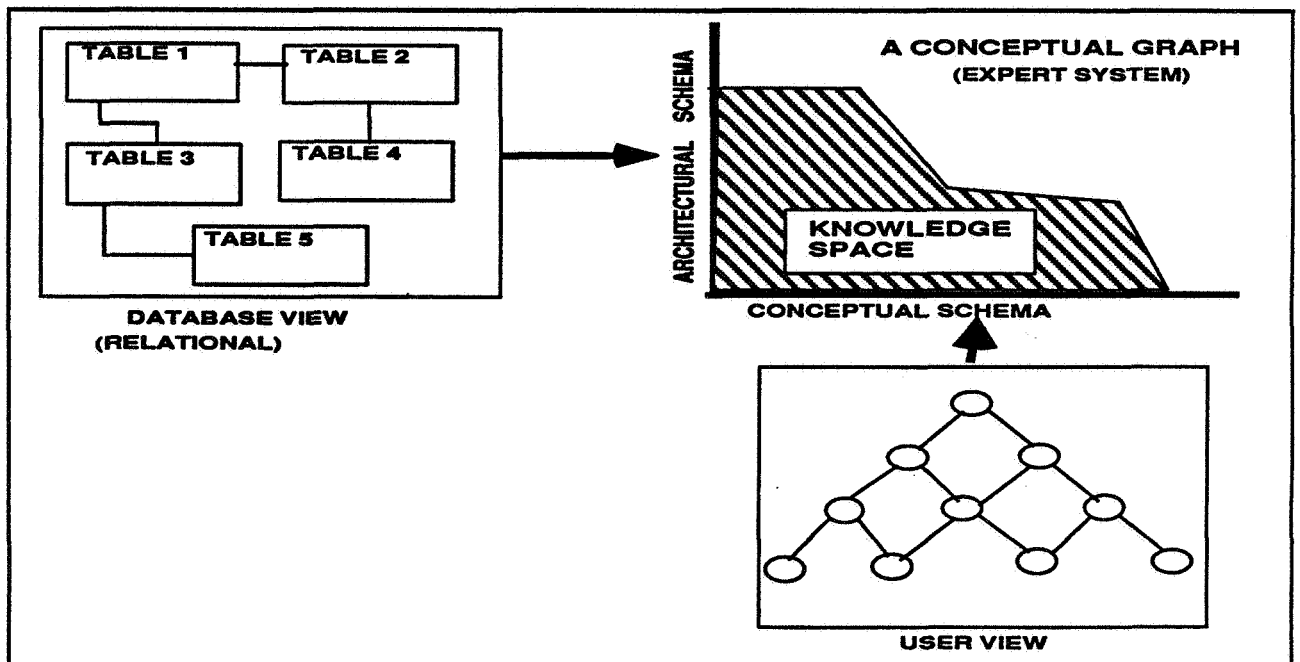


Figure 3. Representation of a Conceptual Graph

example is presented.

In the Crustal Dynamics Project's Data Information System (DIS) [7], a database has been developed at the NSSDC to support the storage of collected and analyzed crustal motion data from sensing stations located at many sites over the entire earth's surface. The users of the Crustal Dynamics database fall into two classes: engineers and scientists. The latter group is primarily interested in the database in the context of the study of the motion of the earth's crust and in earth rotation. The area of science related to crustal motion is commonly called plate tectonics. To get the necessary information related to plate motion, an expert database user must query the database about a specific set of stations, calculate the chord between them for a specific period of time and then determine the slope of the chord.

However, the objects, the chord and its calculated slope are not objects stored explicitly by the database. Therefore, the only way to determine a specific plate's motion is to identify the stations that are located on the specific plate and measure the motion of those stations with respect to stations on another plate (i.e., baseline measurements). Thus, there exist data objects needed in the identification and selection of the appropriate database information that do not exist in the actual database. It is left to the experienced database user (i.e., the expert) to understand and know how to calculate the chords between stations and their slope.

In the IUI concept, such missing information is taken into account so that, when information about a specific plate's motion is wanted, the IUIS knows how to calculate the motion based on the type of sensing system used (SLR or VLBI) [8]. To the casual database user, this intellectual transformation will be transparent. To the IUIS developer, this transformation is in fact the conceptual graph which facilitates the understanding of the meaning of objects in the context of the user's needs, and translates those objects into different objects with meaning to the database. In the formulation of the IUIS, one of the primary functions of the expert system is the creation of such conceptual graphs.

Consequently, a conceptual graph can be said to facilitate the understanding of the meaning of objects in the context of the user's needs, and translate those objects into different objects that have meaning to the database. In the formulation of the IDM concept, one of the primary functions of the expert system is being able to create conceptual graphs.

Using the conceptual graph concept to support a multiview representation of the database, we believe it is always possible to create at least one logical translation path between a user's view of the data (domain specific) and the database management system's view of the data using expert knowledge. This path is, in fact, an expert system and is illustrated in Figure 3.

3.3 The Application of Expert Systems to the IUIS

The basis for the design of the IUIS is the hypothesis that, an expert system can be developed that is able to reason about database information, both from the user's view and from the database's view, in such a manner as to intelligently identify and select required information with only limited guidance by the user. It is the job of the expert system to capture all the necessary knowledge about the structures and contents of the various views of the database and to support the translation between the views. Such an expert system will actually emulate an expert database user within the limited expertise domain.

The decision to use an expert system as the operational environment for IUIS software is based on the following considerations:

1. A database has a finite and rather limited knowledge space which makes the development of an expert system possible.

2. Certain important operations that are performed by an expert database user are based on heuristics (i.e., search strategies used to find information based on past experience).

3. Although a database has a large number of potential data sets available, the actual number of sets is usually limited to a small number, compared to the large number of data objects that could potentially be extracted from the database.

4. Many of the operations performed by the expert user are procedural in nature.

5. Expert system development tools, especially the advanced ones that employ frame representation schemas (i.e., LISP with flavor extensions), allow not only the implementation of multiple database views but also the construction of a logical taxonomy of objects in the database and the subsequent reasoning about such views and objects in a very robust fashion.

In the history of expert systems, the most successful development efforts have involved goal directed, backward chaining paradigms such as MYCIN[9] and PROSPECTOR[10]. In the case of the IUIS, where domain specific information is required, the same kinds of problem solving strategies which where successfully used in the aforementioned examples can also be applied to this development effort. Also, the ability to construct expert systems as conceptual graphs is based on the finite knowledge and syntax domains of a database.

The impact of the above assertions is that, it should be possible to develop intelligent processes for the IUIS using expert development tools for any development effort.

4.0 IUIS PROTOTYPE DEVELOPMENT

A phased approach was used to develop the first IUIS prototype so that each of the IUIS processes (database views and transform filters) could be completed, tested and evaluated independently before they were integrated into a system. The prototype system was named CRUDDES (CRUstal Dynamics Database Expert System) after the database it was built to support. The tasks used for the development of CRUDDES are as follows:

1. Database Selection Evaluation and Characterization.
2. Evaluation of the use of the database by expert users.
3. Formuation a prototype IUIS concept and develop near term system design,
4. Formulation and development of the application view.
5. Formulation and development of the architectural view.
6. Development of transform filters for the application and architectural views.
7. Interface the application and architectural views' transform filters to the NLQP.
8. Customize NLQP to support the application and architectural views.
9. Develop transform filters to support the translation between the application view and the architectural view.
10. Integrate the transform filters from all the views into a single knowledge base.

In the following sections the first eight steps of the development effort are discussed in detail.

4.2 Database Selection, Evaluation And Characterization

The database selected was one developed to support the Crustal Dynamics Project. The database which was implemented in ORACLE[11], a relational DBMS[12],[13], consisted of 104 tables that supported the management of data related to the motion of crustal plates and earth rotation.

The reason for selecting a relational DBMS over other types of database models is the simplicity of design from both the database view point and the user's view point. However, no matter which DBMS was selected, we felt that it was necessary that the system be supported by a natural language query interface. The reason for such a requirement is that formulating a desired database query in English is a much less complicated problem for an expert system than trying to reason in English and then translating to a query language like SQL.[14],[15]

The evaluation and characterization of the Crustal Dynamics Project database was performed by creating an Entity Relationship (E/R)[16] diagram, which is commonly used in initially designing a relational database. In addition to creating the E/R diagram, it was necessary to develop a logical taxonomy of the objects in the database using an approach that is commonly found in a thesaurus. This developed logical taxonomy, along with conversations with the expert users, helped us identify virtual objects.

The result of the evaluation of the database was the identification of three object classes in the architectural view of the database: experimental data, site data, and project management data. Because CRUDDES was a prototype, it was decided to deal only with the first two data classes, namely site data and experimental data, and to do so in two separate steps.

4.2 Evaluation of the Use of the Database by The Expert Users

This task of the development effort involved the

application of knowledge engineering techniques to determine how the database was being used by the application database experts. Two scientists that support the Crustal Dynamics Project were identified and contacted in regard to collecting information related to the actual use of the project's database. The scientists were interviewed on several occasions and, based on the interviews, three domain specific application views were identified:

- Scientific applications,
- Engineering applications,
- Project management applications.

4.3. IUIS Prototype Design Formulation

With the three application views identified, from the expert user, it was then possible to formulate a design for the initial prototype IUIS. The design, which is presented in Figure 4. consists of the scientific application view and the architectural view and is based on two sources of information: first, from the information gained during the task 3 effort and second from the interviews with the expert users in task 4. The basis for the design are the concepts discussed in Section 3.

4.4 Formulation and Development of the Application View

After the overall prototype design had been finalized, further interviews with the expert users were conducted to enable the development team to understand in detail how the database was utilized to support the development of the application view. The result of these interviews was a design concept that decomposed the scientific application view into two subviews: plate tectonics and earth rotation, and then each of these subviews into specific sets of information extraction goals. For the subview "plate tectonics," the information extraction goals were: whole plate motion, regional deformation, and plate stability; and for earth rotation: polar motion (precession) and rotational velocity. Each of the listed information extraction goals were further affected by the application of necessary modifiers such as year, site location and measurement method. However, even when the wherefore modifiers are considered the number of variables was found to be fairly limited. In addition, since most application-view queries usually included both procedural and heuristic knowledge, it appeared that the development of a scientific application view would be a fairly straightforward process that could implemented using expert system development tools, as envisioned.

During this portion of the development effort it became apparent that, more consideration would have to be given to the prototype design. Initially it was thought that, the application view could be developed separately from the architectural view; however, it became apparent that a single view lacked the minimum amount of necessary information that would be required to assist the nonexpert database user. The reason for this conclusion is that, nonexpert users understand little about the information in the database and consequently, might think there was, or should, be additional information extraction goals than had been provided. In other words, the IUIS has to be able to accommodate for the "I don't know what to select" response. This additional complication resulted in the development of a design that attempts to resolve what view best serves the user's needs, which became the context resolution process in the system. The resolution of what was the proper view for the user could only be assured by concurrently providing both the application and architectural views. The resultant design is presented in Figure 4.

The final step in this task was to determine all of the possible combinations of unique data extraction goals (the database objects with their wherefore clauses) required to get the desired information from the database for the scientific view.

After all the information extraction goals for the scientific domain (application view) had been identified and characterized, and a strategy for resolving view conflict resolution determined, it was then possible to develop a knowledge base using the expert system development tool. The goal of the first expert system was to support the automated reasoning required in the determination of desired data objects in the science application domain for a nonexpert database user. Such a determination was to be done in the context of the various information extraction goals available in the scientific application view or where applicable in the architectural view.

The development of the first expert system required about six weeks of effort and only dealt with the knowledge area involving plate tectonics. The system inference process was based exclusively on backward chaining and dealt with the various alternative selections available in the query formulation process by using generalized rule forms with variables. The expert system, in a planned structured

selection process, identified the various components that would make up a question and instantiated them into the variables where rules were used to construct the desired English question.

4.5 Formulation and Development of the Architectural View

The development of the architectural view of the database is based on the premise that it is necessary to provide to the user a generalized view of the database which is a logical organization of the objects that exist in the database and their relationship to each other. With the knowledge gained from this view, one should be able to identify information that is not available through the other views (application and functional) or an extension of an already available application formed query.

As discussed previously the primary reasons for having an architectural view is to support the "I don't know what I want" condition that one can expect a new database user to default to. The approach taken in designing the CRUDDES architectural view was based on the concept of object relationships that would be required if one were developing the view using a frame based expert system. The decision to use a frame type organization was the result of an initial development effort that organized the database into a tree structure.

The impact of selecting the frame over the tree approach was two fold; first, it provided the most complete relationship organization possible including virtual objects, and second, it makes the view reasonably easy to port to a new expert system that is being acquired as part of the FY86 project effort.

The organization for the architectural view consists of the database world as the highest level object with every other object being a component of that object. The next level of decomposition in the architectural view is site information and experiment information. At this level, objects are clustered by their relationship to the ordering. The beauty of this type of view structure is that when inheritance is considered, objects will be able to receive new attributes by the very fact that features have been added to objects at lower levels. Thus, inheritance allows a system to respond as if it were learning automatically.

The development of transform filters that would function as the interface between the architectural and the operational views did not require a great deal of effort. This is because, only a portion of the architectural view was implemented in the first phase of the CRUDDES development effort. In addition, the approach for the development of transform filters between the architectural view and the operational view were very similar to those developed from the application view to the operational view, with the result being that, once the object and cluster object relationship was determined, it was only a matter of creating the necessary rule base to provide a functioning transform filter capability.

4.6 Transform Filter Development to the Operational View

In this task identified objects were translated from the application and architectural views to objects that exist in the operational view and then translated into a syntax that could be understood by the database (SQL).

The first translation process is one of the most important components of the IUI concept because, this is where the expert database user is necessary in order to translate from what the user understands and needs to what the database understands. This conclusion is based on the hypothesis that, there exists in the application and architectural views many objects that do not exist singularly in the functional view. These consist of the clustering or grouping of several objects that exist in the functional view. It is this expertise, along with the procedure knowledge, that is required to form the proper query that signifies the expert database user. In the prototyping of CRUDDES, such knowledge was identified and captured in the expert system.

4.7 Interfacing the Application and Architectural Views to the NLQP

The important contribution of this task effort was the appreciation for the robustness of the English language in being able to formulate, in a very compact way, an expression that communicated what information is desired. It would appear that, when dealing with or reasoning about an object (or information about the object) that exist in a domain specific space, it is best to formulate the questions in the context of that domain using the syntax applicable to the domain. The result of this conclusion for the IUIS design is that, it is best to stay in an object's conceptual domain until the entire query expression has been formed and only then translate the question to a database query using its language.

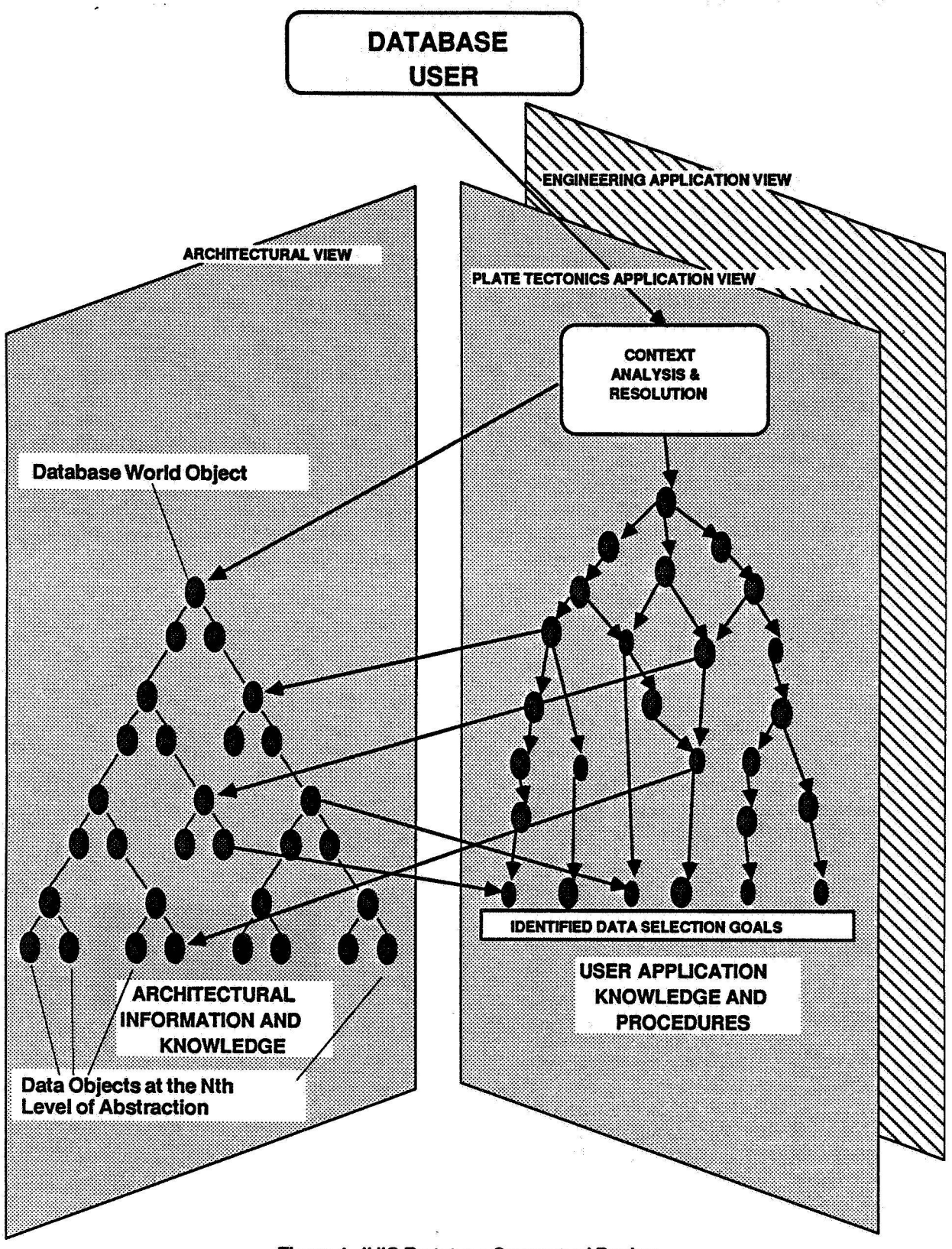


Figure 4. IUIS Prototype Conceptual Design

The customization of the natural language query processor, THEMIS, was a very simple and straight forward process because the system is a functioning commerical product that has been built to support rather large and complex relational database systems. In addition, since CRUDDES is a prototype system that deals with only a portion of the selected database, the number of English queries it is capable of generating is rather small. Consequently the dictionary of lexicons that had to be added was also rather small (due to the limited number of information goals coded into the knowledge base).

However, because the IUIS had to be able to resolve ambiguity in the user to system communication, special attention was paid to the formulation of English queries on the CRUDDES side and their response on the database side.

As the CRUDDES system expands, the ability of the NLQP to deal with imprecise queries may become a more important issue and consequently, how the system will deal with impreciseness is unclear. However, in the long term, it would appear that, any IUI system must be able to understand the user based on more than syntax. Presently, machine understanding at the context or semantic level of abstraction is quite difficult if not impossible to implement and it may turn out that the most difficult IUIS development problems will involve English understanding.

5.0 SYSTEM PERFORMANCE & PROJECT DIRECTION

The last two tasks in the prototyping of the demonstration system are presently under development. However, we feel that their completion will not have a significant effect on the results of this effort. The reason for this assertion is that their inclusion into the system can only improve its performance and not change its ability to function as a substitute for the expert database user, which it already does to some degree.

The system's ability to support inexperienced database users and scientists was tested over a short period of time with people of varying degrees of database skills. The results were quite pleasing in that in most cases, except when the user was very knowledgeable with the database and its operation, the system was able to significantly reduce the amount of time that would be necessary to obtain a specific piece of desired information. In fact, except in a very few instances, the SQL query resulting from one user's interaction was clearly beyond his technical ability to generate it in the normal fashion. Thus it appears that an IUI system allows even the most inexperienced user to function like a database expert, armed only with his knowledge about the technical domain of interest.

Given the above we believe that CRUDDES has demonstrated that it is possible to develop and implement intelligent value-added services to database systems, that are capable of interacting with, and fully supporting the nonexpert database user, in specific domains of interest. In addition, we believe that such a system will be able to function at a level of performance and understanding, that can only be rivaled by the experienced expert user, with working knowledge and experience with information with the information contained in the database.

Thus it now appears that we have the technical basis to begin the design and development of the next generation IUI system, which when married with more robust software and hardware should be able to function very effectively in many NASA operational database environments where research scientists need better data management support.

REFERENCES

[1] Shi-Kuo Chang, Jyh-Sheng Ke, "Translation of Fuzzy Queries for Relational Database Systems", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, No. 3, July 1979.

[2] ibid.

[3] Campbell, W. J., Roelofs, L. H., "Artificial Intelligence Applications Concepts for the Remote Sensing and Earth Science Community" *,Proceedings of the IX Pecora Conference,* IEEE Publication Catalog No. 84CH2079-2, October 2 - 4 1984, pp. 232.

[4] Sowa, J. F. "Conceptual Graphs For A Data Base Interface," *IBM J. Res Develop.,* July 1976.

[5] Date, C. J. "An Introduction to Database Systems", Third Edition, Addison-Wesley Publishing Co., Reading, MA, 1981, pp. 13 - 17.

[6] op. cit., Campbell, W. J., Roelofs, L. H., "Artificial Intelligence Applications Concepts for the Remote Sensing and Earth Science Community"

[7] Noll, C. E., "The Development of Selected Database Applications for the Crustal Dynamics Data Information System, NASA Technical Memorandum 83886, December 16, 1981.

[8] ibid.

[9] Buchanan, B. G., Shortliffe, E. H., "Rule-Based Expert Systems, The MYCIN Experiments of the Stanford Heuristic Programming Project," Addison Wesley Publishing Co., Reading, MA, 1984.

[10] Duda, R. O., Gaschnig, J. G., Hart, P. E., "Model design in the PROSPECTOR consultant system for mineral exploration," In. D. Machine, ed., Expert system in the micro-electronic age. Edinburgh; Edinburgh University Press, pp.153-167, 1979

[11] "SQL/UFI Reference Manual, Version 4.0," ORACLE Corp., June 1984, D002-0684-840521.

[12] Codd, E. F., A Relational Model of Data for Large Shared Data Banks," *Commun. Ass. Comput.* Mach., Vol. 13, June 1970.

[13] op. cit. Date, C. J. "An Introduction to Database Systems", pp. 73 - 81.

[14] Denny, G. H. "An Introduction to SQL: A structured query language,", IBM Tech. Rep. RA93(28099), May 1977.

[15] Chamberlin, D. D., Boyce, R. F., SEQUEL: A structured English query language," *ACM-SIGMOD Workshop on Data Description, Access, and Control,* May 1974.

[16] Chen, P. P. S., "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Trans. Database Syst.,* Vol. 1, pp. 9-36, 1976.

S2-63

233132
8P.

AI Goes FORTH†

W. B. Dress
Oak Ridge National Laboratory
PO Box X, Oak Ridge, Tennessee 37831

Abstract

The Forth language is presented as a vehicle for developing applications constrained by real-time considerations and size of hardware system. A specific example of rewritting OPS5 in a multitasking version of Forth shows that such applications can be extended to the realm of problems requiring a real-time artificial intelligence approach. The goal of high-speed, intelligent software operating in a restricted hardware environment is thus attainable in a cost-effective manner.

Introduction

Artificial Intelligence is a set of methodologies and a philosophy. In this paper we will be concerned with the methodologies. AI will be taken as the art and science of making computers more useful to humans.[1] The methods of AI include those techniques (e.g., recursion), algorithms (e.g., search schemes), and paradigms (e.g., declarative languages) that promise to further the goal of usefulness in specific applications. A particular language (e.g., LISP) is merely a means to an end--it is neither the goal nor the embodiment of AI philosophy.

In moving from the ivory tower to the worlds of industry and applied science, AI has found a need for another property--that of performance. In the past performance in AI has meant running LISP a little faster, but since AI is not equivalent to LISP, this is a narrow view of performance. This paper will hint at a broader definition of performance--an application oriented definition--and examine performance-related features such as code volume, execution speed, and programmer productivity. Ideally, AI has been a software endeavor--ideas and examples have been independent of any particular hardware implementation. When performance becomes a necessity, this ideal view will have to change. For example, parallel hardware architectures are necessarily dictating software architectures. A danger is that this approach can lead to some awkward software, just as hardware implementations of software ideas (e.g., procedure calls) has led to some very awkward hardware. Similarly, the new super LISP chips may codify the dubious identification of AI as LISP--another example of hardware determination of software methodology.

†Research performed at Oak Ridge National Laboratory, operated by Martin Marietta Energy Systems, Inc., for the U.S. Department of Energy under Contract No. DE-AC05-84OR21400.

Another philosophy, one of an integrated approach to hardware and software use, will provide the framework for exploring AI applications requiring the added dimension of high performance. That philosophy, currently evolving in practice and definition, is embodied in the computer language Forth.

FORTH as a Vehicle for AI Applications

Forth has long been recognized (perhaps surreptitously) as a tool supplying performance and economy, particularly for dedicated control applications in which speed of response and efficient use of hardware resources are crucial. However, recognition of its inherent expressivness as a vehicle for high-level application languages is fairly recent;[2] it is this property that we will examine, both from an historical perspective and through a specific high-level language example.

What is FORTH?

Like other modern high-level languages, Forth is recursive in nature (i.e., it is written in Forth and one program can be data for another program). Forth gives the user mastery over the underlying hardware, be it memory, registers, mass storage, or communications ports.

A fundamental difference from other computer languages is Forth's extensibility. Most modern languages are extensible in the form of user-defined subroutines, functions, and macros. Extensibility in Forth radical--the language itself can be extended at the whim of the programmer. The user is able to redefine keywords both cosmetically and functionally, take control of the interpreter and modify it for specific needs such as parsing or stream processing, and define application-specific data structures of any desired complexity including object-oriented and access-oriented structures.

Some of the multiparadigm languages becoming available[3] have all of the features deemed necessary by their authors for AI applications and, as such, are undergoing continuous development and redefinition with no definitive version in sight; new super features must always await the next release of the operating system. Alternatively, Forth programmers must reinvent the wheel each time a special construct is needed, and this is Forth's weakness and strength. (Remember why we are examining this language: for its use as an application-specific language for attaining high performance.) If a language extension is not needed, it doesn't get in the way of the programmer and the application. Forth considers data structures and language extensions the way C considers functions--they belong in a library where they can be used or ignored as desired, thus having no impact on the compiler and on ultimate system performance.

Forth is a weakly typed language, words (subroutines) and data structures have no pre-imposed interpertation. If the programmer wishes to add an integer to the address of a subroutine, there is nothing to prevent it. A more general view is that everything in Forth is typed by its behavior--that is, each word contains a reference to a particular system- or user-defined behavior specified at the time of definition. Thus it is possible to execute a data structure if it is so defined. An example would be an array containing the bitmap of a graphics object to be displayed. Normally, the array is passed as an

argument to a subroutine which then maps the information onto a display. In Forth, merely referring to the array by name invokes its behavior (maps the object it contains onto a display screen). The advantage is conceptual as well as functional. This technique leads to compact and efficient code where the behavior of the object is specified once in the vocabulary (as is a subroutine in an on-line library) and is never explicitly mentioned to invoke its behavior (this is the defining word[4] construct).

Applications development benefits from this "on-line subroutine library," which is developed in an interactive manner *á la* rapid-prototyping methodology. As each module is conceived, written, and verified, it is added to the set of immediately available (and previously verified) modules in an interactive and incremental manner. There are two immediate benefits from this approach: one for the programmer who is no longer bound to the edit-compile-link-debug cycle for each incremental change, and the other for the application which shows a reduction in code volume and concomitant savings in memory use and power consumption for the final system.

Once an application has been developed and globally tested, time-critical areas may be identified and rewritten in assembler code without the need for rewriting or even recompiling the application. Another feature allowed in many systems is writing assembler code inline with high-level code and vice versa. Time-critical loops can be encoded for efficiency without leaving a high-level definition, and assembly code can refer to and call high-level constructs. Thus Forth retains the nature of a high-level language while allowing the performance usually associated with C.

FORTH and AI

Recently, some Forth programmers have been solving problems traditionally thought of as AI applications. Some of the solutions were arrived at independently of AI methodologies, and some were conscious applications of AI methodologies. Some of the many applications and studies carried out in Forth, but which have a direct bearing on the problems of AI, are mentioned below.

Several upcoming papers are particularly worthy of mention as having a freshness in their approach and a timeliness in their subject. One paper[5] suggests using the dictionary and vocabulary structure as key elements in a natural language parser while creating the demons useful for disambiguation via defining words. Another paper[5] shows how to create an inference engine that considers any specified word (routine) to be a production-system rule as long as it returns a truth value when invoked. This engine ran a rule-based Towers of Hanoi in 9.25 s for 10 towers (32-bit Forth on a MC68000 at 10 MHz). A straight Forth or LISP implementation of this benchmark is much faster, while an OPS5 running on a high-end LISP machine took more than 140 s.

The topic of this year's Rochester Forth Conference in June is real-time artificial intelligence, systems, and applications. Among the papers to be presented[6] are a Forth-based Prolog for real-time expert systems, a real-time interpretation of shuttle data, and applications of AI to process control.

It is apparant that AI and Forth have much in common in the area of real-time applications and high performance. The following discussion shows how Forth can be used as a high-level assembly language for writing application-specific languages--even one as rich as the widely used OPS5.

Making OPS5 Real Time

OPS5 is a syntactically and conceptually simple expert-system language developed at Carnegie-Mellon University by Charles L. Forgy[7] in the early 1980s and is a rule-based (production system) language. Programming conceptually consists of encoding procedures, declarations, and heuristics in forms resembling an IF... THEN... statement. The conditional clauses making up the IF or left-hand side (LHS) portion of a rule are known as condition elements and are patterns of desired data to be matched by actual data instances. When the set of condition elements in a rule is satisfied by corresponding instances in the data world (working memory) of the OPS system, that rule is satisfied and is free to "fire," carrying out the actions specified by the collection of THEN clauses on the right-hand side (RHS) of the rule. Since each rule is an independent module, the complexity of OPS5 is considerably less than that of Pascal or an extended Basic. The organization of a program into discrete rules that are logically and functionally independent provides a modularity necessary for the successful maintenance of a complex software system. A well-written OPS5 program is more readable and less complicated than a comparable Fortran or Basic version.

A major characteristic distinguishing OPS5 from other inference engines is the Rete algorithm,[8] which is responsible for the efficiency of the pattern-matching cycle. This algorithm maintains a set of pointers from actions to condition elements. Only those condition elements that change when a working memory element is added or removed need be notified. This obviates a need for rematching each LHS pattern to all of working memory for each cycle.

OPS5 is primarily a data-driven or forward chaining system. Evolution of the expert system toward its goal is determined by the current state of the data set. Goal-directed controlling strategies can be imposed easily by a set of OPS5 rules. A complex control problem usually requires understanding large amounts of data, so an OPS-like language may be essential for the sensor fusion problems gaining importance in space and military applications.

In summary, an OPS5 program is a set of rules consisting of patterns to be matched and actions to be effected. If all of the LHS patterns of a particular rule have consistent instances in working memory, the rule is free to fire, executing the actions specified in the RHS. The actions typically make and remove working memory elements, thus affecting the satisfied status of other rules in a ripple effect.

Rewriting OPS5 for Real-Time Applications

We now look at Forth as a rich and powerful assembly language for rewriting the expert systems language OPS5 in a real-time, multitasking environment. A radical reworking of the OPS5 infrastructure is required to retain the syntax and basic algorithms while allowing multitasking and the immediate inclusion of real-time data for consideration by the rule set. An essential requirement is that access to the system's underlying hardware be kept as close as possible to the top-level OPS rules. Such a feature is necessary for successful real-time operation. The language chosen for this project was a multitasking version optimized for the Motorola 68000 microprocessor and

the Hewlett-Packard Series 200 desktop computer.[9] The only portions not portable to other computer systems are those parts dealing with the specifics of the Series 200 hardware (timers, operator interface, and file system).

The task of rewriting OPS5 consists of implementing the syntactical rules of OPS5, writing code to build the network (Rete algorithm implementation), creating a memory manager for the working memory data, and building the necessary user interface.

Syntax and Parser

A standard method of enforcing a syntax is to build a position-sensitive parser that identifies the reserved words of the language. The parser must keep track of the "parts of speech" and distinguish among various uses of the same symbol. Since the syntax of OPS5 is relatively simple, a straightforward state implementation sufficed.

The state parser consists of a special vocabulary for accepting symbols from the input stream and activating code sequences for compiling the corresponding data structures (class objects and rules). For example, **STATE** changes the parser state and pushes the new state onto a "lifo" stack; **PRIOR.STATE** pops the current state off the stack and changes the parser to the one previously specified. Also, a number of state-specific words (such as **RULE.STATE**, **WRITE.STATE**, etc.) actually do the work of changing the state by calling the routines that build the data structures. Each parser state is responsible for a different portion of the rule being compiled, so changing data structures or adding new features is a simple matter of modifying the appropriate parser module.

The data structures built by the parser were chosen for speed and flexibility, not generality. Both Forth and OPS5 are weakly typed languages, so optimizing data structures for special cases is a simple matter of definition. This feature resulted in considerable savings in memory usage and processor cycles, allowing more specialized data structures than in standard OPS. However, the number of conditions per rule and the number of attribute slots per condition element have been limited to 32. This is not as restrictive as it might appear; the typical OPS program has an average of six conditions per rule and five attributes per condition.[10]

Managing Memory

A special memory manager[11] was written to handle the multitude of dynamic lists required for the internal bookkeeping of the OPS5 inference engine as well as the working memory and real-time data elements. In keeping within the constraints imposed by real-time operation, long and unpredictible memory-cleanup pauses are avoided by requiring that the memory pool be kept in its compacted form at all times. Thus when a data structure is no longer needed, the memory space it occupied is returned to the pool of available memory in a manner avoiding fragmentation and unusable gaps. This method of synchronous garbage collection overcomes the problems inherent in the usual method, which can be fatal for a real-time application.

Running the System

The standard OPS recognize-act cycle has been modified to allow for a rule-priority scheme and multitasking. Thus, the rule with the highest priority is allowed to fire first; if no rules have priority, then the usual OPS5 conflict-resolution strategies are employed. If no rule is able to fire, the system pauses before re-examining the conflict set. Although this last step makes no sense in the usual context of OPS5, in which firing rules or console input are the only sources of new working memory elements, it is essential in a real-time version. The pause lasts a few microseconds (the time it takes the task dispatcher to examine the task queue) and allows any data created by external events via interrupt routines to be entered into working memory and linked to the appropriate condition elements so that the next conflict-resolution cycle can consider any rules responding to the new data.

FORTH and Multitasking

The main system task is running the recognize-act cycle; no other tasks have precedence (except hardware interrupts, of course). Secondary (and hence software-interruptable) tasks include communication to the operator and data collection. The recognize-act cycle cannot be arbitrarily interrupted for adding new memory elements or the system state may become indeterminate. Data tasks are allowed to update working memory at the beginning of each recognize-act cycle, thus acknowledging any external events that may have been captured during the previous cycle.

Most versions of OPS5 allow external calls to routines written in other languages. Since the goal of this work is to provide a language useful in real-world applications, external calls are to Forth words (which the user will have built to handle details of device control, etc.). Any words the user has written for such tasks as instrument control, data collection, or graphics display are callable from the RHS of any expert system rule. Thus, the user has the power of an expert system language while retaining easy access to the underlying hardware.

External Asynchronous Data

The extensible nature of Forth was used in creating a special vocabulary for opening pathways into an expert system for asynchronous data.[12] When the user writes the software designed to drive and control the particular devices making up the application, this specialized vocabulary is available for specifying those data elements and the format for presentation to the expert system. Merely specifying the data elements in a vector format (resembling a condition element) creates the necessary pointers into the expert system's set of class objects. When the data forms are instantiated by actual data, a working memory element is created and becomes accessible (via the Rete network) to all rules potentially needing the information.

Results

A High-Performance PC Expert System Tool

One result of this work has been to make available a high-performance OPS5 running on a desktop computer such as the Hewlett-Packard 200 and 300 Series machines and a promised port to those personal computers based on the MC68000 microprocessor (e.g., Apple Macintosh, Atari 1040 ST, and Commodore Amiga).

The small size of the kernel (operating system) and the efficient use of memory inherent in well-written code imply small overhead for any expert system written in the current version of OPS5. A 1-Mbyte system has room for an extended operating system and the complete interactive and incremental compiling OPS5, with room left over for 2000 rules, 5000 working memory elements, interrupt handlers, data display routines, and other user routines. The composite system is memory efficient and time efficient as well.

In the present version only the memory-management (garbage collection) and pattern-matching sections are written in assembly code; other time-consuming sections such as conflict resolution and the RHS actions remain in high-level Forth. Even so, its performance is considerably better than that of the LISP-based OPS5 running on a VAX 11/780, and it approaches the performance of the new TI Explorer on some OPS5 programs.

A Real-Time Expert System Tool

Perhaps more important for members of this conference than being able to write and execute an expert system using canned knowledge is the ability to write a system that is event driven. The knowledge base becomes a collection of procedures and algorithims with accompaning heuristics based on pattern recognition for selecting which algorithm to invoke and which actions to take. An OPS5 executing on a MC68020-based system with a math coprocessor benchmarks between two and four times faster than the one reported above, making the execution of a Forth-based OPS5 equivalent in performance to a LISP-based OPS5 executing on the promised LISP chips.

The next step in the search for applied AI systems performance has already begun. A novel microprocessor[13] employing RISC techniques in a non-von Neumann setting has a version of high-level Forth as its machine language. The current chip is available in a CMOS gate array with a clock frequency of 6 MHz. This is enough to outperform any current microprocessors, and many mainframes as well. The next version could be a GaAs array[14] running one to two orders of magnitude faster. The result should be an expert-sytem engine executing at a rate of 1,000 to 10,000 OPS-like rule firings per second. Adding a modest degree of parallelism[15] could further boost this by another factor of 10 or more.

Summary

Applications- and performance-oriented programmers have been using Forth since its inception. Recently the idea of Forth as a language for writing other languages, principally applications-specific languages such as for robotics, has taken hold. An extension of these ideas was presented here wherein Forth became the assembly language and the operating environment of the OPS5 expert-systems language, providing improved performance and a needed extensibility into the domain of real-time event-driven expert systems.

References

1. Patrick Henry Winston, *Artificial Intelligence*, 2nd Edition, Addison-Wesley, Reading, Mass., pp. 1-3, 1984.
2. Leo Brodie, *Thinking FORTH,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1984.
3. See the January 1986 issue of *IEEE Software,* IEEE Computer Society, Vol. 3, No. 1, January 1986.
4. Leo Brodie, *Starting FORTH,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981.
5. *The Journal of FORTH Application and Research*, to be published.
6. Lawrence P. Forsley, private communication.
7. C. L. Forgy, "OPS5 User's Manual," Technical Report, Carnegie-Mellon University, Department of Computer Science, 1981.
8. Charles L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence*, Vol. 19, No. 1, 1982.
9. Creative Solutions, Inc., *Multi-FORTH Version 2.00 User's Manual*, Rockville, Md., 1984.
10. A. Gupta and C. L. Forgy, "Measurements on Production Systems," Technical Report, Carnegie-Mellon University, Department of Computer Science, 1983.
11. W. B. Dress, "A FORTH Implementation of the Heap Data Structure," Proceedings, 1985 Rochester Forth Conference, *The Journal of FORTH Application and Research*, Lawrence P. Forsley, Ed., Vol. 3, No. 2, pp. 135-38, 1985.
12. W. B. Dress, "Communicating Asynchronous External Data to an Expert System," *Proceedings, Eighteenth Southeastern Symposion on System Theory,* IEEE Computer Society, April 7-8, 1986, pp. 294-96.
13. Earle Jennings, "The Novix NC4000 Project," *Computer Language*, p. 37, October 1985.
14. Velijko Milutinovic, David Fura, and Walter Helbig, "An Introduction to GaAs Microprocessor Architecture for VLSI," *Computer,* IEEE Computer Society, p. 30, March 1986.
15. Charles Forgy, Anoop Gupta, Allen Newell, and Robert Wedig, "Initial Assessment of Architectures for Production Systems," *Proceedings, National Conference on Artificial Intelligence*, pp. 116-20, 1984.

53-81
233133
15P.

THE EXPERT PROJECT MANAGEMENT SYSTEM (EPMS)

by

Barry G. Silverman
Coty Diakite

IntelliTek, Inc.
Silver Spring, MD 20910

and

The George Washington University
Institute for Artificial Intelligence
Washington, D. C. 20052

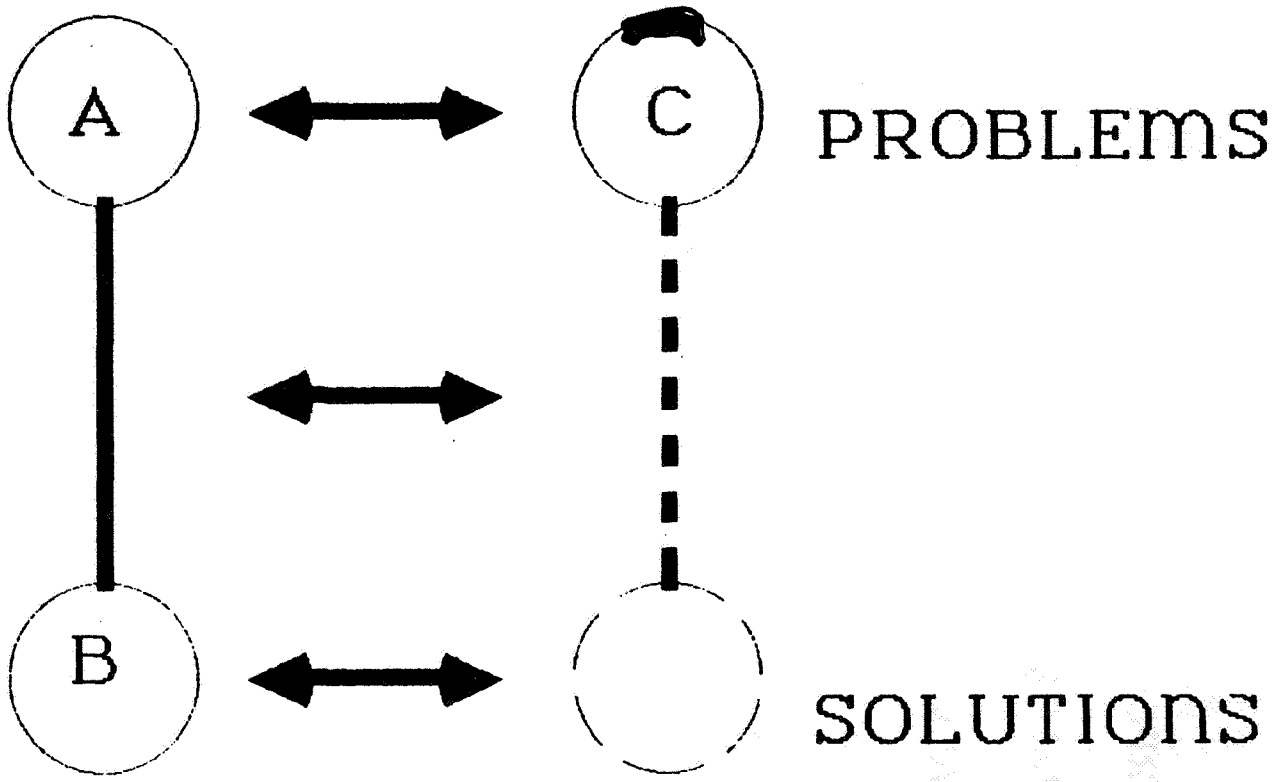
TECHNICAL ABSTRACT

Successful project managers (PMs) have been shown to rely on "intuition," experience, and analogical reasoning heuristics. For new PMs to be trained and experienced PMs to avoid repeating others' mistakes, it is necessary to make the knowledge and heuristics of successful PMs more widely available. The preparers have evolved a model of PM thought processes over the last decade that is now ready to be implemented as a generic PM aid. This aid consists of a series of "specialist" expert systems (CRITIC, LIBRARIAN, IDEA MAN, CRAFTSMAN, and WRITER) that communicate with each other via a "blackboard" architecture. The various specialist expert systems are driven to support PM training and problem solving since any "answers" they pass to the blackboard are subjected to conflict identification (AGENDA FORMULATOR) and GOAL SETTER inference engines.

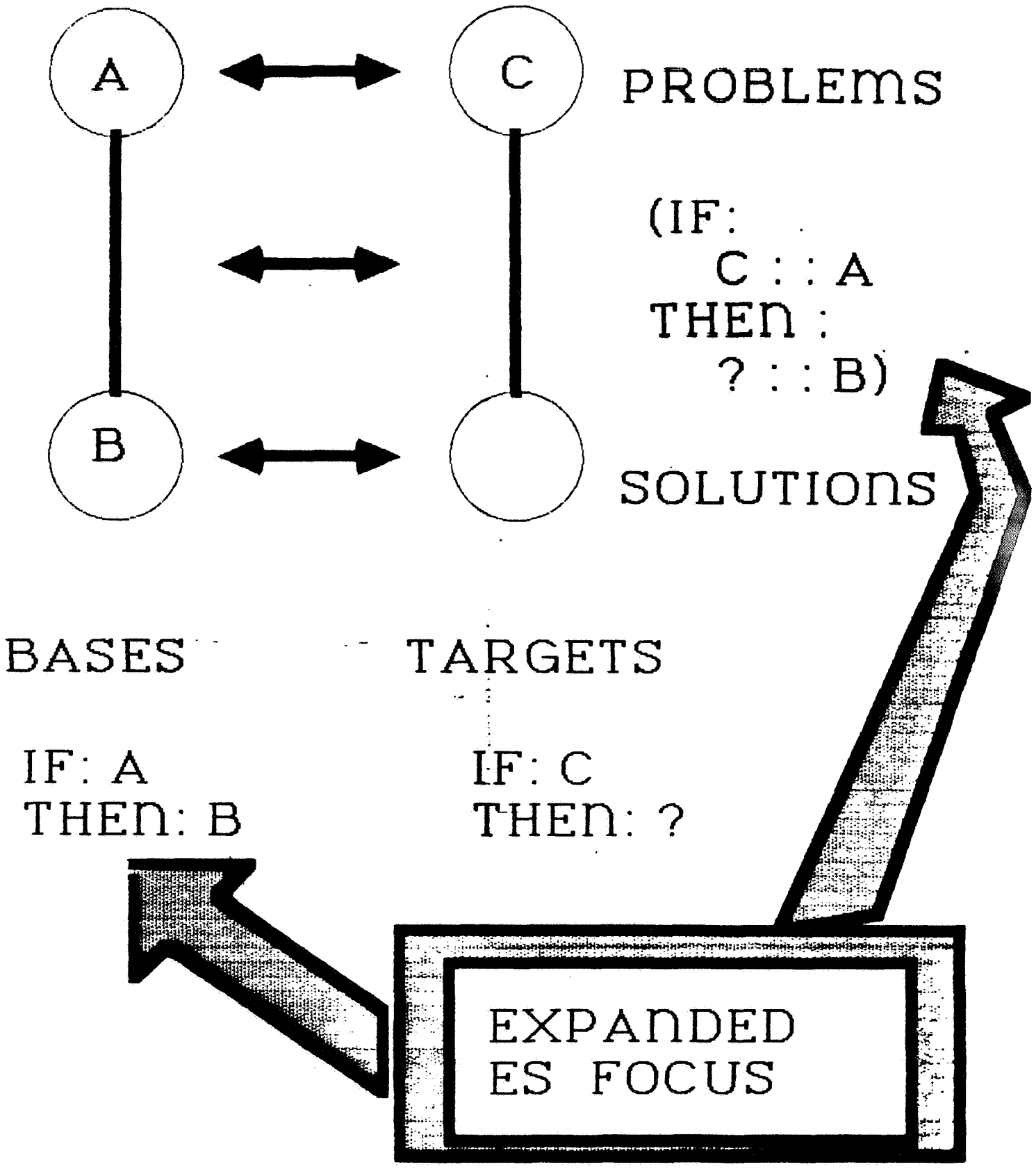
OUTLINE

- PURPOSE & OVERVIEW
- ANALOGY DEFINITIONS
- BLACKBOARD EXPERT SYSTEM TOOL (ARIEL)
- EPMS
  - OVERVIEW & CURRENT STATUS
  - USER INTERFACE SUBSYSTEM
  - KNOWLEDGE BASE
  - INFERENCE ENGINE
- EXAMPLE SESSIONS
- NEXT STEPS

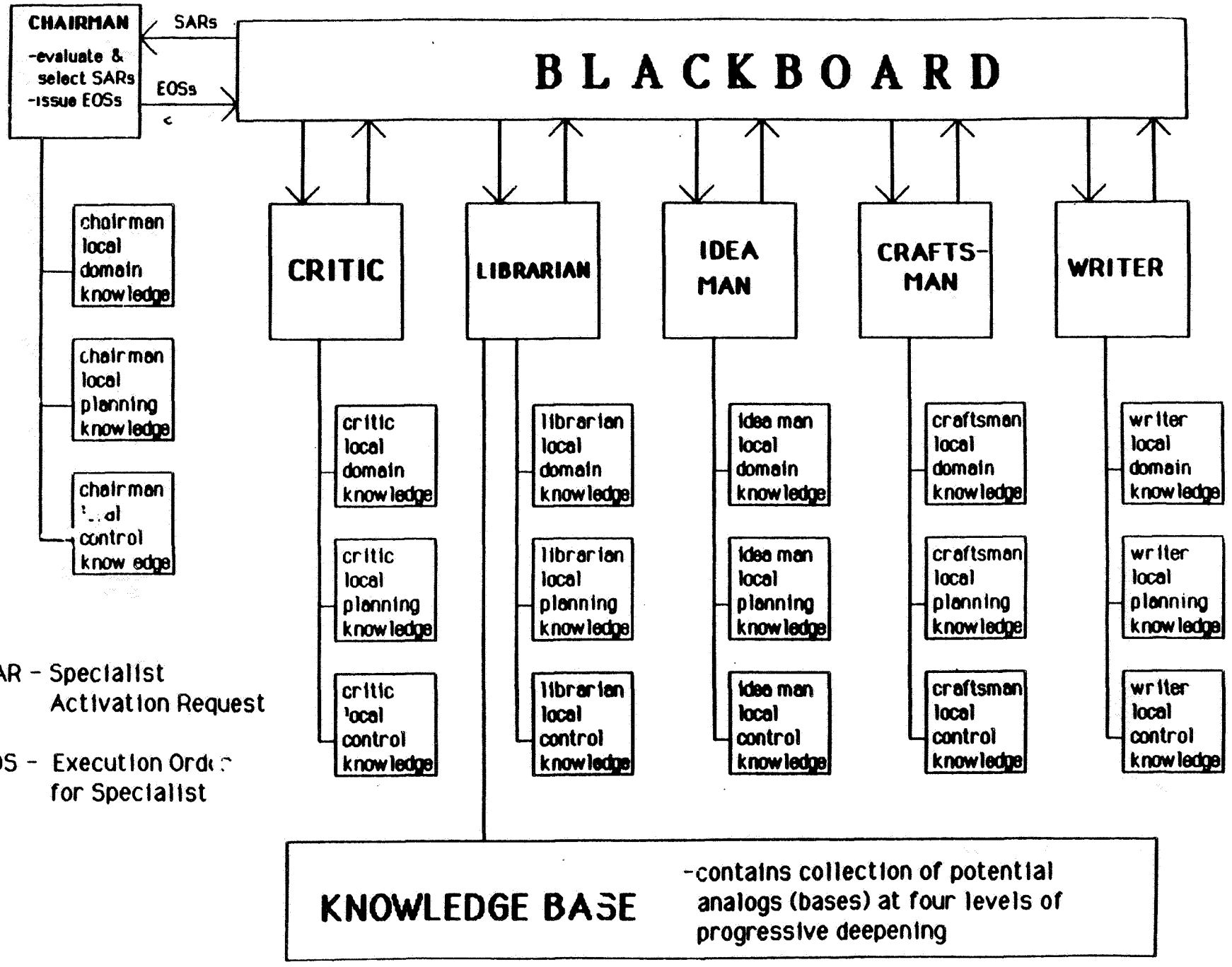
A : B : : C : (X ,Y,Z ?)



BASES TARGET


A
C
PROBLEMS
(IF:
C : : A
THEN :
? : : B)
B
SOLUTIONS
BASES
TARGETS
IF: A
THEN: B
IF: C
THEN: ?
EXPANDED
ES FOCUS

ARIEL ARCHITECTURE



SAR - Specialist Activation Request

EOS - Execution Order for Specialist

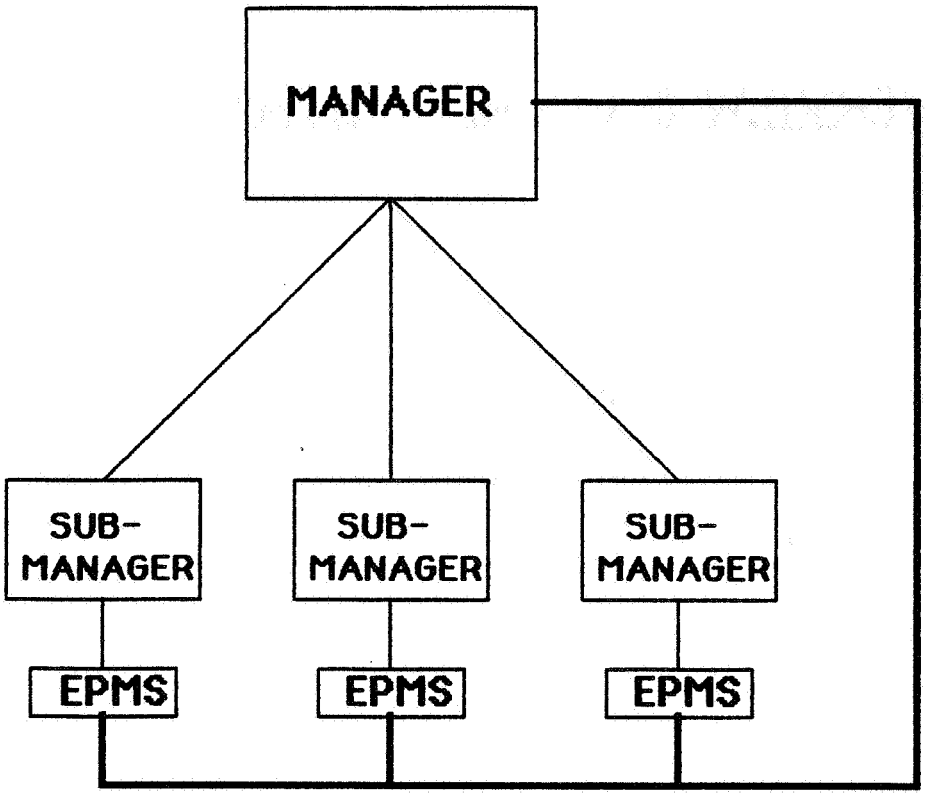
EPMS OVERVIEW & CURRENT STATUS

- o USER INTERFACE SUBSYSTEM
  - EXECUTIVE CONTROL PANEL
  - VISUAL ENGINEERING MODULE
    - Lattices
    - Tables
    - Grapher Package
    - Menus
    - Maps*
    - Iconic Scene Depictions*
    - Organization Charts*
    - Flow Diagrams*

- o KNOWLEDGE BASE
  - ATTRIBUTE LATTICES
  - ANALOGS
    - LANDSAT.D
    - NIMBUS.G
    - SPACE TELESCOPE

- o INFERENCE ENGINE
  - SIMILARITY METRIC
  - MEANS-ENDS ANALYSIS
  - SAR GENERATOR
  - INSPECT FEATURE

* FUTURE ENHANCEMENTS

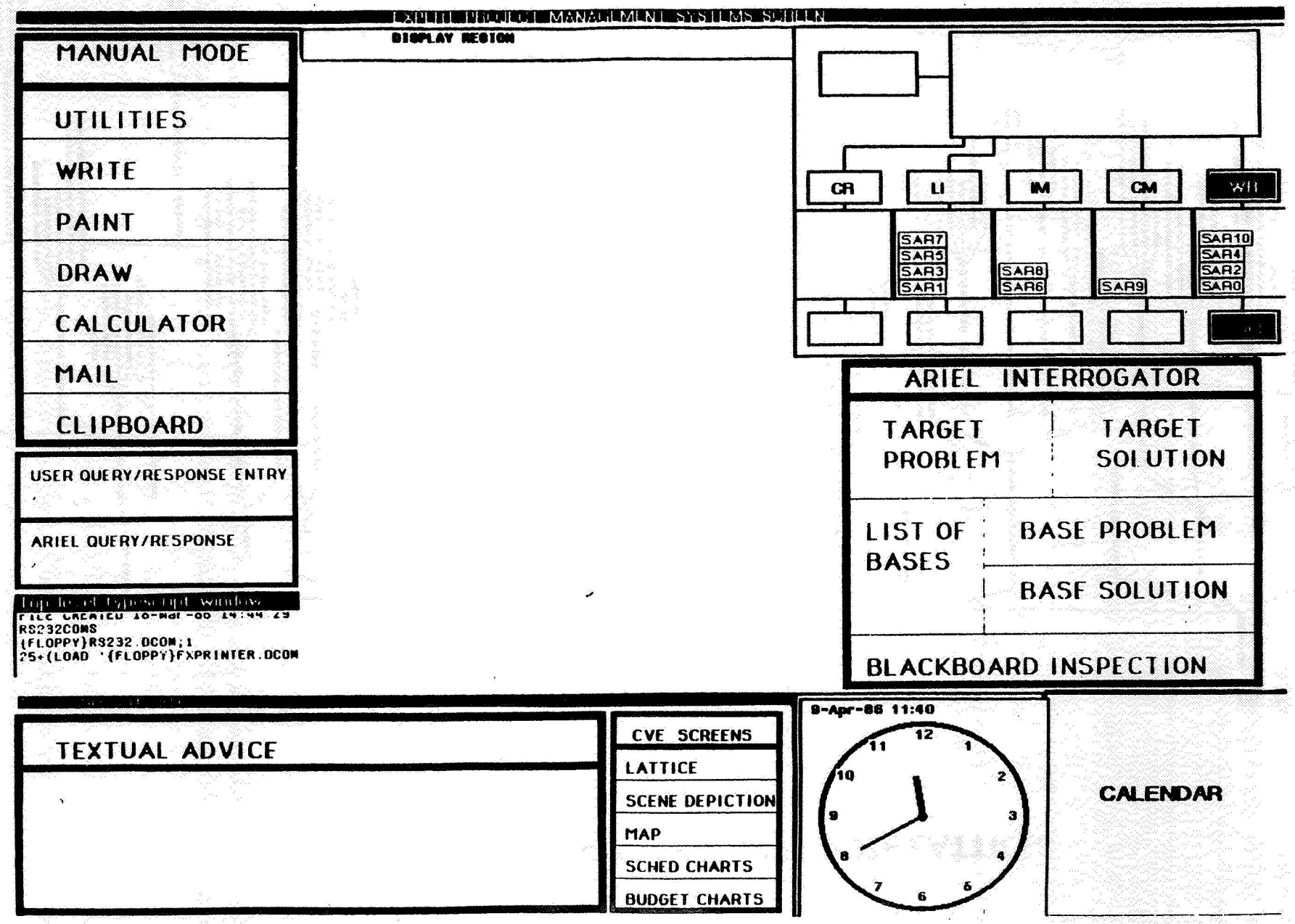


EPMS
ADMINISTRATOR

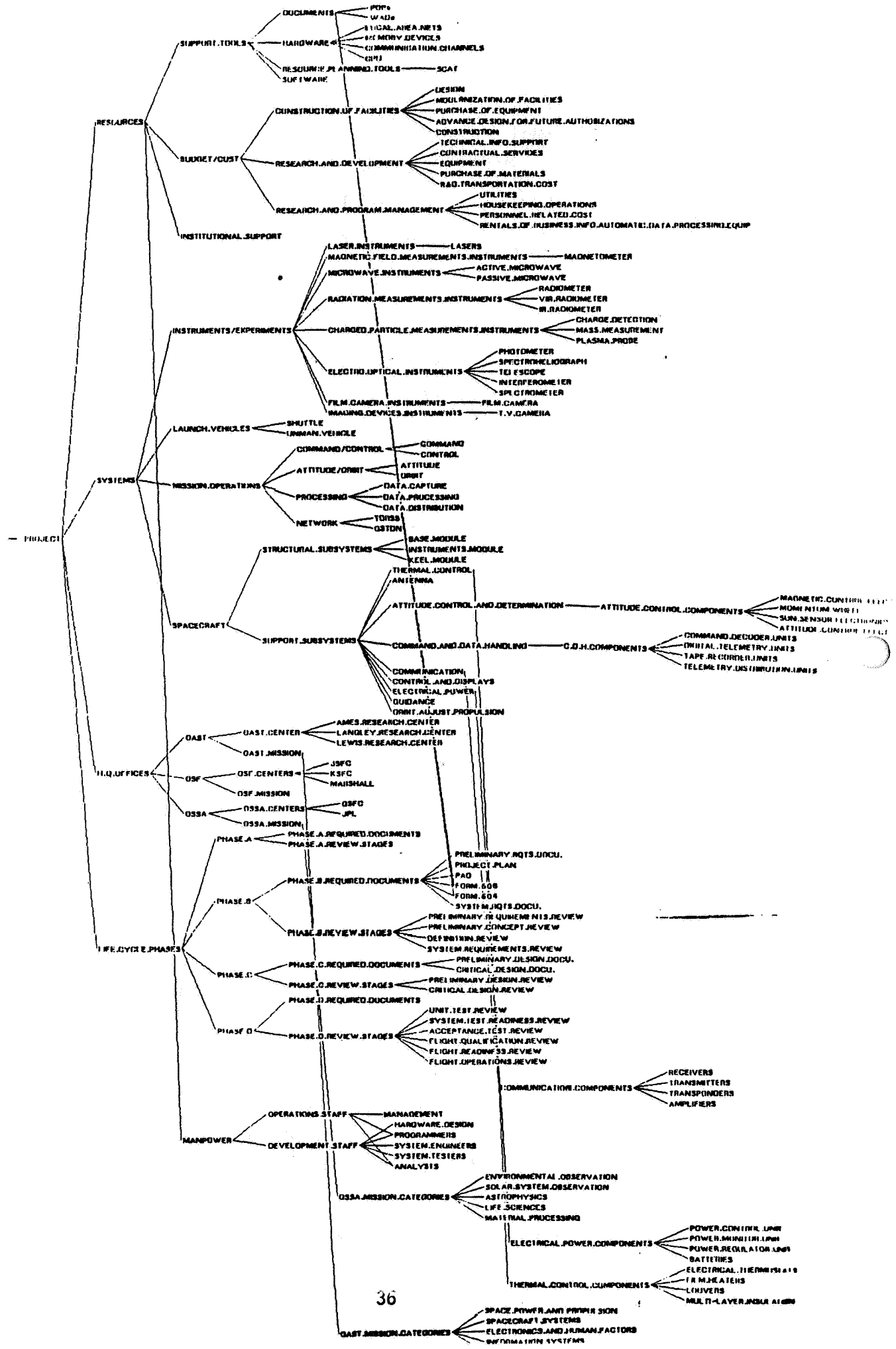
USES EPMS FOR
- PROJECT PLANNING
- PROCESSING STATUS INPUTS FROM SUBMANAGER

- SAME CONCEPT APPLIES AT PORTFOLIO, PROGRAM, PROJECT, SYSTEM, SUBSYSTEM OR TASK LEVEL

USER INTERFACE SUBSYSTEM - EXECUTIVE CONTROL PANEL



KNOWLEDGE BASE – ATTRIBUTE LATTICES



KNOWLEDGE BASE - ANALOGS

- CURRENTLY AT PROJECT LEVEL

- EACH ANALOG CONTAINS THE FOLLOWING KNOWLEDGE:
  - System Attributes
  - Cost Profile
  - Appropriation Accounts
  - Manpower
  - Time
  - Project Organization

- SLOTS AVAILABLE FOR ADDITIONAL OPTIONS
  - WBS
  - PHASES/TIME LINE
  - CONTRACT PERFORMANCE
  - OTHER

- EXAMPLE USER SESSIONS

- OBJECTIVES:

  - PERFORM PLANNING FOR NEW PROJECTS USING LESSONS LEARNED AND OTHER INFORMATION FROM SIMILAR PROJECTS PERFORMED IN THE PAST (SESSIONS 1 & 2)

  - PERFORM PROJECT MONITORING UTILIZING THE CROSS-REFERENCING CAPABILITY OF EPMS (SESSION 3)

SESSION 1

- LOGIN

- DOMAIN SELECTION

- NEW PROJECT DEFINITION
  - SELECT LEVEL
  - SELECT ATTRIBUTES PRESENTED IN MENU GENERATED FOR THAT LEVEL

- ARIEL RUN

- INSPECTION OF:
  - CANDIDATE ANALOGS
  - EVALUATED ANALOGS
  - TARGET PROBLEM
  - TARGET SOLUTION
  - SPECIALIST ACTIVATION REQUESTS (SARs)
  - ANY ELEMENT WRITTEN ON BLACKBOARD

- ARIEL OUTPUT:
  - RECOMMENDED ANALOGS AND CORRESPONDING EVALUATION FACTORS
  - LIST OF ATTRIBUTES FOR WHICH NO ANALOGS HAVE BEEN FOUND

SESSION 2

COST CATEGORY
* TOTAL COST
INSTRUMENTATION
SUPPORT SYSTEMS
OTHER

CROSS-REFERENCE
PROJECT ORG.
APP'N ACCOUNT
SYSTEM ARCH.

COSTS $K / PROJECTS	FISCAL YEARS							
	1	2	3	4	5	6	7	8
ANALOG A								
ANALOG B								
ANALOG C								
TARGET PROJECT								

* CURRENT SYSTEM DEALS ONLY WITH TOTAL COSTS

- COST/PLANNING DATA IS PRESENTED FOR THREE BEST ANALOGS TO THE PLANNED PROJECT
- EPMS HELPS USER MAKE ADJUSTMENTS FOR SIZE DIFFERENTIAL, INFLATION, ETC.
- USER ENTERS PLANNED COSTS FOR NEW PROJECT
- NEW PROJECT COSTS ARE STORED FOR USE IN SESSION 3

SESSION 3

COST CATEGORY
* TOTAL COST
INSTRUMENTATION
SUPPORT SYSTEMS
OTHER

CROSS-REFERENCE
PROJECT ORG.
APP'N ACCOUNT
SYSTEM ARCH.

COSTS $K / PROJECTS	FISCAL YEARS							
	1	2	3	4	5	6	7	8
PLANNED								
ACTUAL								
ANALOG C								
TARGET PROJECT								

* CURRENT SYSTEM DEALS ONLY WITH TOTAL COSTS

- PLANNED DATA IS PRESENTED TO EPMS USER
- USER INPUTS ACTUAL DATA AS PROJECT PROGRESSES
- ANY INPUT EXCEEDING PREDETERMINED THRESHOLDS CAUSES AN ALARM FLAG TO BE PROPAGATED TO ALL CATEGORIES THAT ARE CROSS-REFERENCED TO THE PROJECT (IN THIS CASE PROJECT ORG., APP'N ACCOUNT, SYSTEM ARCHITECTURE)
- FUTURE CALLS TO ANY OF THESE CATEGORIES WILL RESULT IN AN ALARM UNTIL THE CONDITION IS RECTIFIED OR CANCELLED

NEXT STEPS

- CONTINUE DEVELOPMENT/REFINEMENT, INCLUDING FUTURE OPTIONS
- INITIATE CODE 520 APPLICATIONS (TO INCLUDE FIELD TESTING)
- IMPLEMENT EPMS AS A SHELL/GENERATOR TO SUPPORT ANY PROJECT SITUATION

S4-63
233134
12P.

An Expert System for Ground Support of the Hubble Space Telescope

May 15, 1986

Don Rosenthal, Patricia Monger, Glenn Miller[1], Mark Johnston

Space Telescope Science Institute[2]
Homewood Campus
Baltimore, MD. 21218

Abstract

The Hubble Space Telescope is an orbiting optical observatory due to be launched by the Space Shuttle in late 1987. It is a complex, multi-instrument observatory whose resources will be available to the world-wide astronomical community. The "Transformation" system is a hybrid system which utilizes a rule-based expert system to convert scientific proposals into pre-optimized linked hierarchies of spacecraft activities. These activities are generated in a format that can be directly scheduled by the planning and scheduling component of the Space Telescope ground support system. The Transformation system will be described in detail in this paper, with particular attention given to the rule base.

---

[1] Astronomy Programs, Computer Sciences Corporation

[2] Operated by the Association of Universities for Research in Astronomy for the National Aeronautics and Space Administration

1. Overview

Proposals for use of the Hubble Space Telescope are received from around the world, either on paper forms or via our remote proposal submission system. These proposal forms are oriented, as they should be, toward the astronomer who has a particular astronomical objective in mind. In general, the astronomer is properly concerned only with the scientific performance of the observatory which affect that objective (eg, sensitivities of the instruments, available filters), and not with the particulars of how the observation is implemented.

Two of the forms, the target list and proposal logsheet, describe the bulk of the input to Transformation. These forms consist of a few columns each, and exposure lines may be modified by special requirements that describe timing relationships, target acquisition, position or orientation constraints, or repetitions. For example, the exposure logsheet has columns for target name, instrument configuration and mode, aperture and filter to be used, as well as the duration of the exposure. A special requirement might be used to constrain an observation to be made at the same sky orientation as an earlier exposure (e.g. if the light from the object is polarized).

By contrast, the scheduling system requires a detailed description of the spacecraft functions which can then be placed on a timeline and converted into command loads. This description reflects the design of the scheduling software as well as the spacecraft, and is organized into a hierarchy, as follows:

- An "Exposure" is a single instrument operation.
- An "Alignment" is a set of exposures that can be taken without moving the telescope (usually a single instrument and a single target, sometimes multiple instruments and multiple targets).
- An "Observation Set" is a set of alignments that can be performed without affecting the guidance system (that is, without reacquiring guide stars).
- A "Scheduling Unit" is the smallest schedulable entity–it is made up of observation sets.
- Scheduling Unit Links (scheduling units may be linked in time).

Even the first level of the hierarchy cannot be simply mapped from the proposal forms, although both input and output use the term "exposure", as there is so much implict spacecraft activity in a simple scientific description. For example each exposure logsheet line has a handful of attributes, but the scheduling system uses three database relations just for exposures, and the main exposure relation has over 100 fields describing such things as expected data volume, aperture coordinate system descriptions, as well as positions of polarizer and filter wheels.

Obviously, most of the scheduling information is not directly available from the forms. Much of it is derivable, but some is simply a result of decisions made by experts: "Operations Astronomers" who are training themselves in the day to day use of our

ground system. An information expansion factor of about 50 is not extraordinary, and when repeated observations and observation sequences are specified by the proposer, it can be larger than that. Additionally, there are usually several distinct but valid transformations for a given proposal.

2. The Manual Process

When transformation is done by hand, it is divided into two steps. The first step is performed by the Operations Astronomers. They read the exposure logsheets and target lists, and generate a "script" which describes the hierarchical organization they have chosen for that proposal. This script is simply a listing of assignments of exposure logsheet lines to scheduling system exposures, alignments, observation sets, and scheduling units. It also includes the calculated total alignment time, and any timing links that exist between scheduling units. Most scripts are roughly one page long, are handwritten, and may contain comments or warnings to help in the next step. An expert Operations Astronomer can typically generate between one and two scripts per day.

The next step is to take both the script and the proposal to a terminal connected to the planning system, and to enter the details of the hierarchy into the proposal management data base (PMDB). This is performed by "Console Operators", who are aided by a set of written procedures collected into a "cookbook". This cookbook helps the console operators make appropriate choices for the hundreds of fields of the relations in the PMDB. This operation typically takes a period of time equivalent to that of script generation.

3. The Automated System

Transforming proposals by hand is obviously extremely labor intensive, tedious, and therefore error prone, all of which contributed to the decision to computerize the process. Because it is not a well structured problem, because a given proposal might be transformed a number of different ways, and because manual transformation involved considerable expertise, a rulebased approach was adopted.

The Transformation system automates both steps of the procedure described above. The rulebase, written in OPS5, has encoded the expertise of the scripting process as well as the cookbook-aided database generation.

The high level dataflow of transformation is as follows: data is retrieved into the OPS5 working memory and processed by the rulebase. A human readable and editable "assignment-file" is produced by transformation. This assignment file is then run through a processor which converts it into a database command file. This file can then "append itself" into a local database whose relations mimic the operational PMDB. Tuples in the local copy are loaded the operational PMDB on request.

A code generator was used to build the transformation front-end. Given a relation in the logsheet database, and a description of the the OPS5 data structure to receive that data, the code generator writes the actual OPS5 data structure declaration code, as well as the necessary C and database routines to retrieve the data from the data base and load it into OPS5 working memory. The data structure declarations are included in the OPS5 program source, and the routines can be called from the right hand side of an OPS5 rule, as data is needed.

The decision to use the assignment file and its related processing was made for several reasons. Most importantly, however, it is a tremendous aid to building user confidence. When a proposal is transformed, an Operations Astronomer can examine the full assignment file in detail. The Operations Astronomer may make any changes deemed necessary, using any available text editor. The modified file can then be made to "load itself" into the database. In addition, an assignment file summarizer was written which extracts relevant data from the file and reformats it into a close equivalent of a hand-written script. This makes a quick check of the results very straightforward for an Operations Astronomer.

The routines which reformat the assignment file for loading into the PMDB also pull "pass-through" data from the entry database for loading. This pass-through data is typically textual, and may need only minor reformatting. Substantial efficiency is gained by feeding the rule base only that information which is needed for transformation.

4. The Rulebase

The rulebase is goal oriented, utilizing a pre-established chain of goals, as follows:

- Retrieve the input data from the database
- Transform the target data
- Merge exposures into alignments where possible and desirable
- Merge alignments into observation sets
- Merge observation into scheduling units
- Fill in PMDB attributes
- Write the data to the assignment file

Most goals need a bit more structure. For example, the goal for merging exposures has four steps: ordering the exposures, finding potential exposure merges, examining communications needs of the newly formed alignments (the ground system can only support one uplink and/or one downlink per alignment), and finally, assigning the exposure attributes.

The additional refinement is implemented using OPS5 vector attributes, in this case "task-lists". The rule that activates the exposure-merging goal is shown in figure 1, and would be read like this:

```
(p make-goal-merge-exposures

   {<goal-to-remove>
    (goal
        ^has-name              transform-target-data
        ^has-status            satisfied) }

  - (goal
        ^has-name              merge-exposures)

-->

(remove <goal-to-remove>

(make goal
        ^has-name              merge-exposures
        ^has-status            active
        ^task-list             order-exposures
                               find-potential-exposure-merges
                               examine-communications-needs-of-merged-exposures
                               assign-exposure-attributes) )
```

Figure 1: Activating the exposure merging goal

"IF there is a goal which has the name 'transform-target-data' and has the status 'satisfied', AND there is no goal which has the name 'merge-exposures', THEN remove the first goal, AND make make a goal with the name 'merge exposures' and with active status, and with the following task-list: 'order-exposures', 'find-potential-exposure-merges', 'examine-communications-needs-of-merged-exposures' and 'assign-exposure-attributes'.

Most rules refer to a specific goal in their left-hand-sides, and thus can only be instantiated when that goal is active. This is the only sequencing needed in the system, even though goals and tasks will have many associated rules (some have over 60).

Constructing the hierarchy utilizes a technique known as truth maintenance. That is to say, some rules are used to assert facts known to be generally true or useful, and other more specific rules check to make sure that the new assertion violates no constraints, and introduces no contradictions. There are only a few, rather generic assertion rules, but there are many constraint checkers, each testing a very limited special case.

For example, if an observer has requested that an exposure be taken at the same sky orientation as another in the proposal, and the two use the same configuration of the same scientific instrument (SI), then the two exposures can usually be performed in the same alignment. In general, if two exposures can be placed in the same alignment,

then they **should** be placed in the same alignment, as this will reduce slewing and maneuvering overhead. The corresponding rule is listed in figure 2.

To paraphrase the rule:

IF the goal to merge exposures is active, and the present task is finding potential exposure merges, AND there is a link between two exposures specifying that they have the same orientation, AND the exposures in question use the same instrument configuration, THEN make a "mergeable-exposures" record specifiying that they be placed in the same alignment, and note that this is "unmergeable" if a contradiction is found, and that it is a "same-orientation" type merge, with some specified (numerical) level.

Note the use of a table lookup to find the "level" of the exposure merge. In general, exposures may be merged for several reasons, but only the most important or highest level merge need be used. The redundant, less important merges are removed from working memory. Employing a table allows new merging levels to be added, as well as allowing levels to be rearranged, without code changes.

The preceding rule would fire for every pair of exposures which the proposer had specified to be taken at the same orientation, and which used the same instrument configuration. There are potentially a large number of cases where this would actually not be acceptable. Some examples:

- If the High Speed Photometer was to be used, but different filters were requested (the HSP has fixed filters, one per aperture; to change filters the telescope must actually be moved a small amount).
- If one of the exposures was an onboard or interactive target acquisition for the other (by definition within the scheduling system, alignments cannot mix acquisition and data collection exposures).
- If the observer had specified starting times for the two exposures, but the second was to start more than 10 minutes after the end of the first (in that case it might be useful to slew off and do something else, returning for the second exposure later.)

The HSP unmerging rule is shown in figure 3.

Note that this unmerging rule can be applied to other merge types besides "same-orientation", specifically "sequential-no-gap" and "consecutive". This consecutive type merge merits more detailed explanation.

Most of the merging rules take their cue from special requirements explicitly stated by the propoeser. There is usually an implicit link between exposures that are listed consecutively on the exposure logsheet, however, as the proposer often assumes that such exposures will be taken consecutively in time. There is therefore an exposure merging rule that checks consecutive logsheet exposures and attempts to merge them. Since unrelated exposures often are listed consecutively by mere coincidence, there is

a particularly large number of clauses in this merging rule. This is also the lowest category of merges and therefore is susceptible to the most constraint violations and a proportionately large number of unmerging rules.

Once merging and unmerging of exposures has completed, an "assignment record" is generated for each input exposure. Each exposure in the same alignment is given the same alignment order number, and that identifier is recorded in the assignment record. Exposures with the same alignment order are sorted and an appropriate exposure ID is also recorded.

The remaining hierarchy building rules operate in a similar manner. An additional heuristic used at upper levels is as follows: if two lower level entities were unmerged, then they are somehow related, and an attempt should be made to merge them on the next level. After merging and unmerging at each new level, the appropriate identifier is generated and included in the assignment record.

When this first half of transformation is completed, there will be an assignment record for every input exposure, and it will contain the exposure, alignment, observation set and scheduling unit identifiers assigned to that exposure. At this point, data structures corresponding to the PMDB relations are generated for each assignment record, as appropriate, and default values are placed in some fields of those structures. Rules can then fire to change default values, or to assign values to slots which are still empty.

For example, at the observation set level, details of the pointing control system (PCS) need to be described. Gyros can be used if the telescope does not need accurate pointing or stability. This would be the case for an internal calibration (where a shutter is opened, and a lamp of accurately known brightness and color is turned on). If any exposure in any alignment of an observation set is not an internal calibration, however, the Fine Guidance Sensors (FGS) must be used to lock onto guide stars. The corresponding rule is shown in figure 4.

This rule uses the assignment record as a cross reference between a PMDB data structure and the exposure logsheet line that it corresponds to. The single rule is sufficient to check all exposures of a given observation set.

5. Evaluation of the Transformation System

The transformation system was delivered to the Operations Astronomers in December of 1985. It has undergone operational testing since then, and has matured considerably. The present transformation rulebase consists of approximately 375 rules and has been used to transform proposals for scheduling and use during the Ground System Thermal Vacuum test.

The users (who did not mind being automated out of this tedious job) are very satisfied with transformation. It usually performs as well as a human expert, and from time to time the automated transformation has been judged to be better than the manual

version. Cases still come up which are not handled correctly, but these have in the past been easily dealt with by enhancing the rule base.

The adaptibility of the system has been the biggest success of the project. Rulebased systems are reputed to be malleable in general, but there are several reasons why this is true of the Transformation rulebase. Firstly, OPS5 is a particularly expressive language. In addition OPS5 rules, being completely independent of each other, are naturally quite functionally strong, and very weakly data coupled. And, importantly, the problem was well matched to a rulebased solution.

Finally, building the system has served to formalize the "folklore" of transformation, as well as document the process. The notes written by the Operations Astronomers for exposure merging now say, "see cross-reference table for config./mode, or Don's OPS5 rules."

6. Acknowledgements

Dr. Sidney B. Parsons and Dr. Douglas B. McElroy provided the expertise for the rulebase. This project could not have been successful without their contributions.

```
(p find-same-orientation-mergeable-exposures

    (goal
        ^has-name                   merge-exposures
        ^has-status                 active
        ^task-list                  find-potential-exposure-merges)

    (exposure-link
        ^has-proposal-id            <proposal-id>
        ^is-version                 <version>
        ^has-exposure-number        <linked-exposure>
        ^is-linked-to               <main-exposure>
        ^has-link-type               SAME_ORIENT)

    (exposure-specification
        ^has-proposal-id            <proposal-id>
        ^is-version                 <version>
        ^has-exposure-number        <main-exposure>
        ^uses-SI-configuration      <SI-configuration>)

    (exposure-specification
        ^has-proposal-id            <proposal-id>
        ^is-version                 <version>
        ^has-exposure-number        <linked-exposure>
        ^uses-SI-configuration      <SI-configuration>)

    (mergeable-level
        ^symbol                      same-orientation
        ^value                       <same-orientation-level>)

-->

(make mergeable-exposures
        ^first-proposal-id           <proposal-id>
        ^first-version               <version>
        ^first-exposure-number       <main-exposure>
        ^second-proposal-id          <proposal-id>
        ^second-version              <version>
        ^second-exposure-number      <linked-exposure>
        ^is-unmergeable              true
        ^is-mergeable-level          <same-orientation-level>
        ^merge-type                  same-orientation
        ^has-unique-label            (genatom) ) )
```

Figure 2: Merging "same-orientation" exposures

```
(p Remove-HSP-merge-if-filters-differ

    (goal
        ^has-name        merge-exposures
        ^has-status      active
        ^task-list       find-potential-exposure-merges)

   {<mergeable-exposure-link>
    (mergeable-exposures
        ^first-proposal-id          <proposal-id>
        ^first-version              <version>
        ^first-exposure-number      <first-exposure>
        ^second-proposal-id         <proposal-id>
        ^second-version             <version>
        ^second-exposure-number     <second-exposure>
        ^merge-type                 << sequential-no-gap
                                       consecutive
                                       same-orientation >>
        ^is-unmergeable             true) }

   (exposure-specification
        ^has-proposal-id            <proposal-id>
        ^is-version                 <version>
        ^has-exposure-number        <first-exposure>
        ^uses-SI-configuration      << HSP/PHOT HSP/PMT HSP/PRISM
                                       HSP/POL  HSP/ACQ HSP/IMAGE >>
        ^first-spectral-element     <spectral-element> )

   (exposure-specification
        ^has-proposal-id            <proposal-id>
        ^is-version                 <version>
        ^has-exposure-number        <second-exposure>
        ^first-spectral-element     <> <spectral-element> )

-->

(remove <mergeable-exposure-link>) )
```

Figure 3: Unmerging HSP exposures which use different filters

```
(p set-pcs-mode-to-FGS-if-entire-obset-is-not-internal-calibration

    (goal
        ^has-name        assign-PMDB-attributes
        ^has-status      active)

   {<obset-entry>
    (PMDB-obset-entry
        ^has-proposal-id                        <proposal>
        ^has-obset-id                           <obset-id>
        ^has-pcs-mode                           GYRO ) }

    (assignment-record
        ^has-proposal-id                        <proposal>
        ^is-version                             <version>
        ^has-Pepsi-exposure-number              <exp-number>
        ^has-alignment-id                       <alignment-id>
        ^has-obset-id                           <obset-id>
        ^has-exposure-id                        <exposure-id> )

    (exposure-specification
        ^has-proposal-id                        <proposal>
        ^is-version                             <version>
        ^has-exposure-number                    <exp-number>
        ^is-internal-target-type                nil )

-->

(modify <obset-entry>
        ^has-pcs-mode                           FGS ) )
```

Figure 4: A rule which sets the PCS mode

Don Rosenthal is a Senior Computer Scientist in the Observatory Software Branch of the Space Telescope Science Institute. He received an A.B. in Engineering in 1975 and an S.M. in Applied Mathematics in 1976, both from Harvard University.

Patricia Monger is a Senior Scientific Programmer for the Observatory Software Branch of the Space Telescope Science Institute. She has an M. Sc. Degree in Astronomy from the University of British Columbia.

Glenn Miller is an astronomer with the Observatory Software Branch of the Space Telescope Science Institute. He received a Ph.D. in Astronomy from the University of Texas at Austin in 1981.

Mark Johnston heads the Observatory Software Branch of the Space Telescope Science Institute. He received a Ph.D. in Physics from the Massachusetts Institute of Technology in 1978.

S5-61
233135
21P.

AN EXPERT SYSTEM PROTOTYPE FOR AIDING IN THE DEVELOPMENT OF SOFTWARE FUNCTIONAL REQUIREMENTS FOR NASA GODDARD'S COMMAND MANAGEMENT SYSTEM: A CASE STUDY AND LESSONS LEARNED*

Dr. Jay Liebowitz

Department of Management Science
School of Government and Business Administration
George Washington University, Washington, D.C. 20052

ABSTRACT

At NASA Goddard, the role of the command management system (CMS) is to transform general requests for spacecraft operations into detailed operational plans to be uplinked to the spacecraft. The CMS is part of the NASA Data System which entails the downlink of science and engineering data from NASA near-earth satellites to the user, and the uplink of command and control data to the spacecraft. Presently, it takes one to three years, with meetings once or twice a week, to determine functional requirements for CMS software design. As an alternative approach to the present technique of developing CMS software functional requirements, an expert system prototype was developed to aid in this function. Specifically, the knowledge base was formulated through interactions with domain experts, and was then linked to an existing expert system application generator called "Knowledge Engineering System (Version 1.3)." Knowledge base development focused on four major steps: (1) develop the problem-oriented attribute hierarchy; (2) determine the knowledge management approach; (3) encode the knowledge base; and (4) validate, test, certify, and evaluate the knowledge base and the expert system prototype as a whole. Backcasting was accomplished for validating and testing the expert system prototype. Knowledge refinement, evaluation, and implementation procedures of the expert system prototype were then transacted.

This paper will first discuss (Section 1.0) the problem environment of determining CMS software functional requirements, with a special emphasis on the use of analogy. Then, the expert system approach will be discussed for handling CMS requirements development (Section 2.0). Next, validation and evaluation procedures of the expert system prototype will be explained (Section 3.0). Then, lessons learned from developing this expert system, as well as others, will be cited in Section 4.0. Last, conclusions will be drawn in Section 5.0.

1.0 Problem Environment

The Command Management System (CMS) at NASA Goddard Space Flight Center is responsible for transforming general requests for spacecraft operations into minutely detailed operational plans. For each satellite in earth orbit, spacecraft command sequences are needed to position the satellite and to send science and telemetry data from the satellite to its ground control center. These

---

*Part of this paper appeared in the Proceedings of the Expert Systems in Government Conference, IEEE/MITRE, Washington, D.C., October 1985.

command sequences are derived from general requests for specific operations made by experimenters, mission operations personnel, or mission support personnel.

Before these commands can be generated, it is first necessary to develop the software functional requirements which will eventually be transformed into the software used for satellite commands. The present way of discovering the functional requirements for a new mission's command management system software is through a series of meetings, interviews, questionnaires, and documents. Usually a team of contractor personnel with varying years of professional experience is assembled to determine the functional requirements for a new mission's command management system software. Typically, this team will meet with NASA personnel, contractors, satellite project team members, and experimenters (scientists) to synthesize the functional requirements for the CMS. It can take from one to three years of these meetings to ascertain a final list of functional requirements. From these requirements, the necessary software can be designed to accomplish the CMS objectives.

The present technique of gathering functional requirements has various limitations:

- The present method of meeting numerous times with NASA personnel, project team members, experimenters, and contractors is very time-consuming in ascertaining the functional requirements for CMS operation. In fact, by having these meetings and manually recording the results, it has taken one to three years for determining the functional requirements.

- Another element compounding this problem situation is the external pressures of greater frequency of Shuttle launches and less lead time available to develop a new mission's ground system. With less lead time available for the ground system design of a new mission, the luxury of taking up to three years to develop the functional requirements is vanishing.

- Additionally, the team contracted to develop the functional requirements may not consist solely of individuals with many years of professional experience in this area because these persons may be assigned to different tasks within the company. This places a limitation on the wealth of knowledge in terms of professional experience and expertise for determining functional requirements.

To decrease reliance on these individuals, it would be beneficial to NASA to have some automated way of capturing the professional experience of these "experts," before they leave.

In a study performed by Silverman [2-4] on how analogy is used at NASA Goddard, two very interesting paradoxes resulted from a NASA questionnaire response and follow-up interviews. These paradoxes are [3]:

PARADOX 1: Human bias and overconfidence in the face of uncertainty is likely to lead the practitioners to use analogy despite this obstacle.

PARADOX 2: Organizational support for the use of analogy is insufficient and likely to lead to inadequate knowledge acquisition.

These paradoxes suggest that a particular task at Goddard is being done; however, the task is not being accomplished in the most efficient and effective way due to the practitioner's use of suboptimal analogical knowledge [4].

This suboptimal analogical knowledge is caused by the lack of capturing analogical knowledge in the "corporate memory" for later reuse [3]. In the case of determining CMS software functional requirements, there hasn't been a central source that accumulates CMS software functional requirements of previous satellites. Relating to the analogical procedure, the CMS designer, whether knowingly or not, is using analogy as this is an intrinsic operation of the human brain. As a result of analogical analysis, a rudimentary knowledge base is created by the CMS designer. The problem is that the knowledge base is not being developed in any structured way due to the bias pollution caused by the interaction of one's value hierarchies and worldviews, as a result of drawing inferences through analogy [1]. Thus, this knowledge base is not contributing in an optimal sense to the formulation of a "corporate memory." It seems apparent, from the above paradoxes, that a need exists, perhaps in the form of an automated tool, for developing this capability to aid the CMS software designer in the reuse of functional requirements.

The major reason for the need for a new approach for developing CMS software requirements is to enhance analogical analysis. Analogical analysis need to be enhanced because of an inadequate knowledge base resulting from bias pollution, and because of an inadequate procedure for inference. Analogical analysis is predicated upon drawing inference. Optimizing analogical analysis entails minimizing biases in the inference. Inference bias comes primarily from the interactions between the value hierarchies and worldviews of the CMS designer [1]. Through the use of an expert system, bias is dampened out because a "computer" is being used to act as the inference engine. The combination of the software and hardware dampens out bias because expert systems don't have dynamic interactions of value hierarchies and worldviews. By reducing the bias pollution used in drawing inferences, the resulting knowledge base can become purer than that without the use of an expert system.

2.0 Expert System Approach

Many approaches could be used for determining CMS software functional requirements. Among these approaches are data flow-oriented methods (e.g., Structured Analysis and Design Technique), data structured-oriented techniques (e.g., Jackson's method), and data prescriptive methods (e.g., Program Design Language). However, the expert system approach, by nature of its definition and ability to damp out inference bias, allows for specifically capturing the professional expertise of CMS design "experts," which is one of the major problems at hand.

For this research, an expert system application generator was used to develop an expert system prototype for determining software functional requirements for a satellite's command management system. An expert system application generator is a tool used for building expert systems. It is the dialog structure (language interface) and inference engine (expert system component that searches and generates the solution hypothesis) which, when linked to a knowledge base (set of domain facts and rules of thumb), functions as a fully operational expert system. With the use of an expert system application generator, an expert system does not have to be built from scratch and take 10 to 20 man years.

The expert system application generator used for the development of the CMS software requirements expert system prototype was "Knowledge Engineering System" (KES), Version 1.3. KES was developed by Software Architecture and Engineering, Inc., and was modeled after the Knowledge Management System by Dr. James Reggia of the University of Maryland. KES is implemented in Wisconsin LISP, Franz LISP, and IQ LISP, and runs on the VAX, UNIVAC, Symbolics, CDC, DEC, Apollo, and IBM XT.

In using an expert system application generator for expert system development, the major effort is in constructing the knowledge base. The knowledge base is the set of facts and heuristics which are specific to a particular problem domain. There were four major steps used in constructing the knowledge base, to link with KES, to form the expert system prototype for CMS software functional requirements determination.

The first step in developing a knowledge base was formulating the problem-oriented attribute hierarchy [5,6]. This attribute hierarchy is a framework around which the knowledge base will be constructed [7]. It serves as a pictorial diagram in representing the association of components, or "attributes," which ultimately leads to the decision(s) or "ultimate goal(s)" reached by the expert system. An excerpt of the basic structure of the attribute hierarchy for determining CMS software functional requirements is shown in Figure 1. The complete attribute hierarchy consists of 32, 16, and 28 attributes leading to 6, 36, and 139 level 1, 2, and 3 software functional requirements, respectively.

The fragment of the attribute hierarchy in Figure 1 depicts the "ultimate goal" shown at the top level--to obtain third level CMS software functional requirements. The bottom of the hierarchy consists of "attributes" which determine level 1 functional requirements (which eventually lead to level 3 requirements). In this case, the values of these attributes must be supplied by the end-user, and are thus called "input attributes" [6]. All the non-circled attributes in Figure 1 are input attributes. The middle level of the hierarchy depicts intermediate "inferred attributes" (i.e., level 1 functional requirements, circum, level 2 functional requirements) whose values are inferred from the input attributes. The goal at the top level (level 3 functional requirements) is also an inferred attribute, as its value is inferred from the intermediate inferred attributes.

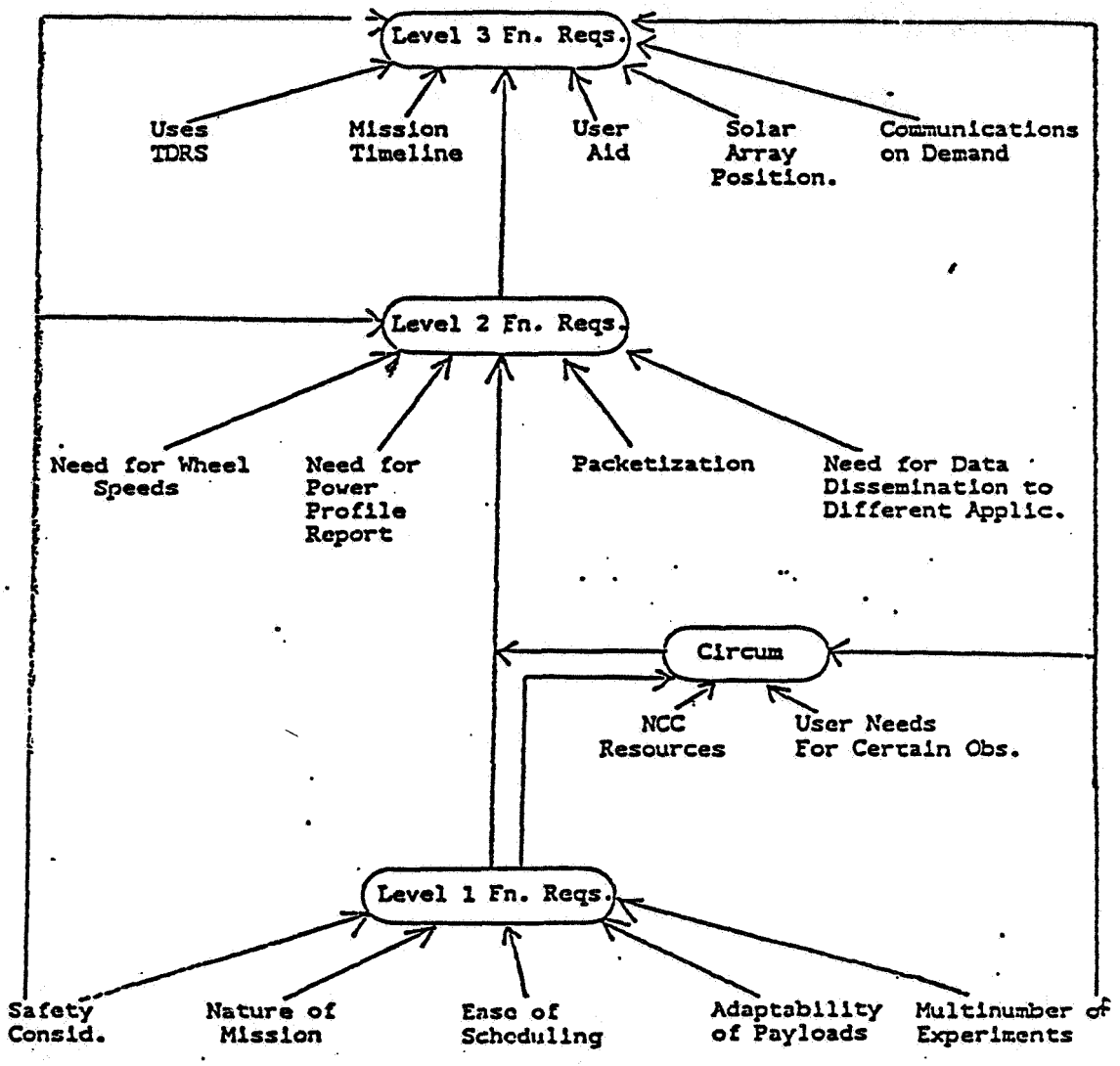
In order to develop the attribute hierarchy for determining CMS software functional requirements, up to three levels, the following steps were used in order:

(1) Revised and studied NASA documents and contractor-generated reports to determine what influences CMS software functional requirements. In particular, the thrust was on discovering the characteristics, or "input attributes," which determined a satellite's CMS software functional requirements. These characteristics represent the bottom line of the attribute hierarchy. These input attributes represent questions to the user.

(2) Met with domain experts to confirm these characteristics and to develop a complete list of characteristics, as additions and deletions were needed.

(3) Through consultations with the domain experts, it was agreed that the knowledge base, as a first cut, should be modeled after the CMS software functional requirements of the <u>most comprehensive</u> satellite to date. The

Figure 1 : BASIC STRUCTURE OF PROBLEM-ORIENTED ATTRIBUTE HIERARCHY FOR DETERMINING CMS SOFTWARE FUNCTIONAL REQUIREMENTS (NOT ALL ATTRIBUTES ARE SHOWN)



reason for this thinking was that the most complex satellite would have the most exhaustive list of software functional requirements. Thus, it would be easier to delete requirements for a less sophisticated satellite than it would be to add many requirements. Using this logic, Space Telescope (ST), which will be launched in August 1986, was used as the model for developing CMS software functional requirements. Since ST represented one of the three future classes of satellite missions, using ST seemed to a reasonable approach.

(4) After reading through numerous ST requirements documents and talking with the domain experts, the level one CMS software functional requirements were developed. This level is represented in the next tier above the bottom level in the attribute hierarchy. After lengthy discussions with the experts, the bottom level attributes were linked to the level one software functional requirements. In other words, the determination was made of "which bottom level attributes influenced which level one requirements."

(5) Next, the level two CMS software functional requirements were developed via ST documents and discussions with the domain experts. Then, the experts related what bottom level characteristics (attributes) influence the level two requirements and what other attributes influence how to get from level one to the level two requirements.

(6) Last, the level three CMS software functional requirements were determined through reading ST documents and discussing the information with the experts. Then, the experts related the bottom level attributes which influence how to get from the level two to the level three requirements.

Once the problem-oriented attribute hierarchy was developed through interactions between the knowledge base author (KBA) and expert, the next major step in knowledge base development was the selection of the knowledge management approach [6]. This deals with the best way of representing the knowledge acquired from the expert.

According to Reggia [5] and Software Architecture & Engineering, Inc. [6], there are three criteria to use in selecting a knowledge management approach. These are: (1) pre-existing format of the knowledge, (2) type of classification desired, and (3) context-dependence of the inference process [6].

The pre-existing format of determining CMS software functional requirements is already organized as rules. Rules are implicitly used in the present way of developing CMS software requirements. The CMS designer thinks of rules in the IF-THEN format but does not formally list the rules. The CMS designers indicate that requirements for new satellites are based on previous requirements for earlier satellites in cases of similar satellite apparatus and functions. By using a rule-based deductive expert system prototype, the knowledge can be kept in the same form as presently being used, thus creating an intuitive appeal [5].

The second factor to be considered for knowledge management selection is the type of classification. Classification types are typically either probabilistic, categorical, or mixed [5]. In the CMS software functional requirements domain, the classification is predominantly categorical. This can be seen as most of the decisions in the requirements determination are yes-no answers. For example, the question of needing real-time user response can be answered by yes

or no. Additionally, since there are three major classes of future satellite missions, analogical reasoning can be used to categorize, under each class, previous satellite CMS requirements information. Conditional probabilities, needed for a Bayesian-probabilistic approach to classification, are almost impossible to determine, in the CMS case, due to a large volume of test data necessary to compute a priori probabilities.

The last major consideration in adopting a knowledge management scheme is the context-dependence of the inference process. Rule-based deduction is an appropriate method for this problem domain when considering context-dependence. This is true because the number of attributes per antecedent is very low [6]. This is helpful for writing a set of rules as all of the context for using each rule does not have to be included in the antecedents of the rule [6].

Based upon the aforementioned criteria, a rule-based deduction approach should be used as the knowledge management (representation) method.

After deciding on a rule-based approach, the next phase involved writing the rules based upon the problem-oriented attribute hierarchy and discussions with the experts. The rules, in handwritten form, were shown to the experts for their comments. Some rules had to be changed due to omissions of attributes, incorrect cause-and-effect relationships, and inappropriate attribute names. Since each rule was in the form "IF antecedent THEN consequent" (certainty factor), the certainty factors next needed to be determined by the expert. These certainty factors were determined by the expert's best judgment, in which values between -1.0 (absolutely false) and 1.0 (absolutely true) were assigned to the consequent of each rule. KES handles uncertainty in its production rule subsystem, in the same manner as does MYCIN, an expert system designed to diagnose bacterial infections in the blood (for more information, see [8,9]).

The third major step in knowledge base development was encoding the knowledge base [5, 6]. This step involves creating a knowledge base using a text editor and storing the knowledge base in a file in the computer's memory [7]. Once encoded, the knowledge base is submitted by the knowledge base author to the KES production rule subsystem (KES.PS) which parses it to create objects that can be operated on by the LISP codes and to check for errors, much as a compiler examines a high-level language program [7].

For encoding the knowledge for determination of CMS software functional requirements, a VAX computer was used, through the kind permission of Software Architecture & Engineering, Inc. A file was created via the use of the UNIX editor. In this file, the knowledge for the appropriate KES sections--certification, attachments, references, attributes, rules and actions--was encoded.

The certification section indicates the knowledge base author, the date of knowledge base development and testing procedures, and special acknowledgements. When "display certification" is entered, the contents of the certification section are displayed to the end-user; otherwise, the message "this is an uncertified knowledge base" is displayed [7].

The attachments section follows the certification section. The attachments section allows the knowledge base author to use free-text attachments for providing definitions, synonyms, calculations, explanations, and questions of an attribute or attribute value. These free-text comments facilitate better

understanding and readability on the part of the user due to enhanced explanation and interaction facilities.

The next section included in this prototype's knowledge base is the references section. This section allows the knowledge base author to cite any sources used to develop the knowledge base. This section, as well as the certification section, is optional for inclusion in a knowledge base. When "display references" is used, the references are displayed to the end-user.

Following the references section in the prototype's knowledge base is the mandatory "attributes" section. This section must be included in a knowledge base, and serves in declaring the attributes shown in the attribute hierarchy and any associated optional attachments (synonyms of attributes) [7]. All attributes in the attribute hierarchy must be included in the attributes section.

The rules section, another mandatory section in the prototype's knowledge base, follows the attributes section. The rules are in the form: if antecedent(s) then consequent(s) (certainty factor). Figure 2 shows an excerpt of the 154 rules for the expert system prototype, as developed through the discussions and iterative rule refinements with the domain experts. The antecedents and consequents are made up mainly of statements, like "nature of mission=commandable." Logical connectors (& and "and," / for "or") can be used for separating antecedent statements, where disjunction has precedence over conjunction [7]. The "/" cannot be used for separating consequent statements. The certainty factors in the rules section are developed by the domain expert's best judgment, and they can easily be changed if needed.

The last section in the prototype's knowledge base, as in all KES knowledge bases, is the "actions" section. Through the actions sections, the knowledge base author can directly influence the operation of the expert system [7]. This can be achieved by including commands (e.g., askfor, obtain, display) in the actions section to control the execution of the expert system. Also, messages can be included in the knowledge base and would then be displayed to the end-user in order to make the expert system easier to use. The actions section must be included in a knowledge base.

Various human factor design elements were incorporated into the expert system prototype. One of the human factors design considerations which was incorporated into the expert system prototype was the use of messages. Constructive and helpful messages were used within the actions section of the knowledge base. These messages:

(1) provide instructions for operating the expert system;

(2) provide reasons if being exited from the expert system;

(3) provide descriptions of some pertinent KES.PS commands and how to use them;

(4) acknowledge the end of the session, and provide information for restarting or terminating the user-expert system session;

(5) allow for information rephrase; and

FIGURE 2: EXCERPT OF THE RULES SECTION OF THE KNOWLEDGE BASE
(TOTAL OF 154 RULES)

```
rule75 if level one functional requirements = data management,
& orbit data = present,
/ telemetry data = present,
/ scheduling data = present,
/ obc data = present,
& need for installation and verification and maintenance and security of data base = present,
then level two functional requirements = manage project data base
& archive data files (0.5).

rule76 [rationale: "This is needed to know if packetization is present."]
  if level one functional requirements = data management,
& telemetry data = present,
& packetization = present,
then level two functional requirements = build telemetry packages.

rule77 if level one functional requirements = data management,
& scheduling data = present,
then circum = need to resolve conflicts and coordinate schedule data with attitude and mission scheduling needs
(0.9).

rule78 if circum = need to resolve conflicts and coordinate schedule data with attitude and mission scheduling
needs, then level two functional requirements = format ncc schedule data (0.8).

rule79 if level one functional requirements = data management,
& onboard computer utilization = present,
then level two functional requirements = maintain flight data base parameters (0.8)
& develop facility to maintain obc software.

rule80 if level one functional requirements = operations support,
& interactive mode = present,
& display = present,
then level two functional requirements = support executive display interface.
```

(6) put conclusions into perspective.

Examples of some of these messages are taken from a user session and are shown below:

> These are the major characteristics influencing CMS software functional requirements determination for a new satellite. Each characteristic will now be displayed along with its associated value(s) and certainty factor(s). The certainty factor ranges inclusively between -1.0 (absolutely false) to 1.0 (absolutely true), and its default value is 1.0.
>
> .
> .
> .
>
> coupling and interrelationships between experiments: present (1.0).

Messages guide the user through the expert system session and they keep the user informed as to what is happening during the session.

A second technique used in the expert system for effective end-user interface design is the use of menus. Menus allow the user to choose what he/she would like the expert system to perform. Menus allow users simply to recognize items rather than to recall them [10]. To avoid menus from becoming tedious if the choices are too finely detailed [11], KES allows the user to also answer "unknown" or "none" if the user is not sure of the answer or feels none of the answers apply. Menus were used in the expert system to:

(1) determine if the user wants to begin the session;

(2) determine if the user would like references displayed;

(3) ask the user if he/she wants:

   (a) to obtain a justification of the requirements just reached by the expert system,

   (b) to continue to obtain lower-level requirements,

   (c) to determine requirements for another satellite,

   (d) to stop.

An example of a menu used in the expert system prototype is shown below through an excerpt of a user session:

> What do you want to do now
>
> (1) get a justification of the first level requirements reached by the requirements aid
>
> (2) continue to obtain the second and third level requirements
>
> (3) determine requirements for another satellite
> (4) stop--I am finished
> = ? (multiple answers permitted) 2

user needs for certain observations:
(1) absent
(2) present
= ? explain
.
.
.

The third human factors design element was to use attachments in the knowledge base [12]. Attachments allow for free-text comments and they were used as follows:

(1) to provide descriptions or attributes and attribute values ("explain" attachment);

(2) to allow for system-generated questions to be posed to the user ("question" attachment)

(3) to allow for synonyms of attributes to be used when typing the knowledge base ("synonyms" attachment); and

(4) to provide rationale for rules ("rationale" attachment).

Through the use of these attachments, the user could obtain descriptions of attributes, attribute values, and rules if the user were unsure of what particular items meant. For example, an excerpt of a sample user session is shown below:

command sequencing planning aids:

(1) absent
(2) present
? 2

instrument management aids:
(1) absent
(2) present
? 2

initial orbits:
(1) complex
(2) average
(3) simple
? explain 1

Command encountered--deferring current question

Refers to a very complex and protracted initial stabilization and attitude control.

Continuing previous line of questioning

```
initial orbit:
(1)  complex
(2)  average
(3)  simple
  ?  explain 3

Command encountered--deferring current question

Refers to very little or no special software or support required for
initial stabilization and control.

Continuing previous line of questioning
```

Other human factor design elements were included in the expert system prototype. These included: providing precautions against undesirable inputs through supplying additional production rules, and having a "help" function for the inexperienced user.

A justification facility, as part of KES, is another special and important feature which is contained within the expert system prototype. The justification facility is vital because the user can see how and why certain attribute values were determined, which creates trust and believability in the expert system. In KES, a justification facility is provided so that conclusions can be justified in the form:

```
a = b

reasons for this statement:

c = d

    .
    .
    .

f = g

(by rule: x1)
```

An excerpt from a user session where the justification process was invoked for level two functional requirements is shown below. This shows a partial listing of the "justified" level two requirements:

```
for level two functional requirements =
     model reaction wheel speeds . . .

Reasons for this statement:

     level one functional requirements =
       mission scheduling and planning
     need for wheel speeds =
       present
     (by rule:  rule46 (1.0))
```

```
for level two functional requirements =
          predict system power usage . . .

Reasons for this statement:

     level one functional requirements =
       mission scheduling and planning
     safety consideration =
       present
     need for power profile report =
       present
     (by rule:  rule47 (1.0))
```

3.0 Validation and Evaluation of the Expert System Prototype

Thus far, three of the four major steps in constructing the knowledge base have been discussed. The fourth step in knowledge base development was testing, certifying, and evaluating the knowledge base and resulting expert system.

KES has two tools which help in validation of the expert system, namely the KES Parser and Inspector. The KES Parser was automatically activated whenever the user needed to run a sample session using KES. It detected a wide variety of syntax errors and inconsistencies. Inspector, contained in the production rule subsystem of KES (KES.PS), detects unattached attributes and direct/indirect recursions, and derives the actual hierarchy from the knowledge base. These tools were employed and were very useful in checking the knowledge base, both syntactically and structurally.

Besides using the Parser and Inspector, backcasting and analogical mapping were used to verify the validity of the expert system-derived requirements for one of the major functions of a spacecraft, Solar Maximum Mission (SMM). Complications occurred in comparing the expert system-generated requirements with the documented requirements for SMM's generator function. There was very poor match-up between the documented requirements and the expert system-generated requirements, as demonstrated by the high rate of mismatch shown in Table 1.

There has been a lack of standardized language used over the years in expressing documented requirements from one satellite to the next. Therefore, in order to adequately map the expert system-generated requirements to the documented requirements, a post hoc salvage operation took place in which a "translated" version of the documented requirements was developed (i.e., the translated version served as the results of the mapping).

Analogical measurement was determined by using pertinent dimensions as expressed in Gentner's [13] and Silverman's [14] frameworks. Their technique determined the "goodness" of an analogy (i.e., how well the fit is between the target and the base). The translation process showed, only for a single function of a single satellite, that the expert system fared well even though the match-up (i.e., exact wording) failed between the documented requirements and the expert system requirements. These results, however, are severely biased because the same person was used to validate the expert system as well as to develop it.

To evaluate the expert system prototype, the Analytical Hierarchy Process was used to quantify subjective judgments made in decision-making. In order to

Table 1 : Degree of Mismatch/Match Between the Wording of the Documented Requirements and that of the Expert System (E.S.) Requirements [a]

Test Case		Number of E.S.-Generated Requirements	Number of Documented Requirements	Number of E.S. to Doc. Matches of Similar Wording	Number of E.S. to Doc. Mismatches of Similar Wording	Number of E.S. Reqs. that should Not be Listed [b]	Number of Doc. Reqs that were omitted but should Not have Been Omitted [b]
1. Model Case:							
ST	Level 1	6	6	6	0	0	0
	Level 2	36	36	36	0	0	0
	Level 3	139	139	139	0	0	0
2. SMM	Level 1	6	3	0	6	0	0
	Level 2	31	14	3	28	0	1
	Level 3	76	72	5	71	8	0
3. DE	Level 1	6	3	1	5	0	2
	Level 2	13	11	0	13	0	0
	Level 3	30	21	1	29	4	0

[a] Sources:

(1) Computer Sciences Corporation, Space Telescope POCC Applications Software Support Functional Requirements, Contract NAS5-26685, July 1983.

(2) Computer Sciences Corporation, Command Management System for Solar Maximum Mission, Critical Design Review, December 1978.

(3) Computer Sciences Corporation, Updates to DE-A and -B Command Management System Requirement Definitions, Contract NAS5-24300, June 1980.

[b] (4) Discussions with Thomas Pfarr, of Computer Sciences Corporation, and Patricia Lightfoot, Nancy Goodman, and Robert Dutilly, of NASA Goddard.

facilitate use of the Analytical Hierarchy Process, a software package called "Expert Choice" [15] was used in which relative priorities were derived by calculating eigenvalues and eigenvalues of matrices representing pairwise comparisons of criteria and alternatives. The evaluation criteria, derived from the works of Boehm et al. [16] and Gaschnig et al. [17], are shown in Table 2: These criteria envelop Gaschnig's [17] set of evaluation criteria, and closely match Boehm's [16] classification of evaluation criteria.

The "Expert Choice" results from the three CMS software designers who used and evaluated the expert system prototype indicate that each user overall preferred the expert system prototype (KES) over the status quo (status) based upon the user's pairwise comparisons of both the criteria and the alternatives. It should be realized, however, that the sample size was extremely small which introduces bias into the evaluation results.

4.0 Lessons Learned from Developing the Expert System

There were some very helpful lessons learned from developing the expert system prototype for NASA CMS software requirements determination. The expert system construction hints, along with those from Hayes-Roth et al. [18], Feigenbaum et al. [19], Parsaye [20], Liebowitz [21-23], and Johnson [24], are shown below:

Task Suitability

- Focus on a narrow specialty area that does not involve a lot of commonsense knowledge.
- Select a task that is neither too easy nor too difficult for human experts.
- Define the task very closely.
- Commitment from an articulate expert is essential.
- Expert systems could have great payoffs in mundane tasks, not necessarily heroic ones.
- Pick a task that will have a significant impact of value to the organization.
- Make sure there is "standardized" language being used within the task.

Building a Prototype

- Become familiar with the problem before beginning extensive interaction with the expert.
- Clearly identify and characterize the important aspects of the problem.
- Record a detailed protocol of the expert solving at least 1 prototypical case.
- Choose a knowledge engineering tool or architecture that minimizes the representational mismatch between subproblems.

Table 2: Evaluation Criteria

Criterion	Definition	Boehm's (B) Category [16] and Gaschnig et al's (G) Category [17]
o Ability to update	Is able to reflect changes in spacecraft characteristics and CMS software functional requirements.	(B) Modifiability (G) System's Efficiency
o Ease of Use	Refers to clear understandability and user-friendliness features.	(B) Human-engineering (G) Quality of the Human-Computer
o Hardware	Refers to the necessary computer equipment for determining CMS requirements.	(B) Portability (G) System's Efficiency
o Cost-Effectiveness	Refers to the costs and benefits involved in determining CMS requirements.	(G) Cost-Effectiveness
o Discourse (Input/Output Content)	Refers to the reasoning approach, help capabilities, and explanation facilities.	(B) Understandability (G) Correctness of the Reasoning Techniques Used
o Quality of decisions, advice, and performance	Refers to the accuracy and completeness of responses.	(B) Reliability (G) Quality of the System's Decisions and Advice
o Design time	Refers to the length of time to develop CMS requirements for a satellite.	(B) Testability

- Start building the prototype version of the expert system as soon as the first example is well understood.
- Work intensively with a core set of representative problems.
- Identify and separate the parts of the problem that have caused trouble for AI programs in the past.
- Build in mechanisms for indirect reference.
- Separate domain-specific knowledge from general problem-solving knowledge.
- Aim for simplicity in the inference engine.
- Don't worry about time and space efficiency in the beginning.
- Find or build computerized tools to assist in the rule-writing process.
- Pay attention to documentation.
- Don't wait until the informal rules are perfect before starting to build the system.
- When testing the system, consider the possibility of errors in input/output characteristics, inference rules, control strategies, and test examples.
- It is good to have redundancy in the expert system in order to give stability to the system (have multiple evidence).
- Have dynamic explanations not static ones.
- Sometimes it might be better to try to get the surface-level rules as opposed to the deep-level knowledge.
- From the beginning, the knowledge engineer must count on throwing efforts away.
- There is a perfect task for every expert system shell or tool, but there is <u>not</u> a perfect shell or tool for every task.

<u>Extend Prototype</u>

- Build a friendly interface to the system soon after the first prototype is finished.
- Provide a "gripe" facility so users can record their complaints about the system without the knowledge engineer being there.
- Keep a library of cases presented to the system.
- It would be wonderful to have multi-layered explanations--ones for the expert, ones for the apprentice, and ones for the end-user.
- If none of the tools you normally use works, build a new one.

- Dealing with anything but facts implies uncertainty.
- The expert system must have very easy ways of allowing the knowledge to be modified so that new information can be added and out-of-date information deleted.

Finding and Writing Rules

- Use the terms and methods that the expert uses.
- If a rule looks big, it is.
- If several rules are very similar, look for an underlying domain concept.

Expert

- You can't be your own expert.
- Give the expert something useful in the way to building a large system.
- Expert will modify rules and values until he/she finds the right answer.
- Insulate the expert, as well as the user, from technical problems.
- Be careful about feeling expert.

Evaluating the System

- Ask early about how the expert would evaluate the performance of the system (i.e., criteria, error rate, etc.).
- The user interface is crucial to the ultimate acceptance of the system.
- Use blind verification studies for testing and validation.

With these tips from the sources mentioned above, the job of the knowledge engineer will be made much easier.

4.1 NASA's Expert System Advice

Tom Davis [25], program manager for artificial intelligence in NASA's Advanced Project and Technology Office at Kennedy Space Center, offers ten lessons for newcomers in building expert systems. These are:

1. Clearly identify the problem to be solved by the expert system and pick one that you think is much too small. However, make sure that this problem is important to the organization.
2. It is a good idea to use consultants while developing your expert system. They can provide valuable advice and save you a lot of money.
3. Prototyping is important--"You need a Gee Whiz System with a lot of bells and whistles if you're going to impress management to get funding."

4. Do a feasibility study to determine if the costs of developing this expert system will support the benefit that will be derived from it.

5. Find a real expert to spend time with. The fewer the better.

6. Use microcomputers for early development and prototyping and then move to specialized hardware. This way you can verify if it is feasible to model this problem with an expert system and save a lot of money if it isn't.

7. Consider using an expert shell for development purposes because it takes someone about five years to become a proficient programmer in a symbolic language.

8. For large-scale applications use hardware that is specifically designed for symbolic software. However, you should carefully analyze which software is best for your application because every hardware manufacturer seems to have their own symbolic software that they think is best. Before any hasty decisions are made you should talk to people who have had experience with that hardware and software and consider their opinions.

9. Train personnel internally to be "knowledge engineers." It is extremely expensive to hire one with experience.

10. Don't hire Ph.D.s to develop the project; only use them as consultants.

5.0 Conclusions

From the results of this research, an expert system approach to determining a sound "first cut" at developing CMS software functional requirements for a new satellite has the potential to be a viable alternative to the status quo. Biases generated from the validation and evaluation procedures of the expert system prototype greatly cloud the results of this prototype's effectiveness. More users are needed to evaluate the expert system as well as the need to perform backcasting of more satellites, using impartial validators, to further validate the expert system. If continued work is done in these areas, then the expert system approach to developing CMS software functional requirements has the potential to contribute to the following goals:

(1) help in standardizing requirements development and language and, in turn, contribute to one of Goddard's goals of minimizing the uniqueness of command management systems;

(2) aid in facilitating the acquisition and representation of expert knowledge on CMS software functional requirements determination;

(3) enhance the CMS software designer's use of the analogy approach; and

(4) provide time and cost improvement over the manual status quo for developing CMS software functional requirements.

Acknowledgement

This paper is based upon the author's dissertation research [21]. Many individuals contributed their time and advice during the development of this

research. Great appreciation is extended to Barry Silverman, Sam Rothman, Ernest Forman, Thomas Nagy, Erik Winslow, Homer Sewell, James Dinwiddie, Robert Waters, Peter Kurzhals, Thomas Grenchik, Thomas Pfarr, Barbara Greenall, and Rita Hodge. Special thanks are for Patricia Lightfoot, Andrew Ferrantino, Nancy Goodman, and Robert Dutilly. A tremendous amount of appreciation is also due to Software Architecture and Engineering, Inc. and Decision Support Software. Part of this research was supported by a NASA Fellowship, NGT-09-010-800.

The views expressed in this paper are solely those of the author and do not reflect views of those at NASA Goddard Space Flight Center.

REFERENCES

1. Sewell, H.B., "Analogical Reasoning Theory," (in formulation), Department of Engineering Administration, George Washington University, Washington, D.C. 20052, 1984.

2. Silverman, B.G., "The Use of Analogs in Systems Engineering: A Software Programming Case Study," Engineering Administration Report, George Washington University, Washington, D.C. 20052, 1983.

3. Silverman, B.G., "Potential Software Management Cost and Productivity Improvements: An Analogical View," IEEE Computer, May 1985.

4. Silverman, B.G., "The Software Engineering Paradox: Unexploited Cost Savings & Productivity Improvements," Journal of Test and Evaluation, Vol. V, No. 1, January 1984.

5. Reggia, J.A. and B.T. Perricone, KMS Manual, Department of Mathematics, University of Maryland, January 1982.

6. Software Architecture & Engineering, Inc., Knowledge Engineering System: Knowledge Base Author's Reference Manual, Arlington, Virginia, March 1984.

7. Software Architecture & Engineering, Inc., Knowledge Engineering System: General Description Manual, Arlington, Virginia, August 1, 1983.

8. Shortliffe, E.H., Computer-based Medical Consultations: MYCIN, American Elsevier: 1976.

9. Buchanan, B. and E.H. Shortliffe, Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley, 1984.

10. Allen, R.B., "Cognitive Factors in Human Interaction with Computers," in Directions in Human/Computer Interaction, ed. by A. Badre and B. Shneiderman, Ablex Publishing, 1982.

11. Savage, R.E. and J.K. Habinek, "A Multilevel Menu-Driven User Interface: Design and Evaluation Through Simulation," in Human Factors in Computer Systems, ed. by J.C. Thomas and M.L. Schneider, Ablex Publishing, 1984.

12. Software Architecture & Engineering, Inc., Knowledge Engineering System: Knowledge Base Author's Reference Manual, Arlington, Virginia, November 1983.

13. Gentner, D., "Structure-Mapping: A Theoretical Framework for Analogy," Cognitive Science 7, 1983.

14. Silverman, B.G., "A Good Analogy and How to Measure It," Institute for Artificial Intelligence Technical Report, George Washington University, Washington, D.C. 20052, 1984.

15. Decision Support Software, Inc., Expert Choice Manual, McLean, Virginia, 1983.

16. Boehm, B., J.R. Brown, H. Kaspar, M. Lipow, G.J. MacLeod, and M.J. Merrit, Characteristics of Software Quality, North Holland, 1978.

17. Gaschnig, J., P. Klahr, H. Pople, E. Shortliffe, and A. Terry, "Evaluation of Expert Systems: Issues and Case Studies," in Building Expert Systems, ed. by F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, Addison-Wesley, 1983.

18. Hayes-Roth, F., D.A. Waterman, and D.B. Lenat. Building Expert Systems, Addison-Wesley, Reading: MA, 1983.

19. Feigenbaum, E.A. and P. McCorduck. The Fifth Generation, Addison-Wesley, Reading: MA, 1983.

20. Parsaye, K. Tutorial on Expert Systems. Expert Systems in Government Conference, IEEE, Washington, D.C., October 1985.

21. Liebowitz, J., "Determining Functional Requirements for NASA Goddard's Command Management System Software Design Using Expert Systems," Dissertation, George Washington University, Washington, D.C., December 1984.

22. Liebowitz, J. Notes on Expert Systems: A Practical Application of Artificial Intelligence, Continuing Engineering Education Program, George Washington University, Washington, D.C., 1985.

23. Liebowitz, J. Expert System Design for Practical Use (in formulation), 1986.

24. Johnson, J. Expert Systems: For You?, Datamation, February 1984, pp. 82-88.

25. Handel, S. "NASA's Expert System Advice," Applied Artificial Intelligence Reporter, University of Miami ICSRI, Vol. 3, No. 2, February 1986.

S6-63
233136
15P.

THE DESIGN AND APPLICATION OF A TRANSPORTABLE INFERENCE ENGINE (TIE1)

David R. McLean

Bendix Field Engineering Corp.
10210 Greenbelt Rd., Suite 310
Lanham, Maryland 20706

Abstract

A Transportable Inference Engine (TIE1) system has been developed by the author as part of the Interactive Experimenter Planning System (IEPS) task which is involved with developing expert systems in support of the Spacecraft Control Programs Branch at Goddard Space Flight Center in Greenbelt, Maryland. Unlike traditional inference engines, TIE1 is written in the C programming language. In the TIE1 system, knowledge is represented by a hierarchical network of objects which have rule frames. The TIE1 search algorithm uses a set of strategies, including backward chaining, to obtain the values of goals. The application of TIE1 to a spacecraft scheduling problem is described. This application involves the development of a strategies interpreter which uses TIE1 to do constraint checking.

Introduction and Motivation

The search for a balance between complexity and generality, on the one hand, and simplicity and narrowness of application, on the other, has lead many AI practitioners to consider implementing custom built tools. During the course of investigating various commercial expert system shells, it became evident that choosing the right shell was no easy task. Lee and Roach [1986], note that generally the available tools impose too many restrictions or they are not user friendly. Most hybrid tools like ART and KEE are not only expensive but should be considered as research tools rather than practical tools that can be used for rapid prototyping, Harmon and King [1985].

From another point of view, Technology Development Chief Robert H. Brown of Johnson Space Center, Marsh [1985], notes that NASA cannot get into a mission-critical situation where they have to depend upon a private company for help. Mr. Brown wants to have the source code for his own expert system shell written in the C language. NASA is also becoming increasingly concerned about the cost of commercial software. Custom tools have a number of advantages which are listed below:

1. The source code is readily available so that extensions and enhancements can be made rapidly.

2. There are no licensing costs when the system is copied or ported to other machine environments.

3. Custom tools are designed specifically for direct application to the task at hand. There is no need to "force fit" the tool to the application.

4. Custom tools written in C can be made to execute faster than most commercially available tools.

Many investigators, including Waterman [1986], have suggested that the development costs of custom inference engines are prohibitive. In addition, they state that developing an expert system in a general-purpose programming language such as FORTRAN, PASCAL, or C further increases the development time. These arguments are certainly valid if the inference engine has to be developed from scratch for each application. However, if the inference engine being developed is general enough to be of use in a large number of applications then the development cost can be recovered many times over. The advantages of custom tools discussed in this section have provided the major motivation for the development of the TIE1 system.

Choice of Implementation Language for TIE1

The following requirements were identified for the implementation language for the TIE1 system:

1. Portablility was seen as the most important requirement for TIE1.

2. It was necessary for TIE1 to interface well with various types of hardware.

3. It was necessary for TIE1 to be implemented in a structured, high level language for ease of modification and maintenance.

Traditionally, inference engines have been written in LISP and PROLOG because these languages were written specifically for AI applications. Unfortunately, these languages are not standard enough across the variety of hardware required for our applications. Our strict requirements for interfacing and portability essentially eliminated these languages from final consideration.
The strict requirements noted above suggested that we consider a language like C. C is very portable and was designed for writing interface software. Also, a current trend in AI is to develop a system in a LISP environment and

then translate the code to a compilable language such as C. The C programming language supports nearly all the capabilities of LISP, including recursion, dynamic memory allocation and complex data structures, and therefore appears to be entirely adequate to develop expert system technology.

Traditionally, C has not been used to develop AI systems because debugging and compilation were so time consuming. The author would like to suggest that this notion is somewhat dated. New high speed microprocessors, such as the MC68020, along with good program development strategies, such as modular structuring, have made compilation times acceptable. In addition, there is an ever increasing number of debugging tools available today. The small price paid in compilation and debugging time is a bargain when one considers the benefits of C: portability, execution speed, and maintainability.

TIE1 Knowledge Representation

A primary goal for the representation of knowledge in the TIE1 system was to provide a structure which was easily readable by people yet rich enough to capture a large variety of knowledge types. The notions of objects and frames were incorporated because of their generality and flexibility. The TIE1 knowledge representation structure is based on a hierarchical network of objects. The overall structure of each object is as follows:

1. Name of object
2. Value of object
3. Default value of object
4. Attribute list
5. Instance list (set of frames).

An attribute list consists of attributes with the following structure:

1. Name of attribute.

2. Text to prompt user for input value or the token '<INFERRED>' if the attribute is inferred.

3. A list of possible values, or a key word such as 'range:' followed by a range specification, or 'calc:' followed by an expression involving a calculation.

4. A '.' on a line by itself marks the end of an attribute.

An instance list consists of a set of frames with each frame having the following structure:

1. Name of frame.

2. A rule which specifies the conditions for a true instance of the object.

3. A '.' indicating the end of the frame.

The reason for calling these structures frames will be clarified when the syntax of frame structure is discussed.

TIE1 Knowledge Base Syntax

Attribute Syntax:

The syntax of attributes is designed for readability by both machines and users. A ';' in column 1 denotes a comment line. Blank lines, leading blanks, and tabs are ignored, which allows the user to use spacing and indentation to show logical structure. A list of indefinite length is terminated by a '.' on the next line. A ',' character is used to denote an 'OR' or to delimit a range, depending upon the context.

Attribute prompts begin and end with a '"' and can presently be five lines in length. Attribute names can be of any length with imbedded blanks. The characters '( ) + - / * % ,' are considered special and should not be used when an attribute is to be used in a calculation. Each possible value for an attribute must be on a separate line. Some examples of attributes and their syntax are listed below.

```
filter selection

        "What filter selection do you require ?"
        none
        red
        blue

.
exposure duration
        <INFERRED>
.
```

tape given

"What amount of tape do you have now ?
(in feet)"

range: 0, 500

tape remaining

"" This null prompt is used as a place holder only.

calc: tape given - (stop time - start time) * 10

Frame Syntax:

A frame consists of a frame name and a frame rule which describes a true instance of the object in terms of the object's local attributes. If the frame rule is found to be true then the frame name is returned as the value of the object. The frame rule has the syntax of attribute slots instead of the more traditional if-then syntax. Each attribute slot appears on a separate line with its attribute name on the left. To the right of the attribute name is a relational descriptor (such as eq, to be described in the next section), and to the right of the relational descriptor are one or more attribute values. Each line of the frame rule is connected with the following line by an implicit logical AND unless the previous line ends with a comma, in which case the lines are connected by an implicit logical OR. A single attribute name can be associated with a number of disjunctive values on the same line by separating the values with commas. Two examples of frames are listed below.

Tape dump (HAO) scheduled through TDRSS

instrument	eq	Tape dump (HAO)
start time	ra	TDRSS start, TDRSS stop
stop time	ra	TDRSS start, TDRSS stop
tape remaining	gt	0

Gamma ray spectrometer scheduled

instrument	eq	Gamma ray spectrometer
start time	ra	day start, day stop
stop time	ra	day start, day stop
start time	nr	SAA start, SAA stop
stop time	nr	SAA start, SAA stop

Attribute slots in frame rules may use several types of relational descriptors. The relational descriptors which are permissible are listed below.

eq	is equal to
ne	is not equal to
lt	is less than
le	is less than or equal to
gt	is greater than
ge	is greater than or equal to
ra	is in the range of
nr	is not in the range of

A frame, then, can be thought of as an inverted if-then rule (a then-if rule) with the value to be assigned to a goal equal to the frame name and the conditions for the assignment in the body of the frame. This syntax provides a more clearly structured rule than if more than one attribute per line were allowed. In addition, this syntax is also easy for the machine to read and requires no compilation.

There is no syntactical difference between variables, which can be names of integer attributes, and string attribute values. When an attribute is tested, it evaluates to an integer, if appropriate, otherwise it evaluates to its string name. Only integer numeric types may be used in the current version of TIE1.

The builder of a knowledge base is at liberty to choose a style which expresses the structure of the knowledge most clearly, as long as the basic syntax rules are adheared to. The use of comments is indicated by a semicolon in the first column and can add to the readability of the knowledge base, as is illustrated in the following example from the "animals world" domain, Winston and Horn [1984]:

```
;  knowledge base name

bird
;       default value:
                false

;       attributes:

        has feathers
                "Does the animal have feathers ?"
                yes
                no
        .
;  etc
        .
```

```
;       frames:

        true
                has feathers    eq      yes
                flies           eq      yes, no
                lays eggs       eq      yes
        .
        false
                has feathers    eq      no
                flies           eq      yes, no
                lays eggs       eq      yes, no

; etc
```

A unique feature of TIE1 is that rules are tested in a context-sensitive manner. When the value of an object is being sought, only those rules which are instances of the object being sought are considered at that time. For example, if the value of the object being inferred is represented by two frames (true and false) then just these two frames would be considered during that inference step. This feature not only increases the execution speed but also organizes the knowledge so that modifying the knowledge base and tracing the action of the inference engine is much easier.

The TIE1 Inferencing Method

The basic inferencing method used by TIE1 is a backward chaining, hypothesize and test search. Each relevant frame is tested in the order that it is listed in the knowledge base. Frames which are shown to be untrue are rejected. After all the attributes of the frames are tested according to the implicit rules that they capture, the name of the first frame which is true is returned. The capability of returning more than one value will be implemented if a need for this feature develops. If no frame is found to be true, the default value of the object is returned.

The search strategy TIE1 uses is to look for imbedded values first. If none are found, then TIE1 looks to see if the value is already known. If the value is still unknown, TIE1 checks to see if a calculation is required. If not, then TIE1 tries to infer the value from other knowledge. Finally, when all else has failed, the user is asked for a value.

The TIE1 Search Algorithm

TIE1's search algorithm is goal directed and uses a set of strategies to obtain the values of goals. The following is an outline of the algorithm:

1. Find the object associated with the goal's name.

2. Consider each attribute of the object to be a subgoal whose value is to be determined.

   Find and test the value of each attribute.

   a. Select the next attribute whose value is to be determined.

   b. Make this attribute the new goal to be obtained.

   c. Use the following strategies in the order they are listed to obtain the goal's value.

      i. If an imbedded value exists then use the imbedded value.

      ii. Otherwise, if the value is already known (on the facts blackboard) then use this value.

      iii. Otherwise, if a calculation is required then it is performed.

      iv. Otherwise, if the goal is of type inferred, then use the TIE1 search algorithm to obtain its value recursively.

      v. Otherwise, ask the user for the value.

   d. Write the attribute value on the facts blackboard.

   e. Use the obtained attribute value in the rules section of each frame and reject those frames whose rules are inconsistent with the value obtained.

   f. If all the frames have been rejected then set the goal's value to the default value of that object.

   g. Otherwise, if all the attributes have known values then set the goal's value to the name of the first nonrejected frame.

   h. Otherwise, if there are more attributes whose values are to be determined then find and test the values of the remaining attributes.

3. Report the obtained value of the goal.

The User Interface

TIE1's stand-alone user interface provides a window for displaying messages printed from the inference engine. When the system asks for the value of an attribute which is of character string type, the user can select one or more of the options listed by using the arrow keys and the return key. In the "learn" mode, TIE1 uses information in the attributes section to prompt the user and build new frames.

Summary of TIE1's Features

* TIE1 is written in C for portability.

* No compilation of the knowledge base is required before TIE1 can be used.

* TIE1 can be used in a stand-alone mode or in an imbedded mode.

* In the imbedded mode, the system automatically builds messages to explain why requests for resources are rejected.

* TIE1 has a learn mode which allows the user to easily add new frames to the knowledge base. TIE1 does this by using the attributes section to prompt the user for new instance frames.

* TIE1 has a trace option which allows the user to view the list of contending hypotheses as each new piece of data is added to the system.

* TIE1 has a minimum-question mode which allows a minimum amount of evidence to establish the truth or falsehood of a hypothesis. In this mode, the system does not ask an exhaustive set of questions.

* TIE1 has a debug mode which builds messages to explain why a particular hypothesis is rejected.

* TIE1 allows the user to redirect his input to a file so that the user is not asked for data. TIE1 then obtains the data from the file. This feature lets the user debug the imbedded mode without having to use the custom interface.

* TIE1 has its own user-friendly interface which is used in the stand-alone mode.

* The execution speed of TIE1 on a PC is about 10 times faster than commercially available inference engines written in the C language.

Future Features of the TIE1 System

Additional features will be implemented in the TIE1 system as needed. Some features under consideration are:

1. More Lengthy explanations.

2. Certainty factors.

3. The ability to return more than one value per goal.

4. Floating point arithmetic.

A Spacecraft Scheduling Application using TIE1: Interactive Experimenter Planning System (IEPS)

TIE1 is being used in this system scheduling spacecraft tape recorder and instrument events. Orbit data is read by the system and heuristics are used to indicate when particular events may be scheduled. Events can be scheduled interactively or in a batch (automatic) mode.

The main components of IEPS are described below:

1. The TIE1 system and its associated knowledge base which act as the constraint checker.

2. A batch scheduler with its associated knowledge base which is used in the automatic mode.

3. A user interface which displays resource information graphically or numerically.

4. Miscellaneous files:

   a. Input schedule file
   b. Output schedule file
   c. Configuration file
   d. Orbit data file
   e. Output messages file

The following two diagrams show the overall relationships between the components of IEPS. In both modes, the configuration file is read first by TIE1 to provide basic information required for the scheduling process. The main difference between the two modes is that the user and user interface are replaced by the batch scheduler when the batch option is selected.

Interactive Scheduler:

```
          Orbit Data File    User input       Constraint
                             |   |            Knowledge Base
                             |   |                |
                             V   V                V
Configuration File--->|---------------|---->|--------------|
                      |               |     |  Inference   |
Request Knowledge---->|User Interface |     |  Engine      |
        Base          |               |     |  (TIE1)      |
Input Schedule File-->|---------------|<----|--------------|
                              |
                              V
                        Output Schedule File
```

Batch Scheduler:

```
                     Orbit Data File          Constraint
                                |             Knowledge base
                                |                 |
                                V                 V
Configuration File--->|---------------|---->|--------------|
                      |               |     |  Inference   |
Request Knowledge---->|Batch Scheduler|     |  Engine      |
        Base          |               |     |  (TIE1)      |
Input Schedule File-->|---------------|<----|--------------|
                          |       |
                          V       V
      Output Schedule File      Output Message File
```

In the interactive mode, the user performs scheduling via the user interface and constraint checking is performed as user requests are made. The following is an example of a configuration file with the interactive mode selected:

```
Start date:
        850805

Number of days to be scheduled:
        4

Schedule mode (batch or interactive):
        interactive

Schedule request file:
        request.sch

Conflict knowledge base file:
        smm.pro
```

Orbit data file:
smm.orb

Input schedule file:
new.smm

Output schedule file:
smm.out

Output error file:
smm.err

Names of events to be read from orbit data file:
Daylight
South Atlantic Anomaly
Radio frequency interference
TDRSS viewing periods
GSVP (Ground station viewing periods)

In the batch mode, IEPS uses the request knowledge base to automatically generate a schedule. This knowledge base contains the knowledge of an expert scheduler. After a schedule is built in the the batch mode, the user may edit the schedule in the interactive mode. The batch scheduler has the following features:

1. The user can specify events to be scheduled by creating a file which is read and interpreted by the scheduling system.

2. The user can specify when the event is to be scheduled, the duration of the event, and the frequency of the event.

3. The user can also specify a set of alternative scheduling strategies to be tried in case the first event requested is denied because of scheduling constraints.

4. The scheduling strategies which are currently implemented are:

WHEN	name of event window within which to schedule
BIAS	bias start time by a given amount
AFTER	try scheduling after the conflicting event
DURATION	specify a new event duration
EWINDOW	use the entire window for the event

NEXT	use the next available window for the event
SQUEEZE	try to squeeze (move to one side) the conflicting event
BUMP	shift start and stop times of the event by a predefined amount

Structure of the Strategies Knowledge Base

The strategies knowledge base contains a list of events to be scheduled. These events have a priority implicitely determined by the order in which they are listed. Each event request has the following structure:

1. Name of event to be scheduled.
2. Frequency of event to be scheduled (every nth window).
3. Duration of event.
5. A list of strategies for scheduling the event.

The user may also specify a list of reserved time slots for any event or resource.

Syntax of Strategies Knowledge Base

The syntax of the strategies knowledge base is similar to TIE1's knowledge base syntax. First, the name of the event to be scheduled is stated. Then, the frequency of the event to be scheduled is indicated by a positive integer and its duration is specified in hours and minutes. Next, strategies are indicated by a list of key words which specify the actions to be taken during the scheduling process. Some of these key words and the arguments they can take are shown in the example which follows:

```
Tape dump (HAO)
;       frequency of event:
                4
;       original duration request:
                0:15
;       strategies:
                WHEN TDRSS viewing periods
                BIAS 0:10
                DURATION 0:10
                AFTER
                WHEN GSVP (Ground station viewing periods)
                BUMP 0:20
                NEXT
```

The Strategies Interpreter

Scheduling strategies are tried in the order they are specified until the event is successfully scheduled or the list is exhausted. In this respect, IEPS can be thought of as "opportunistic". The structure of the constraint knowledge base is such that there are frames which contain constraint information for each event being scheduled. If all of the constaints are satisfied, then the event is scheduled. A log is kept of the messages returned from the constraint checker and of the strategies tried by the interpreter. This log can then be examined and analyzed for refinement of the strategies.

It is interesting to note that the system works somewhat like a production system with the if and then parts organized into separate knowledge bases. The strategies interpreter, being the then part, forward chains through its request and strategies list while the constraint checker, being the if part, backward chains through its object hierarchy.

Conclusions

- During the three months that TIE1 has been used for evaluation, its ease of use and maintence, its extensibility, and its execution speed have all surpassed expectations. These results and the fact that the IEPS prototype uses "real" orbit data to generate a schedule demonstrate that TIE1 is already powerful enough to be of use in solving "real-world" problems.

The portability of IEPS (and TIE1) has been demonstrated in a number of ways. TIE1 was originally developed on an IBM PC-XT. It was then ported to an Integrated Solutions MC68020-based workstation. This porting task took less than one week. Most of this time was spent becoming familiar with the workstation's cursor package. Because the user interface is completely separate from the inference engine it should be possible to easily port TIE1 to any environment which has a C compiler.

Acknowledgements

The author would like to thank Ronald Littlefield and David Beyer of Bendix Field Engineering Corporation for their input and assistance in the preparation of this paper. Jay Johnson, also from Bendix, wrote the calculator portion of TIE1. This work was supported by NASA contract NAS5-27772.

References

Brundisk, F., J. Dumer, T. Hanratty, P. Tananbaum, "GENIE: An Inference Engine With Diverse Applications", Proc. IEEE Second Conference on Artificial Intelligence Applications,1985.

Daniel, L., in T. O'Shea and M. Eisenstatdt (ed.), Artificial Intelligence Tools, Techniques, and Applications, "Planning and Operations Research", Harper & Row Pub., N.Y. pp. 423-452,1984.

Hankins, G. B., J. W. Jordan, J. L. Katz, A. M. Mulvehill, J. N. Dumoulin, J. Ragusa, "Expert Mission Planning and Replanning Scheduling", Proc. IEEE 1985 Expert Systems in Government Symposium, pp. 43-49,1985.

Harmon, P., D. King, Expert Systems: Artificial Intelligence In Business, John Wiley & Sons, Inc.,1985.

Kerninghan, B. W., D. M. Ritchie, The C Programming Language, Prentice-Hall Inc., 1978.

Lee, N. S., J. W. Roach, "GUESS/1: A General Purpose Expert System Shell", Porc. IEEE Expert systems in Government Symposium, pp. 275-283,1985.

Marsh, A. K. (ed.), Advanced Military Computing Including Artificial Intelligence, Pasha Pub. Inc., Vol 1, #5,1985.

Szolovits, P., "Expert System Tools and Techniques: Past, Present, and Future", Proc. Artificial Intelligence: Current Applications, Trends and Future Opportunities, 1986.

Tassle, R. V., "Languages Then and Now", The DEC Professional, Professional Press, Inc.,1985.

Waterman, D. A., A Guide to Expert Systems, Addison-Wesley Pub. Co.,1986.

Winston, P. H., Artificial Intelligence (Second Edition), Addison-Wesley Pub. Co.,1984.

Winston, P. H., B. K. Horn, LISP (second editon), Addison-Wesley Pub. Co.,1984.

S7-63
233137
15P.

A Prototype Expert System in OPS5 for Data Error Detection

James Rash

NASA Goddard Space Flight Center
Telecommunication Systems Branch, Code 531
Greenbelt, Maryland

ABSTRACT

A prototype expert system has been developed in the OPS5 language to perform error checking on data which spacecraft builders/users supply to the NASA Goddard Space Flight Center for processing on the Communications Link Analysis and Simulation System (CLASS) computer. This prototype expert system, called Trajectory Preprocessing System (TRAPS), contains 49 rules and at present runs on an IBM PC in the OPS5+ software package from Artelligence, Inc. In its operational phase, TRAPS will run in the Oak Ridge Production Language (ORPL) on the CLASS computer (a Perkin-Elmer 3244 supermini). ORPL, an implementation of OPS5 by the Oak Ridge National Laboratory in MULTIFORTH on a Hewlett-Packard 9836 desktop computer, is now being ported to SS-FORTH on the CLASS computer. This paper discusses the expert system problem domain, development approach, tools, results and future plans stemming from the TRAPS project.

INTRODUCTION

CLASS

The Communications Link Analysis and Simulation System (CLASS) comprises several major software systems (written in FORTRAN) designed to analyze space and ground communications system performance [Godfrey, 1983 and 1984]. CLASS was developed by NASA primarily for the prediction of user spacecraft communications system performance through the Tracking and Data Relay Satellite System (TDRSS) (the term "user spacecraft" or "user vehicle" is used to refer to a spacecraft which is utilizing services provided by TDRSS). CLASS capabilities are not limited to TDRSS. In fact, CLASS has been applied to a broad range of problems in the design and evaluation of ground and space communications networks and systems. End-to-end as well as system-by-system analysis can be accomplished through CLASS. CLASS is

designed for convenient, reliable use by personnel with a wide variety of backgrounds in telecommunications systems.

CLASS Functions

The basic functions facilitated by CLASS are: (1) communications system design, (2) communications system performance analysis, evaluation, and prediction, (3) spacecraft mission planning, and (4) post-launch trouble shooting of communications problems. Relevant environmental factors such as interference from the sun and extraneous radio frequency (RF) sources can be included in the analyses and simulations.

CLASS Computer

The CLASS computer, located at NASA's Goddard Space Flight Center (GSFC) in Greenbelt, Maryland, is a Perkin-Elmer 3244 supermini with 16 megabytes of RAM, serving up to 32 simultaneous users locally or in distant cities. It will soon be upgraded to a Perkin-Elmer 3280 with multiple processors.

CLASS Users

Users of CLASS include spacecraft builders and designers who must establish that their spacecraft communications systems meet prescribed requirements. Other CLASS users include the scientific investigators and teams of spacecraft operations personnel who plan and direct specific mission activities for their spacecraft. In addition, NASA analysts utilize CLASS for analysis and simulation of communications links between NASA ground stations and the Tracking and Data Relay Satellite (TDRS), and in trouble shooting user spacecraft communications problems.

Need for Expert System Capabilities

Making effective use of the numerous, complex CLASS capabilities has been very demanding upon the CLASS users. During the past two or three years it has been realized that the users needed intelligent assistance from the CLASS software as they prepared and conducted their runs. Thus arose the concept of a CLASS control program to provide intelligent assistance and a higher level of internal management of software modules during runs. It was concluded that expert systems would be needed to provide such capabilities.

Further, certain other types of problems within the scope of CLASS gave impetus to develop expert system capabilities for CLASS. First, there was frequently a need to do fault diagnosis on the communications links to a TDRS or user spacecraft. This was recognized as a

classic problem domain for expert systems. Secondly, there existed in CLASS a need to check user-supplied input data for certain kinds of errors. An expert system which provides the capability to check user-supplied input data for errors will be the focus of this paper.

It should be noted that the term "expert system" is used somewhat loosely in this paper to mean any rule-based system whether "expert" or not.

SELECTION OF DEVELOPMENT/IMPLEMENTATION TOOLS

Basic Constraints

The basic constraints on expert system software for CLASS are that it must be imbedded within the CLASS environment, interface efficiently with existing CLASS programs and run in a multiple-user mode. An additional consideration was that of a limited budget.

Alternatives

Following the initiation a year ago of efforts intended to bring expert system technology into the CLASS environment, it was found that supported software for expert system development on the Perkin-Elmer 3200 series computers was not available. Nonetheless, there remained a need to develop expert system capabilities for CLASS. The situation was analyzed in terms of three general categories of alternatives:

A. High-cost approach.

- -- Sophisticated development system.
  - -- LISP workstation (Symbolics, LMI, etc.).
  - -- Expert system builder package (ART, KEE, etc.).
- -- Advantages: more efficient expert systems, more rapid prototyping, greater productivity, good programming and applications support within NASA and from vendors.
- -- Disadvantages: high initial cost for hardware, software, and training; unfeasibility of direct use of results in CLASS (LISP workstations are single-user systems, whereas CLASS is a multiple concurrent-user environment); unfeasibility of porting results to CLASS (general purpose

computer, no LISP).

B. Medium cost approach.

-- Contracted porting of an existing expert system tool to CLASS (e.g., KES II, RULEMASTER).

-- Advantages: resulting system would be embedded in CLASS, providing multi-user environment; possible local vendor support (KES II).

-- Disadvantages: Quality, suitability, and final cost of the tool were indeterminate; date of availability likely to be unacceptable; doubtful availability of programming and applications support within NASA with regard to such a tool.

C. Low cost approach.

-- Use OPS5 as a tool with which to begin development of expert systems.

-- Advantages: OPS5 is one of the most widely used high-level expert system languages; availability of support within NASA; availability of low cost versions of OPS5 for the IBM PC and (in a beta test version) for the then-existing CLASS workstations; possibility of implementing OPS5 on the CLASS mainframe.

-- Disadvantages: Lower productivity because OPS5 is less sophisticated than high-priced tools; time required to implement OPS5 on CLASS mainframe.

Alternative Selection

The first alternative, the high-cost approach, was eliminated for two reasons: (1) high cost and (2) incompatibility with the basic requirement for embedded, multi-user operation in CLASS.

Cost was also considered to be a disadvantage in the case of the medium-cost approach. In addition, the medium-cost approach involved such inherent uncertainties and probable delays that it was not acceptable.

The final decision was to adopt a low-cost, low-risk approach using OPS5 as a start-up language for developing expert system applications in CLASS.

INITIAL PROBLEM SELECTION

After consideration of several possible applications of expert systems for CLASS, it was decided that an expert system to preprocess user trajectory data would be constructed. One of the considerations in choosing this problem was that it seemed to be a simple, straightforward problem for demonstrating the approach to be used.

Another consideration centered on the value of a comparison of the OPS5 language with FORTRAN (the language of the existing CLASS applications). The problem selected was one which was known to be solvable with FORTRAN and was believed to be solvable with OPS5. The main factors to be compared were processing efficiency and program maintainability. It was projected that OPS5 would provide adequate processing efficiency for the problem selected, as well as a significant advantage in program maintainability. Program maintainability appeared to be particularly important in this problem domain because of the many possible variations in mission and spacecraft data requirements, which would necessitate development and maintenance of many code modules.

TRAPS PROBLEM DOMAIN

Analysis of Trajectories

User spacecraft missions are considered to fall into two categories: planetary probes and earth orbital missions. Typical CLASS analyses of a planetary mission, such as Centaur Galileo, may involve processing several hundred possible trajectories as specified by the CLASS user. Each trajectory will have 150 to 250 time points starting at the time of release of the spacecraft from the Space Shuttle and ending at the time communications with TDRSS are lost following the completion of the engine burn phase. Perhaps 2 to 3 hours total time is covered by the analysis for each such trajectory.

Analyses for orbital missions are essentially the same but may involve much longer time periods as well as special factors such as earth occultation and interference from earth multipath reflections.

Requirements on User Input Data

Users seeking valid results from CLASS are asked to furnish input data in a standard format on magnetic tape. For various reasons, such as the unavailability of computer programming resources within the user's organization, the data tapes provided to CLASS may be formatted improperly. Once the data are in the required format, there are still a number of specific rules which the data must satisfy. Some of these rules are mission-

unique, and the rest are generic. However, in all cases the rules are derived from the logic of space communications rather than from the internal requirements of CLASS programs.

Consequences of Bad Input Data

Until recently there did not exist a practical means to determine whether the user-supplied input data fulfilled the basic requirements (indicated above) prior to the initiation of actual analysis and simulation runs in CLASS. Consequently, if input data errors occurred a considerable amount of CLASS computer time and personnel time might be wasted on runs which could only produce invalid results if they completed at all. Recently, for example, a data record was inadvertently repeated in the incoming data causing two records to have the same value for the ground elapsed time. This error, if encountered during automatic overnight processing on the CLASS computer, could have resulted in the loss of more than twelve hours of computer time and caused the same amount of delay in completing the runs.

DESCRIPTION OF TRAPS

CLASS is now being given the capability to preprocess the user-supplied input data. This is the function of the Trajectory Preprocessing System (TRAPS), an expert system designed to protect CLASS from bad input by (1) recognizing bad data (usually) before the start of processing by the analysis and simulation programs, and (2) reporting the data errors to the CLASS analysts.

Input Data

TRAPS is designed to check the input data elements listed in Table 1. Each data record is identified by its unique ground elapsed time (GET) tag.

Figure 1 displays the first few records of a typical data file processed by TRAPS. The data in this file were extracted from a user-supplied input data tape for the Centaur Ulysses mission. Each record consists of ten data elements as listed in Table 1.

TRAPS Rule Base

The initial requirements ("human rules") on which the TRAPS expert system prototype was based are given in Table 2. Each of the six "human rules" was translated into from one to eight OPS5 rules for processing the data records, plus from one to five additional OPS5 rules for generating the TRAPS output. With the rules for initialization and

TABLE 1.
Input Data elements checked by TRAPS.

Mission ground elapsed time (GET)
Flag indicating user spacecraft communications link 3 is active
Link ID for link 3 (if active)
Code for the TDRS supporting link 3 (if active)
Code indicating a user antenna switch event for link 3 (if active)
Flag indicating user spacecraft communications link 4 is active
Link ID for link 4 (if active)
Code for the TDRS supporting link 4 (if active)
User spacecraft configuration code (indicating various events such as beginning or end of a slow spin period)
Burn flag indicating engine firing

0	1	LOW_RATE__	E	___	0	_____	_	RCSI_____	0
16	1	LOW_RATE__	E	ASW	0	_____	_	RCSI_____	0
136	1	LOW_RATE__	E	___	0	_____	_	RCSI_____	0
256	1	LOW_RATE__	E	___	0	_____	_	RCSI_____	0
376	1	LOW_RATE__	E	___	0	_____	_	SUNA_____	0
377	1	LOW_RATE__	E	___	0	_____	_	SUNA_____	0
496	1	LOW_RATE__	E	___	0	_____	_	SUNA_____	0
600	1	LOW_RATE__	E	___	0	_____	_	SUNA_____	0
736	1	LOW_RATE__	E	___	0	_____	_	SUNA_____	0
796	1	LOW_RATE__	E	ASW	0	_____	_	SUNA_____	0
916	1	LOW_RATE__	E	___	0	_____	_	SUNA_____	0
1036	1	LOW_RATE__	E	___	0	_____	_	SUNA_____	0
1096	1	LOW_RATE__	E	ASW	0	_____	_	SUNA_____	0
1216	1	LOW_RATE__	E	___	0	_____	_	SUNA_____	0
1336	1	LOW_RATE__	E	___	0	_____	_	SUNA_____	0
1396	1	LOW_RATE__	E	ASW	0	_____	_	SUNA_____	0
1516	1	LOW_RATE__	E	___	0	_____	_	SUNA_____	0
1576	1	LOW_RATE__	E	___	0	_____	_	SUNA_____	0
1696	0	_____	_	___	0	_____	_	SUNA_____	0

Figure 1. Records From a Typical TRAPS Input Data File.

for reading the input data, the total number of OPS5 rules in the TRAPS prototype (third version) is 49.

For convenience in debugging and maintenance, all the OPS5 rules derived from any one "human" rule were collected into one logical unit. Each such logical unit of OPS5 rules is saved in a separate file.

As an example of the result of transforming human rules into OPS5 rules, Figure 2 shows a listing of the OPS5 rules corresponding to human rule #2 in Table 2. The first OPS5 rule in Figure 2 is the direct expression of human rule #2 and fires only while records are being processed ("phase-process-a-record"). The second and third OPS5 rules in Figure 2 control the printing of messages following detection of a violation of the human rule, and fire only during "phase-printout" after all the data records have been processed.

At the present time, the expert system produces messages indicating any errors found. TRAPS does not (yet) have the capability of correcting or modifying the input data in any way.

TABLE 2.
"Human rules" on which TRAPS is based

(1)	During a slow spin period a user antenna switch must occur at least once in every 12 minute interval, but not more than once in any 5 minute interval.
(2)	During any trajectory there must occur at least one switch of the supporting TDRS, but there must not be more than 5 such switches.
(3)	There must exist a burn flag between 30 and 60 minutes ground elapsed time.
(4)	For any user communication link which is active, the link ID must be either LOW RATE or HIGH RATE.
(5)	Every valid user vehicle configuration code must appear at least once in each trajectory, and every configuration code in the input data must be on the list of valid codes for that particular mission.
(6)	The GETs must appear in strictly ascending order in the input data for any trajectory.

```
; Rules to check for the proper number of switches
;  of TDRS support.

(p link-support-3-switch-counting
   (phase-process-a-record)
   { (current-support ^current-support-code <cs>
                      ^switch-count <k> ) <c> }
   (record ^link-support-3 { <s> <> <cs> }
           ^link-3 <> 0 )
  -->
   (modify <c> ^current-support-code <s>
               ^switch-count (compute <k> + 1) ))

(p error-link-support-3-too-low-printing
              ; mustn't fire til end
   (phase-printout)
   (current-support ^switch-count < 1 )
  -->
   (write (crlf) (crlf)
    |File contains no TDRS link-3 support switch.| ))

(p error-link-support-3-too-high-printing
              ; must not fire til end
   (phase-printout)
   (current-support ^switch-count > 5 )
  -->
   (write (crlf) (crlf)
    |File contains more than five TDRS
    |link-3 support switches.| ))
```

Figure 2. OPS5 Rules Corresponding to "Human Rule" #2.

EVALUATION OF THE PROTOTYPE

The TRAPS prototype has been tested sufficiently to establish that its operation is consistent with its design. It is capable of finding real errors in trajectory data input to CLASS. The TRAPS prototype will be used as a model for the development of expert systems for other CLASS applications.

Efficiency

The first version of TRAPS (based on the first five rules in Table 1) read in all records from the input file before the data-checking rules in the knowledge base were allowed to begin firing. This version was very inefficient because as the number of data records increased, the size of OPS5 working memory rapidly increased and the processing speed rapidly decreased.

The second version of TRAPS (also based on the first five rules in Table 1) used a record-by-record processing approach in which each record was processed against the data-checking rules before the next record was read. This record-by-record processing minimized the number of OPS5 working memory elements regardless of the number of records to be processed. The second version of TRAPS processed records approximately 14 times faster than the first, but had some inadequacies.

The third version of TRAPS was derived from the second version by adding a module of OPS5 rules corresponding to rule #6 in Table 1 and by revising some of the rules to improve readability and to handle special cases. These enhancements caused the third version to run somewhat slower than the second version.

Table 3 summarizes the processing efficiency of the three versions on the IBM PC, based on processing a data file containing 163 data records using OPS5+ Version 2.0003. The values for processing speed are not highly significant since they are data dependent and sensitive to the structuring of the condition elements within the rules [Brownston, 1985].

TABLE 3.
TRAPS Processing Efficiency

Version	No. of Rules	No. of Rule Firings	Processing Speed	
			Records/second	Rules/second
1	41	397	0.11	0.26
2	40	849	1.6	8.1
3	49	862	1.3	6.8

Maintainability

An important feature of OPS5 code is that literals (corresponding to variable names in other languages) can have virtually any length. This feature of OPS5 was utilized to fully satisfy the requirement that the code for TRAPS be easily maintainable by being easily read.

Future enhanced versions of TRAPS can be developed readily by simply including one or more additional code modules derived from new human rules. The ability of OPS5 to accomodate this modular approach further ensures maintainability.

Comparison with FORTRAN

To facilitate the goal of comparing OPS5 and FORTRAN, a FORTRAN program was written to perform the functions of the first five rules of Table 2. Figure 3 shows a listing of the subroutine from this FORTRAN program corresponding to the second human rule (Table 2). A comparison of this FORTRAN subroutine with the equivalent OPS5 code in Figure 2 shows that OPS5 is more readable than FORTRAN, as has been noted by others.

In comparing Figures 2 and 3, it would seem that OPS5 is more terse than FORTRAN. However, this is often not the case since a number of functions, especially input/output functions and general numerical computations, are cumbersome in OPS5 [Brownston, 1985]. The FORTRAN code developed for the TRAPS application was about 30% shorter than the OPS5 code because lengthy literals were deliberately used in the OPS5 program to enhance readability.

Based on the experience of developing three different versions of this prototype expert system as well as a more or less equivalent FORTRAN program, it is the author's opinion that OPS5 offers a considerably higher degree of maintainability than does FORTRAN.

However, it should be stated that developing OPS5 programs is not without pitfalls. Subtle interactions can occur between the rules in a production system program during interpretation by the inference engine. These interactions can be difficult to predict and tricky to debug [Brownston, 1985]. Programming in OPS5 requires carefulness and skill, perhaps beyond that required for procedural languages such as FORTRAN.

EVALUATION OF THE DEVELOPMENT ENVIRONMENT

As far as the prototyping environment is concerned, OPS5+ on the IBM PC served fairly well. In the "develop" mode, OPS5+ provides a window and mouse menu interface. However, the windows on the IBM PC are not the same as windows on a LISP workstation such as the Symbolics machine. The OPS5+ windows are simple divisions of the screen into fixed areas or panels which display either menus or results of program execution. In the mouse menu panel, OPS5+ achieves the functional equivalent of the LISP machine windows by employing an overlay technique.

Any expert system development tool is enhanced by a good editor. OPS5+ has a built-in editor which was not used during TRAPS development -- instead, WordStar was used under an OPS5+ feature which permits the user to pause the execution of OPS5+, edit, and then re-enter

```
******Subroutine TTWO checks for failure on rule #2
*      NSWT: Count number of TDRS support switches
*      MSUP: Index value for SUPPORT
*      MCODE: SUPPORT code
*      SUPL: Last SUPPORT
*      MFLAG: Flag for SUPPORT
*****************************************************
*
      SUBROUTINE TTWO(ITEST, NSWT, I, NPTS)
      DIMENSION ITEST (6)
      CHARACTER * 1   SUPL, MCODE(4)
$INCLUDE COMMONA.FTN
      DATA MCODE  /'E','W','S',' '/, MSUP /3/
      IF (I .EQ. 1) THEN
         IF (LINK(MSUP,I) .EQ. 1) THEN
            MFLAG=0
            DO 10  L=1, 4
              IF(SUPPORT(MSUP,I) .EQ. MCODE(L))MFLAG=1
10          CONTINUE
            IF (MFLAG .EQ. 0)WRITE(1,300)
     +          SUPPORT(MSUP,I),I
            SUPL=SUPPORT (MSUP,I)
         ENDIF
      ELSE
            IF (LINK(MSUP,I) .EQ. 1)THEN
               MFLAG=0
               DO 20  L=1, 4
                IF(SUPPORT(MSUP,I) .EQ. MCODE(L))MFLAG=1
20             CONTINUE
               IF (MFLAG .EQ. 0)WRITE(1,300)
                  SUPPORT(MSUP,I),I
                IF (SUPPORT(MSUP,I) .NE. SUPL)THEN
                  NSWT=NSWT+1
                  SUPL=SUPPORT(MSUP,I)
                ENDIF
               ENDIF
            IF (I .GE. NPTS) THEN
                IF (NSWT .EQ. 0)WRITE (1,100)
                IF (NSWT .GT. 5)WRITE (1,200)
           ENDIF
      ENDIF
      RETURN
100   FORMAT(' ERROR#2: NO SWITCH AFTER CHECKING '
     +       'ALL THE RECORDS')
200   FORMAT(' ERROR#2: MORE THAN 5 SWITCHES AFTER '
     +       'CHECKING ALL THE RECORDS')
300   FORMAT(' ERROR#2: ILLEGAL SUPPORT ID:',A25,
     +       'RECORD#',I5)
      END
```

Figure 3. FORTRAN Subroutine Corresponding to "Human Rule" #2

OPS5+ at the pause point. Each pause puts the user at the PC's operating system level, where the user may explicitly execute the editor/word processor of his or her choice. All of this is functional, if tedious and time-consuming, especially when using floppy drives.

The drawbacks of OPS5+ as a development tool probably have more to do with the hardware than with the software. First, since the IBM PC display is not bit-mapped it does not facilitate true windowing, and is thereby limited in its ability to display enough information. Second, the IBM PC does not have enough raw speed, especially when using floppy disk drives (this problem is essentially solved by using a hard disk on an IBM PC/AT).

From the TRAPS development effort, OPS5+ received a positive evaluation overall. It was found to be an effective and essentially bug-free tool with a favorable price-performance ratio.

CURRENT WORK

ORPL, the Oak Ridge National Laboratory implementation of the OPS5 language in FORTH [Dress, 1986], is currently being ported onto the CLASS mainframe. This porting effort started in March, 1986 after a brief period was spent becoming familiar with the SS-FORTH package (a direct-executing version of FORTH [Pawley, 1984]) recently acquired for the CLASS mainframe.

FUTURE PLANS

When ORPL becomes available for use on the CLASS mainframe, it will be validated by running the TRAPS code as well as the code for other expert systems written in OPS5 which may be available from other AI groups within NASA (or elsewhere). Benchmarking of ORPL against other implementations of OPS5 is also planned.

Beyond the completion of the porting of ORPL, future efforts are expected to divide into two areas: (1) other CLASS expert system applications, and (2) enhancements to ORPL.

Some of the applications within CLASS which are candidates for solution via expert systems are:

(a) Communications link fault diagnosis
(b) User spacecraft communications system design
(c) CLASS "super executive" control program

Efforts to enhance ORPL may include an object-oriented knowledge representation paradigm and user-friendly features such as color graphics, a mouse interface, and window-based editing. Another enhancement to ORPL that will be important for some applications is

the capability to incorporate or directly utilize routines written in FORTRAN.

CONCLUSION

Preprocessing incoming data to verify its acceptability prior to its use in critical or resource-intensive processing is essential in many computer applications. This is the case in the CLASS environment and has led to the development of the TRAPS prototype expert system. This project has shown that an input data preprocessor developed as a rule-based system in the OPS5 language is a viable alternative to programs in traditional languages. It was shown that OPS5+ on an IBM PC ran fast enough to satisfy CLASS requirements for processing efficiency. This suggests that the application will also run fast enough when ported to the CLASS mainframe. In addition, this project has shown that OPS5 offers improved program maintainability in comparison with FORTRAN. Based on these results, development of additional expert systems for CLASS using the Oak Ridge Production Language (ORPL) is planned.

ACKNOWLEDGEMENTS

Robert Brown and Robert Savely of the Johnson Space Center (JSC) are thanked for making possible the technology transfer project between NASA/GSFC and NASA/JSC that initiated the development of TRAPS. Liu Wang of JSC is especially thanked for his participation in the joint coding of the first two versions of TRAPS in OPS5. Dr. Ronald Littlefield of Allied Bendix Aerospace Corporation is thanked for his helpful suggestions in the writing of this paper.

REFERENCES

1. Godfrey, R. D., "The Communications Link Analysis and Simulation System (CLASS)," NASA Conference Publication 2342, Part 2, Session 8, Space Shuttle Technical Conference, Johnson Space Center, Houston, Texas, June 28-30, 1983.

2. ---, "Space Communications Performance Prediction Utilizing the Communications Link Analysis and Simulation System (CLASS)," CLASS 101, Goddard Space Flight Center, Greenbelt, Maryland, July 1984.

3. Brownston, L., Farrell, R., Kant, E., and Martin, N., Programming Expert Systems in OPS5, An Introduction to Rule-Based Programming, Addison-

Wesley Publishing Company, Inc., 1985.

4. Dress, W. B., "AI Goes FORTH", Proceedings, 1986 Conference on AI Applications, Goddard Space Flight Center, Greenbelt, Maryland, May 15, 1986.

5. Pawley, W. E., "Using Native Machine Code Analogs of Interpreted FORTH's Elements", Proceedings, 1984 Rochester Forth Conference, published by the Institute for Applied Forth Research, Inc., Rochester, New York, 1984.

58-63
233138
8P.

MULTIPERSPECTIVE ANALYSIS AND TESTING OF EXPERT SYSTEMS

Terry B. Bollinger
Bendix, Systems Engineering Office
10210 Greenbelt Road, Suite 200
Seabrook, Maryland 20706

Phone: (301) 731-1173

ABSTRACT -- The paper describes a technique which the author developed for testing expert systems. The technique, which he calls multiperspective testing, can be applied during both the knowledge engineering phase and the acceptance testing phase of developing an expert system. The first step in multiperspective testing is to define a group of performance measures ("perspectives") that focus on the behavior of the knowledge base. For each such measure, the results of testing are summarized in four scores, which the author calls "expansion," "detection," "discrimination," and "comprehesion." These scores have the advantage of providing more specific information about how the knowledge base should be updated or corrected.

1. INTRODUCTION

A major constraint in building and delivering a reliable expert system is the need for testing techniques that produce detailed, quantified answers about the quality and accuracy of its knowledge base. Such answers are not always easy to come by. Test results can be difficult to interpret, and the relationship between test results and the knowledge base itself may be obscure.

The purpose of this paper is to describe a technique called multiperspective testing, which the author developed while trying to organize testing of a prototype expert system. In multiperspective testing, the first step is to define a group of performance measures ("perspectives") that focus on the behavior of the knowledge base to be tested. For each performance measure, test results are summarized in four scores called "expansion," "detection," "discrimination," and "comprehesion." These scores, which were developed by the author, have the advantage of providing more information about how the knowledge base should be updated or corrected in response to test results.

2. MAJOR CONCEPTS

The major steps in performing a multiperspective test are:

- Definition of customized performance measures.
- Actual performance of tests.
- Calculation of the "perspective" scores.
- Comparison of scores to an "acceptance thresholds" table that defines acceptable performance levels.

These steps are described in the paragraphs below.

2.1 DEFINING THE PERFORMANCE MEASURES

To use multiperspective testing, an expert system must first be assigned a set of unique, customized performance measures called perspectives. The term "perspective" here refers to evaluation of an expert system in much the same fashion that a panel of specialists (vs. a single expert) might evaluate it, with each member the panel judging the the behavior of the system from his own "perspective," and ignoring results that are outside of his area of specialization.

Multiperspective testing uses a similar divide-and-conquer approach, although without using an actual panel of specialists. Multiple results are instead derived from one expert who "shares" his time among several previously-defined specialty areas. The specialty areas, or perspectives, may vary widely in complexity and difficulty, ranging from "comprehensive" perspectives that look at large segments of the total test results, to "specialized" perspectives that look only at a narrowly defined data sets.

Individual perspectives need not be complex or difficult to define. For example, assume that the purpose of an expert system is to provide advice on how to handle power or cooling losses in a multi-building complex. In this case, the structure of the problem domain (the multi-building complex) could be used to define several comprehensive perspectives. One group of perspectives could be defined to look at test results for each building of the complex; another pair of perspectives could look at the two major subsystems (power and cooling) of the complex. More specialized perspectives might look at responses concerning a specific type of equipment; perspectives of this type could be useful if there were unique, critical requirements for such equipment.

Another way to define testing perspectives is to look at known intermediate steps in how the system arrives at a final result. For example, if the purpose of an expert system were to locate equipment failures in a large computer network, a perspective could be defined that would look at whether the expert system can detect the presence of an equipment failure, regardless of whether the expert system can actually determine the location of the failure. Using known intermediate steps to define perspectives can be particularly useful for determining whether "good" final results are based on dubious intermediate conclusions.

Creating a complete, effective set of testing perspectives is not a trivial task; as a minimum, it requires a good understanding of both the problem domain and the functional requirements of the expert system. However, once defined, such performance measures can be helpful both in organizing the testing activity, and in steering the overall direction of the knowledge engineering effort.

2.2 PERFORMING THE TEST

Multiperspective testing assumes that a reasonably large set of realistic test data is independently available. How to create such a data set is itself a complex task, one that is beyond the scope of this paper.

While running test data, the test conductor will often need to note results in two or more of the performance measure categories. Although it may be possible to extract data "after the fact" from a single detailed set of test notes, it is generally better for the test conductor to have forms prepared for noting multiple results during testing.

The test conductor also needs to be aware of the special scoring methods used in multiperspective testing, which are described in the next section.

2.3 SCORING

Figure 1 shows a graphical representation of a 3-perspective test, in which each of the dimensions corresponds to a distinct performance measure. The volume labeled with an alpha represents the set of all conclusions reached by an accepted authority (the expert) for the test data; the overlapping space, which is labeled with a beta, represents the set of conclusions reached by the expert system. Perpendicular to each of the projection axis is a surface onto which the alpha and beta sets are "projected," or viewed through one of the perspective measures. The results are two (generally overlapping) subsets of the original test results, shown as the Venn diagrams labeled A and B.

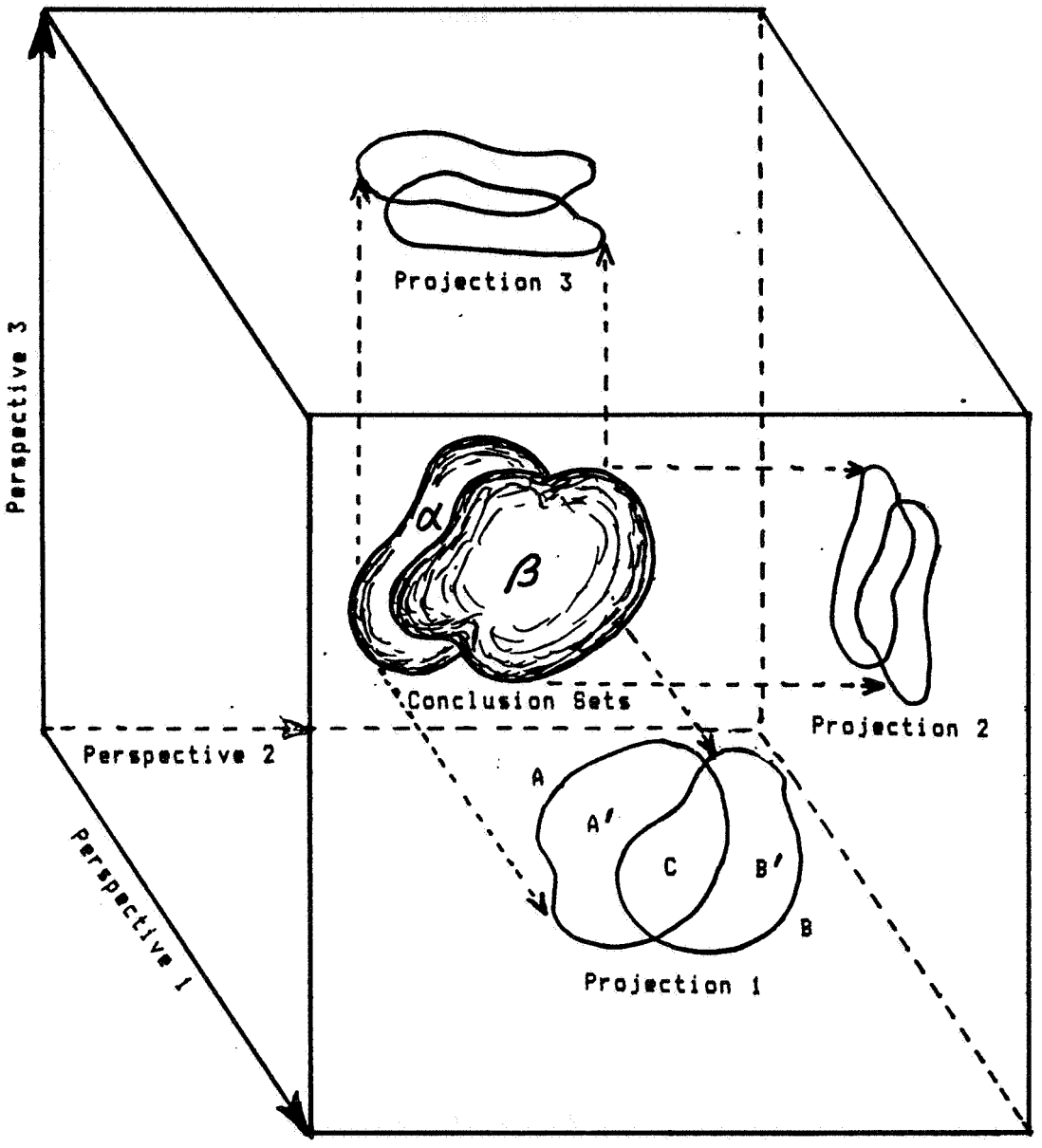


FIGURE 1. Graphical Representation of a 3-Perspective Test

For each projection, the following Venn diagram regions are defined:

A -- The set of results projected from the alpha (accepted authority) conclusion set. (Note that in some cases, results which the authority accepts as valid may be uncovered by the expert system during testing. For consistency, such results should be added to the alpha and A sets before the perspective scores are calculated.)

A' -- The set of all members of A which do not overlap with B (A - B , where "-" is set subtraction)

B -- The set of results projected from the beta (expert system) conclusion set.

B' -- The set of all members of B which do not overlap with A (B - A , where "-" is set subtraction)

C -- The intersection of A and B (that is, the set of "correct answers" produced by the expert system).

The four "scoring values" for a perspective can now be defined in terms of the Venn diagram regions:

a. EXPANSION -- The expansion score E is an optional rating that is used to explicitly rate the expert system for finding correct answers of which the "accepted authority" was unaware. The expansion score gives the percentage of increase in the number of the accepted correct answers (for that perspective) that is directly due to the detection of such answers by the expert system during testing. (Note that discrimination and detection ratings are always calculated using the expanded set of correct answers, to insure that detection and discrimination ratings remain less than 100%)

The expansion score is defined below. (Note that paired vertical bars simply indicate that the number of items in the set should be counted and the result used as the value of the parameter. Also, note that Aold refers to the A set that was used before the new results were added to it.)

$$E = \frac{100 * (|A| - |Aold|)}{|Aold|}$$

b. DETECTION -- The detection score D is a percentage measure of the ability of an expert system to find correct answers, regardless of how many wrong answers are also produced. The detection score is defined as follows:

$$D = \frac{100 * |C|}{|A|}$$

It is not necessarily difficult for an expert system to achieve a detection rating near 100%, since this score does not take the number of wrong answers into account; in many cases, a system could do fairly well by simply "overreacting" to its inputs and producing large quantities of invalid conclusions, with a relatively small proportion of valid conclusions mixed in. It is the combination of a high score in detection with a high score in discrimination (see below) that is difficult to achieve, and which makes the detection score more indicative of a good knowledge base. A low detection score generally indicates a need to expand or refine the knowledge base components which deal with that particular performance measure. It may also indicate that the approach used in the knowledge base is too conservative, and should be modified in favor of taking more risks.

c. DISCRIMINATION -- The discrimination score S is a percentage-form measure of the ability of an expert system to avoid erroneous conclusions, regardless of how many correct answers it finds. The discrimination score is defined as follows:

$$S = \frac{100 * |C|}{|B|}$$

(Note that if |B| is zero, the detection score is declared to be undefined for the current test set.) One way to obtain a high discrimination score would be to take an conservative approach in which only very safe conclusions are asserted. Unfortunately, such an approach would also result in a low detection score, since few of the known conclusions would be found. A low discrimination score would generally indicate poor "modeling" of the subject area of the performance measure; the knowledge base could be making overly simplistic assumptions about the nature of the subject area, or it could simply contain erroneous entries. A low discrimination score could therefore indicate a need to review or expand the knowledge base for that performance measure.

d. COMPREHENSION -- The comprehension score is a derived result; it is simply the product of the detection and discrimation ratings, divided by 100 to keep the result in percentage form. A comprehesion rating of 100% would indicate that the conclusion set reached by the expert system is exactly the same as the conclusion set reached by the accepted authority. The term "comprehension" is used to describe this score because, for the subject area of the performance measure, it is perhaps the single best indicator of the overall level of sophistification of the knowledge base.

The comprehension score is defined as follows:

$$I = \frac{D * S}{100} = \frac{100 * |C|**2}{|A| * |B|}$$

2.4 THE ACCEPTANCE THRESHOLDS TABLE

Although performance measures and scoring values can provide useful data about the behavior of a knowledge base, they do not by themselves define what is an "acceptable" level of knowledge base performance. To determine acceptability, a table of "acceptance thresholds" is needed.

An acceptance thresholds table is simply a list (by performance measure) of the minimum acceptable percentages for the detection, discrimination, and (optionally) comprehension scores. Note that expansion scores represent transient data obtained during construction of the knowledge base, and are not appropriate for acceptance testing. Comprehension threshold values are optional because comprehension scores are derived from detection and discrimination scores, and thus give redundant information.

The ability to have independent acceptance thresholds for detection and discrimination is important, since the relative significance of the two may vary with the objectives of the expert system. For example, if the expert system were only required to find a single answer to a problem that is known to have many answers, it might be acceptable to set the detection threshold at a low value. On the other hand, if reliability is extremely important, the discrimination threshold should be set at 100% to indicate that wrong answers are never acceptable.

3. RESULTS OF USING MULTIPERSPECTIVE ANALYSIS

Multiperspective testing was developed at Bendix primarily as an internal tool for testing and evaluating expert systems. It has been used fairly extensively in the testing of at least one such expert system, although the test results for that system were based only on a moderate quantity of test data. In one case, an evaluation of one performance measurement resulted in a low discrimination score, which suggested that the knowledge base needed stronger modeling of "normal" problem domain behavior. In this instance, the direct result was an effort to strengthen the knowledge base by adding discrimination-type rules.

Even with the small set of tests performed so far, results have been encouraging; at very least, the technique provides a "handle" for getting started in a difficult test domain. Additional trials of the technique, including its use in acceptance testing, are planned.

4. CONCLUSIONS

Multiperspective testing is a promising technique, but it needs to be applied to a broader range of systems before firm judgements can be made. Perhaps its strongest features are its ability to provide better correlation of test results with the knowledge engineering effort, and its ability to permit varying levels of thoroughness and reliability to be specified for different functional characteristics of a single knowledge base.

Another simple but important advantage of multiperspective testing is that it can be used to monitor and demonstrate progress in specific, critical areas of the knowledge engineering effort. This ability could probably be used to show progress in much the same fashion that software builds can be used to demonstrate and test key features of a large software system before the system as a whole is complete.

Finally, many of the specification techniques described here for testing could also be used "up front" to define generic requirments for how to build a knowledge base. Unlike a traditional requirments specification, a Multiperspective Requirements Specification would have the advantage of providing specific guidance about overall features and levels of reliability needed for an expert system, without attempting to pin the system down to deterministic lists of detailed functional requirments.

PAPERS OF GENERAL INTEREST

Sq-63
233139
6P.

Simple Methods of Exploiting the Underlying Structure of Rule-Based Systems

James Hendler
Department of Computer Science
University of Maryland
College Park, Maryland

May 4, 1986

1. Introduction

Much recent work in the field of expert systems research has aimed at exploiting the underlying structures of the rule base for reasons of analysis. Such techniques as Petri-nets and GAGs have been proposed as representational structures that will allow complete analysis. Much has been made of proving isomorphisms between the rule bases and the mechanisms, and in examining the theoretical power of this analysis. In this paper we describe some early work in a new system which has much simpler (and thus, one hopes, more easily achieved) aims and less formality.

The technique being examined is a very simple one: OPS5 programs are analyzed in a purely syntactic way and a FSA description is generated. In this paper we describe the technique and some user interface tools which exploit this structure.

2. Anssumptions

Our starting assumptions are important in understanding the work described in this paper.

(1) *We believe that the reduction from the rule-base to the network should be purely syntactic.* This restriction limits the power we can expect from this system, but allows us to avoid the mistake of making the reduction a more powerful, but less well understood technique.

(2) *The reduction should not need to account for variable bindings.* If variable bindings between rules is taken into account the mechanism performing the reduction ends up needing to unify rules clauses to each other. We are purposely trying to design a system which can be simple and efficient by avoiding such decisions.

(3) *The network produced by the reduction can include "errors" which more complete analysis could catch.* By this we mean that as we reduce a rule-base to a network we are essentially stating what rules are reachable from which others. Often, however, rules which our mechanism says are reachable won't be, since for example, the whole set of activating info for some rule may never turn out to be in working memory at the same time. Again, our system is kept simple and syntactic. It essentially produces a "heuristic approximation" of the set of possible futures. We view it as an aid for human analysis of the system, not an analysis device in its own right.

3. The reduction

The methodology we use for reducing the rules to the network is extremely simple (and somewhat simplistic as discussed elsewhere in this paper). We simply take each rule and index it according to the working memory elements its conditions either match or require to be excluded (via negation), and on the working memory items it makes and/or removes (modify assumes a remove and a make). Everything else (write, read, lisp, etc.) is ignored. All working memory element variables are assumed to match any variable or constant, wherever they appear.

For example, consider the following OPS5 rule from the famous "monkeys and bananas" example:

```
(p mb1
   (goal ↑status active ↑type holds ↑object <w>)
   (object ↑name <w> ↑at <p> ↑on ceiling)
   -->
   (make goal ↑status active ↑type move ↑object ladder ↑to <p>))
```

Our system stores the information that mb1:

```
1) Depends-on:
     (GOAL ACTIVE HOLDS ? ?)
     (OBJECT ? ? CEILING)
2) Creates:
     (GOAL ACTIVE MOVE LADDER ?)
```

(Thus, the system has rendered all variables into '?' and stored the conditions and actions as separate properties of 'mb1.')

If we had a more complex rule such as:

```
(p mb99
   (goal ↑status active ↑type holds ↑object nil)
   (object ↑name <o> ↑at <p>)
   (monkey ↑holds {<x> <> nil})
   - (open ↑door <d>)
   -->
   (make open ↑door <d>)
   (modify 3 ↑at <p>))
```

We would add the following to mb99:

```
1) Depends-on:
     (GOAL ACTIVE HOLDS NIL ?)
     (OBJECT ? ? ? ?)
     (MONKEY ? ? ?)
2) Creates:
     (OPEN ?)
     (MONKEY ? ? ?)     <from the modify>
3) Depends-on-not:
     (OPEN ?)
4) Removes:
     (OPEN ?)     <also from the modify>
```

We can turn this into a usable form by first creating a state space of all the memory elements which are found in the 'creates' of a rule-set (merging those which are equal), and then matching the 'depends-on' information against this space. Thus if we had the state, S1, with the pattern "(object ladder ? heavy ?)," mb99 above would depend on it since S1 matches "(object ? ? ? ?)" which is listed in mb99's 'depends-on' set. 'Depends-on-not' and 'removes' function analogously.

For the rest of this short paper we will use the example rules:

```
(p t1
      (start 1)
   -->
      (make monkey ↑at 5-7 ↑on couch)
      (make object ↑name couch ↑at 5-7 ↑weight heavy)
      (make object ↑name bananas ↑on ceiling ↑at 2-2)
      (make object ↑name ladder ↑on floor ↑at 9-5 ↑weight light)
      (make goal ↑status active ↑type holds ↑object bananas))

(p mb1
      (goal ↑status active ↑type holds ↑object <w>)
      (object ↑name <w> ↑at <p> ↑on ceiling)
   -->
      (make goal ↑status active ↑type move ↑object ladder ↑to <p>))

(p mb2
      (goal ↑status active ↑type holds ↑object <w>)
      (object ↑name <w> ↑at <p> ↑on ceiling)
      (object ↑name ladder ↑at <p>)
   -->
      (make goal ↑status active ↑type on ↑object ladder))

(p mb3
      (goal ↑status active ↑type holds ↑object <w>)
      (object ↑name <w> ↑at <p> ↑on ceiling)
      (object ↑name ladder ↑at <p>)
      (monkey ↑on ladder)
   -->
      (make goal ↑status active ↑type holds ↑object nil))

(p mb4
      (goal ↑status active ↑type holds ↑object <q>)
      (object ↑name couch  ↑weight heavy at <l>)
      (monkey ↑at <p> ↑on couch)
   -->
      (make goal ↑status active ↑type holds ↑object nil ↑to <l>)
      (make goal ↑status active ↑type on ↑object ladder))
```

which are transformed into:

```
t1:
   creates:
   (s4 s3 s2 s1 s0)
   depends-on:
    nil    <since no other rule can create the pattern (start 1)>

mb1:
    creates:
     (s5)
    depends-on:
     (s4 s7 s1 s2)

mb2:
    creates:
     (s6)
    depends-on:
     (s4 s7 s1 s2 s3)
```

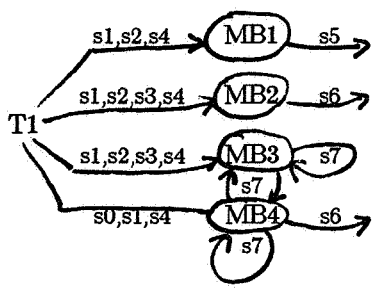
```
mb3:
    creates:
     (s7)
    depends-on:
     (s4 s7 s1 s2 s3)

mb4:
    creates:
     (s6 s7)
    depends-on:
     (s4 s7 s1 s0)
```

where the states are:

```
s0:
  (monkey |5-7| couch ?)
s1:
  (object couch |5-7| heavy ?)
s2:
 (object bananas |2-2| ? ceiling)
s3:
  (object ladder |9-5| floor light)
s4:
  (goal active holds bananas ?)
s5:
  (goal active move ladder ?)
s6:
  (goal active on ladder ?)
s7:
  (goal active holds nil ?)
```

The information yielded in this manner enables us to create a finite state automata network, with the rules as nodes and the states as transitions. To do this we connect each rule which creates a state to all rules which depend on that state. For the example above the FSA looks like:



We would also link the 'removes' and 'depends-on-not' states, and also add an inward link to each node which has a depends-on-not from each node which doesn't include that node's input states in its 'creates' field (not shown in this example).

4. Using the information

The FSA created by the above processes can clearly not be used for a complete analysis of the expert system. Variable bindings have been ignored—we cannot really count on our results. What is it good for?

We are presently building an expert system rule interface which relies on this network. Notice that with the information of section 3 we are able to answer a large set of questions—although somewhat imperfectly. From the FSA we can answer questions like "what rules might fire as a result of applying T1?" or "what are all the rules that might come immediately before MB4?" Further, by doing some simple boolean operations on the network we can also answer more complex questions such as "What rule might be fired after applying T1, and T3 and assuming T4 is unable to match?"

Another aspect of this system is the state information stored in the FSA. Because of this we can answer questions about specific working memory elements that the user is interested in examining. For example, we could ask "What rule might fire if I add (OBJECT ↑ NAME COUCH) and (MONKEY ↑AT 5-7) to the working memory?" or "What rules might be effected if (GOAL ↑STATUS ACTIVE ↑TYPE HOLDS) was removed?"

We believe that this sort of information, even flawed as it is in our system, provides a powerful tool for debugging a rule set. The system will let the user train through a set of rules looking at potential antecedent and consequent rules. Once interesting property of this method is that our system may make errors of commission, but won't make errors of omission—all the correct rules will be reported in the answer to a question, although some of the rules reported may not actually be applicable.

5. Conclusions

The simple technique described in this paper is a powerful tool in the creation of a system for debugging expert system rule bases. Unlike most such systems ours is based on a simple technique, easily implemented and efficiently run. Also unlike these systems ours makes mistakes which the user is expected to catch. We are willing to put up with this type of error since the question of how to do a full analysis is presently an unsolved problem.

510-63
233140
18P.

DIOGENES

EXPERT SYSTEM FOR EXTRACTION OF DATA SYSTEM REQUIREMENTS

Robert W. Hobbs
T. Patrick Gorman
Computer Technology Associates, Inc.
7501 Forbes Blvd., Suite 201
Lanham, MD 20706

DIOGENES: AN EXPERT SYSTEM FOR EXTRACTION OF USER REQUIREMENTS

We have developed a prototype expert system which has established the feasibility of automating a scenario-driven methodology for deriving top-level specifications and preliminary designs for user data systems. This scenario-driven methodology uses an initial design, an initial operations concept, and user scenarios as the starting point for system definition. The top-level initial design is a functional description of the system in the form of an annotated data flow diagram. The initial operations concept expresses information about system objectives, and defines the system users, system interfaces, and operational performance constraints. The user scenarios are detailed time-lined descriptions of user activities, developed by prospective end users. These scenarios, along with the initial design and operations concept, are analyzed and iterated by the expert system to form a consistent set. The resulting User Scenario-Operation Concept Set plays a key role in the development of requirements and system tests.

The expert system which we have developed and named DIOGENES provides an automated means for systems engineers (who are typically not the end users of data systems) to communicate with prospective end users (who are typically not systems engineers). A brief description of how the full-scale Diogenes system will work is as follows: A system engineer who has developed an initial design and an operations concept interacts with the expert system to describe the top-level system functions, data flows, data rates, I/O devices, etc. From this interaction, the expert system creates an annotated data flow diagram in the form of a display list to describe this design. The expert system then encodes the list into a set of rules. A prospective end user can then interact with these rules to describe his specific use. The system guides the user in entering a scenario which can be carried out within the bounds of the initial design but also allows the user to take exception to it, if (which is likely)

the user scenario cannot be carried out within the initial design. The systems engineer is provided with a copy of the scenario and a description of the exceptions taken. The engineer can then modify the design and interactively enter the modified design into the expert system. This process is repeated until the design and scenario converge.

The task of designing user data systems thus becomes a process of coupled interaction carried out by the system engineer and the prospective end user, with each successively interacting with the expert system. A top level graphical illustration of the expert system and its use is shown in Figure 1.

Our system is based on the following assumptions:

1. A system end user is the best source of information in establishing top-level requirements for user data systems.

2. End users typically can conceptualize time-ordered sequences of activities (scenarios) that describe their potential use of the system in advance of system design.

3. A structured grammar can be developed which allows a user accurately and flexibly to express these scenarios.

4. The structured grammar can have subjects, verbs, objects, etc., which are defined both by a systems engineer, who has developed a concept of operation for the data system, and by the user.

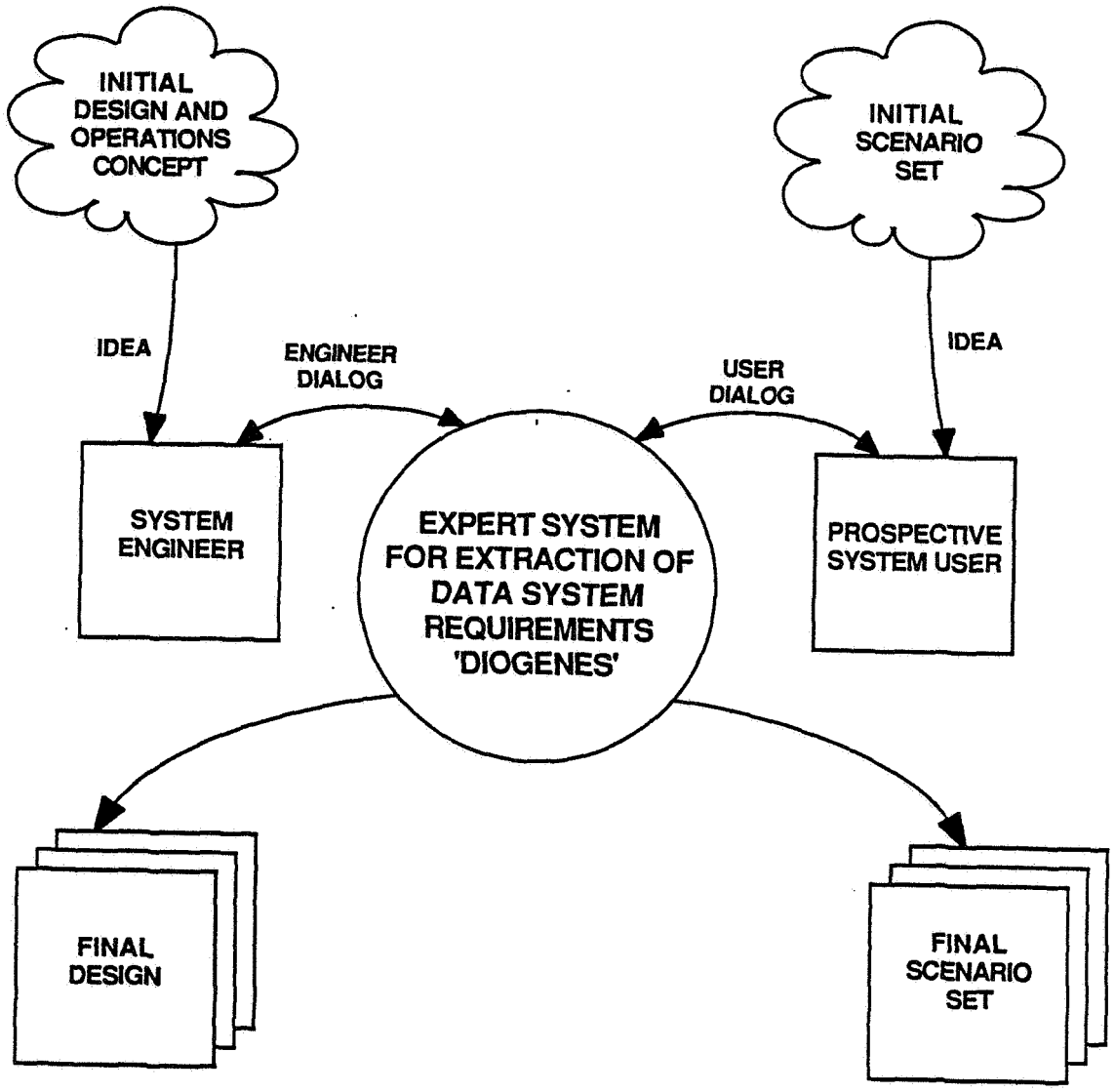


Figure 1. Top-Level Graphic Description of an Expert System for Extraction of Data System Requirements

5. The expert system can a) guide the systems engineer and the user in setting up the subjects, verbs, objects, etc.; and b) guide the user in entering a scenario in such a way that engineering principles can be evoked by the expert system to derive performance and functional requirements for the data system.

The prototype, which we have developed, can carry out a small set of the functions possible for a full scale system. Namely, this prototype:

1. Guides a Systems Engineer in establishing an Initial Design Concept and in expressing system level "*a priori*" constraints, requirements and design parameters.

2. Guides a prospective data system end user in entering a scenario which flexibly describes the system from his point of view.

3. Prompts for changes to the user scenarios or lists required changes in the initial design to make the set of descriptions (initial design, user scenario; and requirements) consistent.

4. Generates a list of system functional requirements.

This prototype has successfully demonstrated the feasibility of developing a full scale expert system for extraction of data system requirements. The expert system we have demonstrated is unique in that it provides expert assistance in two (as opposed to the typical single) domains of knowledge: the system engineering domain and the user domain.

Our approach in developing an expert system for automatically extracting requirements from a user scenario is

influenced by the requirement that the proposed expert system in its completed configuration has two users: the user who will be entering a scenario; and the systems engineer who will be first entering an operations concept and initial design, and later using the system to extract functional and performance requirements. The knowledge base must therefore be effective in two domains: a context-free user domain which allows the scenario user to enter a scenario descriptive of his desired use, and the engineering domain.

In this paper, we will address two main elements 1) language development and 2) prototype development. Originally it was planned to carry out several other individual tasks involving the development of rules for scenario descriptions and for requirement extraction. Two considerations led us directly from language development to prototype development. First, the structure of the rules depends to a great extent on the prototype architecture, and second, it seemed unnecessary to limit the scope of potential "data system" designs based on a specific set of derived rules. For the latter reason we chose to investigate the possibility of having some of the rules encoded by the expert system, thus having the "expert knowledge" in the area of rule generation (meta rules) rather than in the area of specific data system design. This approach, as will be shown later, widens the scope of the expert system to any system that can be described in terms of an automated data flow diagram (a data flow diagram with data rates, I/O devices, etc., specified).

LANGUAGE DEVELOPMENT

The objective of this activity was to develop and test a structured language for expressing user activities in a way suitable for use in an expert system for deriving requirements and extracting system functions. We were guided in this development by two considerations. First, the language must be able to capture in a simple way a user's description of

activities he wishes to carry out using a system under development. Second, the language structure must be able to describe virtually any activity whether carried out by the system automatically or under the control of the user.

We have chosen the Operation as the fundamental transaction to be described by the language. An operation is shown graphically in Figure 2.

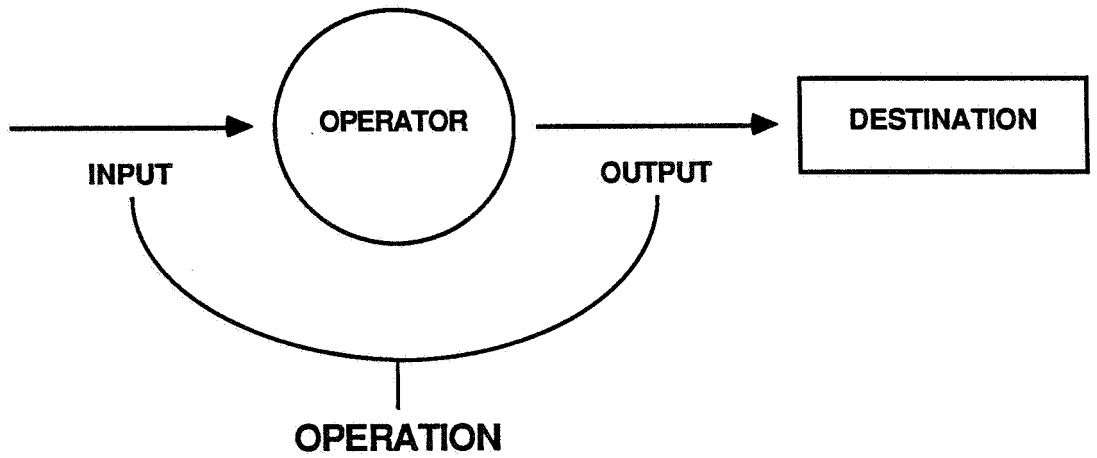


Figure 2. Language Structure

Each operation is completely described by an input, an operator, an output, and a destination for that output. We suggest that, for this language, inputs and outputs can be associated either with data or with control. Likewise, operators can be either algorithms, logical operations, or control operations. The concept of an operation, as opposed to a process, as being the basic transaction allows greater flexibility in expressing user scenarios.

The language which is developed as a result of this research will be based on the general form:

From WHAT INPUT use WHAT OPERATOR to perform WHAT OPERATION to produce WHAT OUTPUT which goes to WHAT DESTINATION?

The inputs and outputs can either be external inputs, internal data flows, or control statements. The latter two cases can be completely specified by the user. In the case that the user defines inputs to operations, those inputs must form a logical sequence through the scenario -- that is, inputs must be either external data sources or outputs from previous scenario statements. This requirement will be expressed by rules.

Operators are selected from a limited list of operators which have specified functional implications for the operations concept and initial design. Sample operators might be:

INTERNAL DATA
DEFAULT VALUE
MANUAL DATA ENTRY
LOGICAL OPERATION
ENUMERATION
ALGORITHM
POINTING DEVICE

Other operators could be specified by the user, but if not included in the initial design, they would be considered as "exceptions". Details of operators such as algorithms would be specified by the user.

Likewise operations are selected from a fixed list which are related to system functional capabilities. Sample operations would be:

CONVERT
LABEL
SELECT
DISPLAY

DELETE
INITIALIZE
DETERMINE
IDENTIFY

Other operations could, of course, be designated by the user. We anticipate that the set of operators and operations presented here is not complete and would have to be modified and expanded depending on the intended use and design of a specified data system.

Several sample scenarios were developed using this language to test for the ability of the language to express the user's activities. An existing user system scenario was used as a starting point, namely, one which was developed for operations concept analysis of the Space Telescope (ST) Science Operations Ground System. The use of an existing scenario has two advantages: First, the effort to develop a realistic scenario need not be repeated and second, the realism of the scenario has already been verified. There is one disadvantage in that the existing ST scenario is presented in greater detail than can be expected to be handled by our small scale prototype system. However, the advantage of being able to test the language with realistic scenarios we believe is extremely valuable. Two such scenarios are presented here:

ST DEMONSTRATION SCENARIOS

Scenario #1

Description: "Identify star to be observed"

INPUT	OPERATOR	OPERATION	OUTPUT	DESTINATION
User Action	cursor position	select	target position	subsequent operation
User Action	keyboard entry	specify	image display co-ordinate request	next operation and display
image display co-ordinate request	logical operation	convert	image display	scratch file and display

Scenario #2

Description: "Calculate Offset Parameters"

INPUT	OPERATOR	OPERATION	OUTPUT	DESTINATION
User Action	keyboard entry	specify	SI-acquisition	scratch file and display
User Action	keyboard entry	specify	SI-observant aperture	scratch file and display
User Action	keyboard entry	specify	small slew offset request	next operation
Offset Request	logical operation	select	scratch file	next operation
Scratch file	algorithm	calculate offset	small slew offset request	

Several other small scenario segments using this language were developed in an attempt to demonstrate the utility of the structured grammar described above, and we believe these demonstrate that the language structure chosen can express the scenario accurately. As expected, more lines are used in the derived scenario then in the initial scenario. This is a direct result of the more general formulation expressed by the language.

Based on the success of our attempt to develop this structural grammar and its test against actual ground system operations scenarios, we proceeded to the prototype development in order to validate the appropriateness of the grammar for use in the expert system.

One aspect of an operations scenario that is not captured by the structured grammar is the time-line. For this initial feasibility study, we have limited the research to time-ordered as opposed to time-lined scenarios. We felt that time-lining was probably not feasible using IQLISP on the IBM PC, the system chosen for the prototype development. Time-lining is, however, extremely important in establishing performance requirements.

PROTOTYPE DEVELOPMENT

The architecture of the prototype provides a framework for development of all aspects of the expert system. Because this is a complex system comprising two interfaces (system engineer and end user) and two types of expert knowledge (system knowledge and requirement extraction knowledge), we expect that the structure of the rules will depend in fact on the architecture of the system. We therefore chose to develop the basic system architecture before proceeding to develop the necessary rules. The system has four crucial elements: The annotated data flow diagram to be entered by the systems engineer, the scenario to be entered by the user, rules for scenario entry, and rules for requirements extraction. As stated earlier, in order to

generalize the system, we have chosen to have the rules for scenario entry be derived from the system description (i.e., the annotated data flow diagram).

The architecture of our Diogenes prototype is shown in the data flow diagram of Figure 3. It is made up of two user interfaces, two knowledge bases, six processes and ten dataflows. Discussion of these elements follows, proceeding clockwise from 9 o'clock. At the engineer's interface, Diogenes engages the engineer in a predetermined dialog, presenting menus based on any known initial requirements. It then assembles choices entered by the engineer into an IQLISP-acceptable display list which represents the annotated data flow diagram of the engineer's proposed data system.

Diogenes then encodes this display list into a set of production rules based on expert knowledge, in the form of meta-rules, of how these annotated diagrams are related to rules-of-use for data systems.

Next, Diogenes executes a dialog with the system user which is defined by the rules entered in the previous step. The structure of the engineer's system is presumed fixed and the user is invited to attempt to enter the scenarios which define his agenda within it. However, provision is made for user choices which are "non-conforming". These choices represent functions which the engineer's system cannot now perform but which the user believes it must be able to perform in order to be satisfactory.

Diogenes assembles a usage record from the user-entered choices after rejecting those which are identified as simple entry error. This time-ordered listing of the user's dialog represents the user's scenario, modified by its interaction with the system rules.

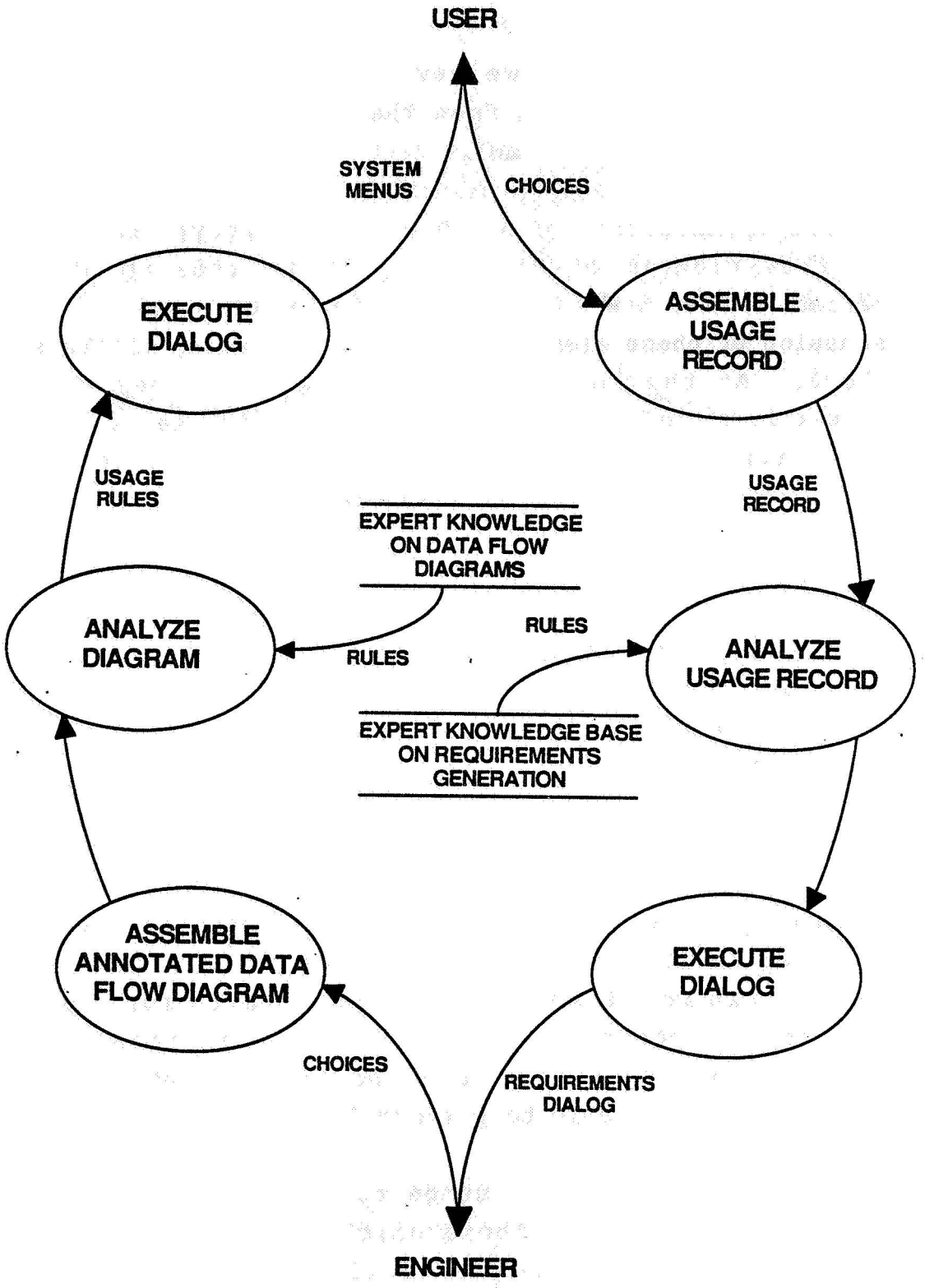


Figure 3. Diogenes Prototype Architecture

Now Diogenes analyzes the usage records created by the modified scenario based on the meta-rules in its knowledge base which embody its expertise on the effect of usage records on system characteristics. It produces sets of rules which define the system, now modified to accommodate the user scenario.

Finally, Diogenes executes a dialog with the engineer based on the new memory-producing rules. This step closes the engineer-user system design loop.

The overall control of Diogenes is exercised by two sets of meta-rules. One set embodies knowledge of the relationships between system descriptions in the form of annotated data flow diagrams and rules for system usage. The other represents knowledge of the relationships between usage records and system performance requirements.

The use of the annotated data flow diagram as the framework for the scenario entry process is, we believe, a feature of the proposed expert system: the data flow diagram allows virtually any design (of any type system: i.e., ground system, spacecraft on-board system) to be described. The scenario entry user works within this framework and can define his own operations within it. For example, the system engineer may describe data inputs to major functions, but the scenario user can define outputs of operations and inputs to subsequent operations with a great deal of flexibility; these operations become elements of the next level of design, as requirements are extracted by the expert system and presented to the system engineer. A sample of an annotated data flow diagram of the type to be used for the development of this prototype is shown in Figure 4.

The first implementation of our Diogenes prototype was designed to be up and running quickly, so as to help us in our research. The prototype uses an inference engine encoded in

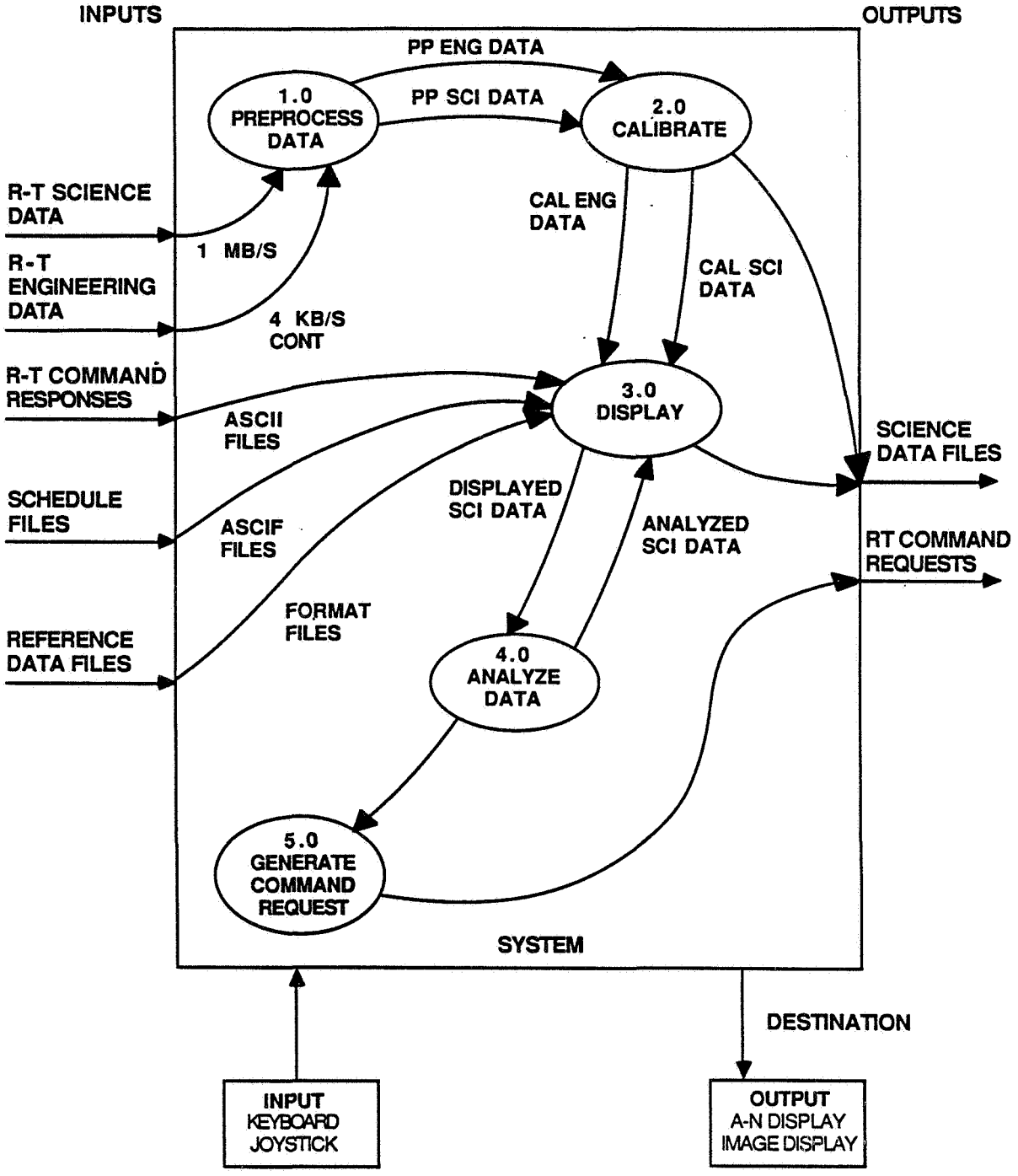


Figure 4. Annotated Data Flow Diagram

IQLISP and can be run on an IBM PC. As a result, it represents Diogenes in broad outline only and in many ways simply foreshadows the fully functioning system.

For example, the first implementation can execute and assemble the results of a dialog with a systems engineer and with a system user. The latter is rule-based, the former is not. The first implementation does not separately represent its knowledge bases as the full system will. The first implementation does not produce system rules from annotated diagrams or requirement rules from usage records. It performs only a rudimentary usage record assembly function by counting the frequency with which each menu entry was chosen by the user.

The prototype operates as follows:

First, the systems engineer interacts with Diogenes to describe the design of a proposed data system. The design can be expressed at any desired level of detail. Typically, we expect that initially the design would be expressed at a high level and through successive iterations come to be expressed in greater and greater detail as user scenarios are likewise entered with greater detail at each iteration. The output of the interaction would be an annotated data flow diagram with standard operations, standard operators, inputs, outputs, and if known, performance characteristics.

Second, a prospective end user interacts with Diogenes to describe and record time-lined descriptions of his proposed use for the final data system. Scenario entries are validated against the proposed design based on rules following from the system engineer interaction and exceptions noted.

Descriptions of exceptions are used by the system engineer in the next successive design description. This is likewise an iterative interaction. Initially there would be many exceptions

noted, and as the proposed system design is iterated, exceptions would be expected to become less frequent. The man-machine interface for Diogenes is a keyboard and an alphanumeric display. Diogenes is <u>not</u> a prototyping system. There is no capability for building displays, for example.

The dialog for the system engineer is fairly simple, consisting of a series of queries regarding the system description which is to be in the form of a data flow diagram, annotated as to data flow descriptions, data rates, volumes, etc., and with I/O devices.

The scenario entry interaction dialog is likewise simple, but related to a rather sophisticated scenario language in the form described earlier in this paper.

As stated above, inputs to the system are the systems engineers' input to describe the system, and the prospective end user's input to describe his scenario. The scenario entry dialog includes information statements to the prospective end user to describe whether his scenario statement was accepted (validated against the design) or rejected. If rejected, reasons are given.

A printed description of the proposed data system is available after each systems engineer interaction. At the completion of an interaction (one system engineer session followed by a prospective end user session) the following printed outputs are available:

1. Complete description of the scenario entry process (all user keyboard entries and system responses)

2. The set of validated scenario statements

3. A list of user exceptions (cases where proposed design cannot accommodate the desired use)

4. A list of requirements extracted from the scenario relative to the proposed system design.

The systems engineering user is constrained to describing the system in terms of an annotated data flow diagram. The prospective end user is constrained to describe scenario activities in terms of the scenario entry language.

We believe that the proposed system and its possible extensions have great applicability. An expert system which assists a designer in providing a computer system having high fidelity to user requirements has the potential for playing a key role in the many user-oriented computer systems that will be developed in the future. The beneficial products of such a system are two fold. First, the system provides an analyst with a rapid, accurate means of deriving functional and performance requirements. Second, the development of any system which employs a user-oriented requirements study as a part of a front-end analysis is likely to be more cost effective if user needs are considered earlier rather than later in the evolution of the system.

In addition, the eventual implementation of such a system is likely to include a very effective means of rapid man-machine interface prototyping and a means of deriving acceptance tests procedures. Another key feature of the system enables the knowledge base to grow to represent the aggregate expertise of the best systems engineers. Considering the benefits of such an expert system for assisting in the development of large scale systems we conclude that most future systems development programs will not only employ the methodology described here, but will also make use of an expert system for that purpose.

Acknowledgement

The work described in this paper was performed for the Jet Propulsion Laboratory as NASA SBIR 0708-5300.