(NASA-CR-187408) EVALUATION OF THE TRAJECTORY OPERATIONS APPLICATIONS SOFTWARE TASK (TOAST) (Houston Univ.) 65 p CSCL 09B

N91-13104

Unclas
G3/61 0312546

EVALUATION OF THE TRAJECTORY OPERATIONS APPLICATIONS SOFTWARE TASK (TOAST)

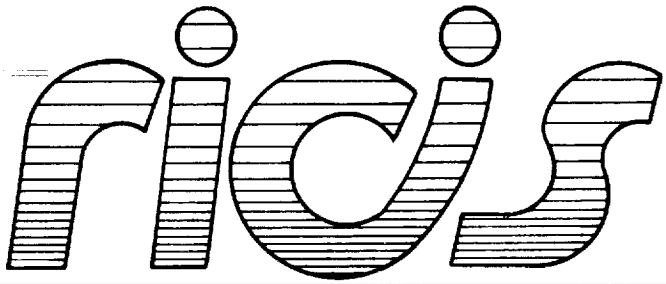
Sharon Perkins
University of Houston-Clear Lake

Andrea Martin
Bill Bavinger
Rice University

August 27, 1990

Cooperative Agreement NCC 9-16
Research Activity SE.36

NASA Johnson Space Center
Mission Operations Directorate



Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

EVALUATION OF THE TRAJECTORY OPERATIONS APPLICATIONS SOFTWARE TASK (TOAST)

Preface

This research was conducted under the auspices of the Research Institute for Computing and Information Systems by: Dr. Sharon Perkins, and Dr. Alfredo Perez-Davila, both Assistant Professors of Computer Science, University of Houston-Clear Lake; Ms. Andrea Martin, Manager, Computing Resource Center, Rice University; Bill Bavinger, Assistant Professor of Architecture, Rice University; David Boyes, consultant; and Dr. Livia Polanyi, consultant. Dr. Sharon Perkins served as RICIS research representative.

Funding has been provided by Flight Design and Dynamics, within Mission Operations Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Mike Evans.

The views and conclusions contained in this report are those of the author and should not be interpreted as representative of the official policies, either express or implied, of NASA or the United States Government.

Overview

The Trajectory Operations Applications Software Task (TOAST) is a software development project under the auspices of the Mission Operations Directorate. Its purpose is to provide trajectory operation pre-mission and real-time support for the Space Shuttle program.

As an Application Manager, TOAST provides an isolation layer between the underlying Unix operating system and the series of user programs. It provides two main services:

1. A common interface to operating system functions with semantics appropriate for C or FORTRAN

2. A structured input and output package that can be utilized by user application programs.

These two services can be used independently of the environment, providing a flexible application toolkit.

In order to evaluate TOAST as an Application Manager, RICIS undertook an evaluation of the system under NASA Cooperative Agreement NCC 9-16. The task was to assess current and planned capabilities, compare capabilities to functions available in commercially-available off the shelf (COTS) software, and analyze requirements of Mission Control Center (MCC) and Flight Analysis Design System (FADS) users for TOAST implementation. The project team consisted of faculty, staff, and students from University of Houston-Clear Lake and Rice University. Principal investigators were Sharon Perkins, Andrea Martin, and Bill Bavinger.

The evaluation began on March 28, 1990 and completed September 1, 1990. Preliminary results were presented to Flight Dynamics on June 1. Security briefings were delivered on May 24 and June 7. An executive briefing for management was presented on June 21. A formal presentation to the NASA community was presented on June 22. An additional executive briefing for Flight Design management was delivered on July 10.

As a result of our investigation, we found that the current version of TOAST is well implemented and meets the needs of the real-time users. The plans for migrating TOAST to the X Window System are essentially sound; the Executive will port with minor changes, while Menu Handler will require a total rewrite. In this report, we include a series of recommendations for future TOAST directions, which is summarized as follows:

Plan for a distributed operating environment with services, such as event notification, authentication and configuration management, database, and graphical user interfaces, provided via a high speed network.

Table of Contents

List of Figures 6

Acronym Glossary 7

I. The Task 8

II. Project Team Synopsis 8

III. Our Approach 11

IV. High Level Summary 12

V. Needs Analysis 13
A. User Requirements 13
1. Evaluation Strategy
2. Task Analysis 14
a. Tasks Common to FDOs and Orbit Flight Design Users 14
b. Tasks of Orbit Flight Design 14
c. Tasks of Flight Dynamics Officers (FDOs) 15
3. Requirements 15
a. Requirements of Orbit Flight Designers 15
b. FDO Requirements 15
c. Additions to TOAST to Support Flight Design 16
d. Summary 16
B. Environmental Constraints 16
C. Design Philosophy 17
1. Role of an Application Manager 17
2a. Executive-based Design 18
2b. Overall Evaluation of TOAST Executive 19
3a. Structured Display Control 19
3b. Continuum of User Interfaces 20
3c. Overall Evaluation of Menu Handler 20
4a. Menu-Application-Display (MAD) Model 21
4b. Overall Evaluation of Menu-Application-Display (MAD) Model 21
5a. User Requirements versus Design Implementation 22
5b. Overall Evaluation of User Requirements Implementation 22

VI. Technical Analysis 23
A. Fault Tolerance 23
1. Task of Fault Tolerance 23
2. Evaluation Strategy 23
3. Fault Tolerance in TOAST 23
4. Strengths of Present Implementation 24
5. Weaknesses of Present Implementation 24
6. Overall Evaluation of Strengths and Weaknesses 24
7. Recommendation 24
8. Summary 24

B. Portability 25
1. Task of Portability 25
2. Evaluation Strategy 25
3. Portability in TOAST 25
4. Strengths of Present Implementation 25
5. Weaknesses of Present Implementation 25
6. Recommendation 25
7. Summary 25
C. Comparison: Multi-Level Menu System to COTS Application Manager 26
1. Task of a Menu Application System 26
2. Evaluation Strategy 26
3. Multi-Level Menu System in TOAST 26
4. Strengths of Present Implementation 26
5. Weaknesses of Present Implementation 27
6. Overall Evaluation of Strengths and Weaknesses 27
7. Recommendation 27
8. Summary 27
D. Comparison: Menu Handler to Text-based COTS Toolkits 27
1. Task of Forms Generation and Display Packages 27
2. Implementation in TOAST 28
3. Evaluation Strategy 28
4. Strengths of Present Implementation 28
5. Weaknesses of Present Implementation 29
6. Overall Evaluation of Strengths and Weaknesses 29
7. Recommendation 29
8. Summary 29
E. Data Structures 30
1. Task of Data Structures 30
2. Data Structures in TOAST 30
3. Evaluation Strategy 30
4. Strengths of Present Implementation 30
5. Weaknesses of Present Implementation 30
6. Recommendation 30
7. Summary 30
F. Security 31
1. Task of Security 31
2. Evaluation Strategy 31
3. Security in TOAST 31
4. Strengths of Present Implementation 31
5. Weaknesses of Present TOAST Implementation 31
6. Weaknesses of the Environment 32
7. Overall Evaluation of Security Strengths and Weaknesses 32
8. Recommendation 32
9. Summary 32

VII. Man/Machine Interface 33
A. Applications Programming Environment 33
1. Evaluation Strategy 33
2. Strengths of Programming Environment 33
3. Weaknesses of Programming Environment 33
4. Recommendation 34
B. User Interface Capabilities 34

VIII. Software Engineering 35
A. Coding Practices 35
1. Evaluation Strategy 35
2. Evaluation Criteria 35
3. Implementation in TOAST Software 36
4. Recommendation 37
5. Summary 37
B. Documentation Review 38
1. Evaluation Strategy 38
2. Evaluation Criteria 38
3. Implementation in Menu Handler Documentation 38
4. Strengths of Menu Handler Documentation 39
5. Weaknesses of Menu Handler Documentation 39
6. Recommendation 39
7. Summary 40
C. Revision Practices 40
1. Evaluation Strategy 40
2. Evaluation Criteria 40
3. The Revision Process for TOAST 40
4. Problem Areas 41
5. Recommendation 41
6. Summary 42

IX. Future TOAST Migrations 43
A. TOAST Under X 43
1. Evaluation Strategy 43
2. The X Window System 43
3. The Plan for TOAST under X 45
4. Discussion of Implementation Plans 46
a. Features that can be Implemented using Existing X Tools 46
b. Clocks and DDDs 46
c. Single Window 46
d. Function Keys 47
e. TOAST Resource Manager 47
5. Recommendation 47
6. Evaluation of Extent of Code Revision to Migrate TOAST to X 49
B. Expandability 50

X. Industry Perspective 51
A. Graphical User Interfaces (GUI) 51
B. Standard Graphics Packages 51
C. Database Systems 52
D. Distributed Authentication Systems 52
E. Network Services 52
F. Distributed Operating Systems 53

XI. Recommendations for Future Directions 54

XII. Further Reading 56

XIII. Acknowledgements 62

List of Figures

Figures	Page
1. Evaluation Criteria	8
2. TOAST Evaluation Interview Subjects	13
3. TOAST as Application Manager	18
4. Continuum of System Executives	19
5. Continuum of User Interfaces	20
6. Menu-Application-Display (MAD) Model	21
7. Prologue used in TOAST programs	36
8. Features that can be Implemented using Existing X Tools	46
9. Expandability: Advantages and Constraints of the Current Version	50

Acronym Glossary

AEP	Arbitrary Event Protocol
API	Applications Programmer Interface
BSD	Berkely Standard Distribution
COTS	Commercially-available Off The Shelf
CR	Change Request
DDD	Discrete Data Display
DoD	Department of Defense
DR	Discrepancy Report
DWIM	Do what I mean
FADS	Flight Analysis Design System
FDO	Flight Dynamics Officer
FDS	Flight Design System
GKS	Graphics Kernel System
GOSIP	Government Open System Interface Program
GUI	Graphical User Interface
I/O	Input/Output
IPC/RPC	InterProcess Communication/Remote Procedure Call
ISO	International Standards Organization
JCL	Job Control Language
LAN	Local Area Network
MAD	Menu, Application, Display
MOC	Mission Operations Computer
MOD	Mission Operations Directorate
MCC	Mission Control Center
ODP	Orbit Design Panel
ONAV	Orbit Navigation
OS	Operating System
OSI	Open Systems Interconnect
PEX	PHIGS Extension to X
PHIGS	Programmer's Hierarchical Interactive Graphics System
QBE	Query by Example
RAVL	Rice Advanced Visualization Lab
RICIS	Research Institute for Computing and Information Systems
SAA	Systems Application Architecture
SOW	Statement of Work
SQL	Structured Query Language
SVID	System V Interface Definition
TCP/IP	Transport Communications Protocol/ Interconnect Protocol
TOAST	Trajectory Operations Applications Software Task
TRM	TOAST Resource Manager
TWG	TOAST Working Group
WAN	Wide Area Network
WEX	Workstation Executive
X11R4	Version 11 Release 4 of MIT's X Window System

I. The Task

The purpose of this evaluation was to asses the capabilities of the Trajectory Operations Applications Software Task (TOAST) as an Application Manager. Our task was to assess current and planned capabilities, compare capabilities to functions available in commercially-available off the shelf (COTS) software, and analyze requirements of Mission Control Center (MCC) and Flight Analysis Design System (FADS) users for TOAST implementation.

Specific evaluation criteria listed in the NASA Statement of Work dated March 6, 1990, and sections of this report in which they are discussed are shown in Figure 1.

Criteria	SOW Section	Report Section
Environmental constraints	2.1	V. B
Standard application manager capabilities and services	2.2	V. C, VI
User requirements versus design	2.3	V. C
User/programmer tools	2.4	VII. A
User interface	2.5	VII. B
Application manager applications	2.6	V. C, VI
User protection and user access	2.7	VI. F
Data structures	2.8	VI. E
Coding practices	2.9	VIII. A
Portability	2.10	VI. B
Reliability	2.11	VI. A
Expandability	2.12	IX. B
Vulnerability	2.13	VI. F
Application interface	2.14	V. C, VI. C
X-windows	2.15	IX. A
Future of TOAST	2.16	X, XI

Figure 1: Evaluation Criteria

II. Project Team Synopsis

Evaluation of the NASA TOAST Executive involved application and integration of expertise, equipment, techniques and methods in the areas of task analysis, software engineering practice, man/machine interface, and systems simulation and testing. To provide the technical response to the interdisciplinary challenge of the TOAST Executive audit, RICIS assembled a team of experts from UH Clear Lake and Rice University equipped with the hardware and software resources necessary to meet the challenge of the evaluation task.

The principal investigators composed faculty and staff from UH Clear Lake and Rice University. Dr. Sharon Perkins, a faculty member in Computer Science at UH Clear Lake, provided overall project coordination. Ms. Andrea Martin, head of the Computing Resource Center at Rice University, served as project manager and coordinated the day to

day operation and scheduling of project activities. Together, they were responsible for reporting the interim and final results of the audit to responsible supervisory personnel at NASA. Professor Bill Bavinger, director of the Rice Advanced Visualization Lab, provided technical direction in the analysis of the evaluation data and graphics directions.

The Principal Investigators were assisted by Mr. David Boyes, Dr. Livia Polanyi, Dr. Alfredo Perez-Davila and technical staff. Mr. Boyes, who is president of dboyes Consulting and subcontractor on this research proposal, performed the TOAST software technical analysis and provided research support for comparisons with emerging and established industry standards. Dr. Livia Polanyi, a member of the Linguistics and Semiotics Department at Rice, developed the interview materials, administered the interviews, and analyzed the resulting data. Dr. Polanyi was responsible for developing the model of the task, users, and environment that was used in evaluating the adequacy of the TOAST environment to meet fully the challenge of the Mission Operations Directorate (MOD) project requirements. Dr. Alfredo Perez-Davila, a faculty member in Computer Science at UH Clear Lake, provided support for operating systems analysis.

Investigators

Sharon Perkins, Assistant Professor of Computer Science, UH Clear Lake
Dr. Perkins is an Assistant Professor in the Computer Science department at the University of Houston-Clear Lake where she is involved in teaching and research in image processing and computer graphics. She has been involved with several NASA-JSC research projects concerning workstation evaluation and software life cycle definition for the Engineering Directorate. She worked for IBM in Austin from 1980 to 1982, and has been a reviewer for *IBM Systems Journal* since 1985. She taught computer science at UT Austin and North Texas State University. She received a Ph.D from Texas A&M University in 1980.

Andrea Martin, Manager, Computing Resource Center, Rice University.
As head of user services, Ms. Martin directs a group of programmers, local area network specialists, trainers, and technical writers who support over 4000 users on Unix, IBM, and microcomputer systems. A member of the Rice Advanced Visualization Lab research team, she has worked as a research associate on a grant from IBM. From 1984 to 1987, she was director of the Rice Macintosh Software Development Project, which developed and delivered software applications including Conformal Maps, TSO Kermit, FlashCard, Pecos Pictorial Database, Plotting Calculus Derivatives, and Better Letter Guide. She currently serves as an advisor for the Smartnode program for the Cornell National Supercomputing Facility and is on the steering committee for the Rice Center for Scholarship and Information. She received a masters degree from Rice University in 1984.

Bill Bavinger, Assistant Professor of Architecture, Rice University.
Mr. Bavinger is Director of the Rice Advanced Visualization Lab. He has served as an investigator on grants and studies from the Department of Energy, U.S. Army Corps of Engineers, National Science Foundation, the Houston Design Center, the NASA JSC Engineering Directorate, and IBM. A practicing architect, he has been involved in 4 exhibits. As president of the Third Coast Computer Graphics Group, he published Computergraphia: New Visions of Form, Fantasy, and Function. Since 1979, he has published 6 articles and 12 major research reports. His research interests are in leading

edge graphics systems, geographic information systems, enterprise planning, and scientific visualization.

Technical Staff

David Boyes, President, dboyes Consulting.
Mr. Boyes is the head of dboyes Consulting, a computer consulting firm. He serves as a systems programmer responsible for IBM mainframe systems supporting a wide range of numerically intensive and educational computing resources at Rice University. He has pursued intensive research in artificial image enhancement systems, networking, interactive computer graphics and three-dimensional solid modeling systems using the X window system, and character recognition algorithms as part of an NSF/NEH joint project to enhance damaged or eroded monuments in Roman Gaul. He has worked on a JPL grant to study data flow to improve imaging performance for the Voyager project, supported a distributed operating systems project at the University of Oregon, implemented IBM's network job entry system for TOPS-10, TOPS-20, and Unix, and has researched network bridging services and protocol conversion for TCP/IP and DECnet. His interests include distributed operating systems and real-time embedded control systems, as well as design of cooperative work environments for non-traditional computer users. He is participating on the review committee for the Government Open System Interface Program (GOSIP), which is developing a user interface standard for applications running on government systems. He has published papers in distributed operating system design and received an M.A. degree from the University of Oregon in 1988.

Consultant

Livia Polanyi, Associate Professor of Linguistics and Semiotics, Rice University.
Dr. Polanyi is an Associate Professor in the Linguistics and Semiotics department at Rice University where she is involved in teaching and research in computational linguistics, cognitive science, design of intelligent tutoring systems, and communication and discourse theory. She is a member of the Computer and Information Technology Institute, an interdisciplinary consortium of Rice faculty who perform research in areas related to leading edge computing technology. She received a Ph.D from University of Michigan in 1978. From 1978 to 1985, she worked as an Associate Professor at the University of Amsterdam. She has worked for BBN Labs as a senior scientist in the Artificial Intelligence Department and as a consultant to the Information Sciences Division. During her tenure there, she led a project for the US Navy to design and implement an intelligent document analysis and retrieval workstation involving advanced human interfaces, intelligent text analysis, and expert system technology. She has served as principal consultant on grants from the Ford Foundation and the National Institute of Mental Health. She is on the review board for *Language*, reviews proposals for the National Science Foundation, is on the editorial board for *TEXT*, and serves as a reviewer for 4 journals. She is the author of The Structure of Discourse and Telling the American Story: A Structural and Cultural Analysis of Conversation Storytelling, and has published 30 articles. Her research interests are in computational linguistics, artificial intelligence, knowledge representation and acquisition, and communication theory.

Alfredo Perez-Davila,Assistant Professor of Computer Science, UH Clear Lake
Dr. Perez-Davila is an Assistant Professor in the Computer Science department at University of Houston-Clear Lake where he is involved in teaching and research in

operating systems. He was recently involved in a NASA-JSC research project to interface an artificial intelligence printer controller running under OS/2 with the host environment in the MCC. From 1987 to 1989, he was an Assistant Professor of Computer Science at the University of Pittsburgh. He received a Ph.D from Vanderbilt University in 1987.

III. Our Approach

The evaluation team approached the review of the TOAST Executive in five stages:

1. We built a conceptual model of the framework of tasks that the end users are expected to complete within the environment and established a set of appropriate criteria to use in determining how closely TOAST conformed to the original design criteria and user expectations. To gather the information necessary to build a useful model, we conducted, transcribed, and edited over 30 hours of interviews with users, system designers, application programmers, and management. Volumes I and II of the interview transcriptions are attached as separate documents. To understand the needs of flight design users better, we viewed a demonstration of the Flight Design System (FDS). We also observed TOAST use in the MCC during STS-31.

2. We conducted a detailed technical review of the performance of the TOAST Executive under the stress of seriously abnormal operating conditions as well as normal conditions. As part of the data gathering for this phase, we reviewed the code with the TOAST developers, extensively tested and used the Masscomp development systems, and investigated the MCC TOAST environment hands-on during User Computation Time.

3. We analyzed the man/machine interface. Our programmers wrote FORTRAN and C TOAST applications to exercise the programmer applications interface.

4. We examined software engineering practices. We reviewed the TOAST code and examined external documentation. We also investigated the techniques used by the development team to manage system configuration, user problem reports, and changes to the system.

5. We developed a set of recommendations for enhancements or changes to the existing system based on the results of the tests and analyses conducted during the review process and current trends in industry.

IV. High Level Summary

The TOAST software is workstation software that was developed for the trajectory flight controllers. It consists of an Executive, which provides an operating environment and coordinates user applications, and Applications, which are user programs. It provides the capability for several users to access TOAST simultaneously (multi-user), more than one process to execute simultaneously (multi-tasking), TOAST users to access multiple independent concurrent sessions (multi-session), and for users supporting more than one flight control position to simultaneously use TOAST (multi-position). Transparent cooperative processing (multi-machine), where users access resources across a network, is a future goal.

This section presents a high level summary of the evaluation results. All results are described in detail in the body of this report.

Needs Analysis (section V)

- TOAST can serve the needs of flight design and real-time.
- The overall design philosophy is sound.
- The Menu-Application-Display model is a good approach for dividing program functionality.
- TOAST implements initial user requirements.
- Environmental and systems management constraints severely hinder the development process.

Technical Analysis (section VI)

- TOAST is not fault tolerant.
- TOAST is portable.
- Menu Handler compares well with other COTS text-based menu handling toolkits.
- Multi-level menus are sufficient for current needs.
- Flat files are a reasonable method for storing baseline, incremental, and configuration data.

Man/Machine Interface (section VII)

- Applications are reasonably easy to develop.
- The FORTRAN interface to TOAST provides reasonable services.
- Menu Handler provides unique features for user interface in data validation, time and date fields, and scrolling regions within menus.
- A menu layout tool would be helpful.

Software Engineering (section VIII)

- The code is well written and well implemented.
- External documentation needs an index and more detail in tables of contents.
- The revision process needs changes to accommodate an increasing user base.

TOAST under X (section IX)

- The TOAST under X design is generally sound.
- Clocks and Discrete Data Displays (DDDs) will be difficult to implement.
- The TOAST Executive will port to X with minimal change.
- Over 90% of Menu Handler will need rewriting to utilize X.

The next sections of this report provide detail for the findings in the high level summary. Specific areas that are discussed are

V. Needs Analysis
VI. Technical Analysis
VII. Man/machine Interface
VIII. Software Engineering
IX. Future TOAST Migrations

V. Needs Analysis

Needs analysis examines the environmental constraints, user requirements, system constraints, and design philosophy to provide a basis for software evaluation. Developing an understanding of the tasks that the users of the system perform and the hardware, software and managerial environment in which they work equipped the evaluation team with a basis for judging the system's real worth -- its value to users in accomplishing their tasks more efficiently.

A. User Requirements

1. Evaluation Strategy

To perform a full task analysis, it is necessary to supplement information obtained through interviews with intensive onsite observation of workers carrying out their jobs. Such an analysis lay beyond the scope of the present evaluation effort. However, we did interview both real-time Flight Dynamics Officers (FDOs) and Orbit Flight Design users, as well as TOAST developers, managers, and orbit navigation users, and gained much useful insight into their practices and responsibilities. Subjects who were interviewed are listed in Figure 2 below:

Real-Time Users	Flight Design
Matt Abbott	Wayne Black
Roger Baletti	Phillip Gentry
Tim Brown	Bill Hollister
Mike Evans	
Keith Fletcher	Orbit Navigation
Mark Haynes	Malise Haynes
Mark Riggio	Tony Pocklington
Bill Tracy	
Management	TOAST Developers
Scott Anderson	Diane Campbell
Chirold Epp	Ken Wallis
Greg Oliver	
Kevin Williams	
Bruce Williamson	

Figure 2: TOAST Evaluation Interview Subjects

2. Task Analysis

In the following sections, we present a synopsis of our view of tasks and the requirements that carrying out those tasks presents for the TOAST system. Our analysis is based on the interview materials and orientation information supplied by NASA.

a. Tasks Common to FDOs and Orbit Flight Design Users

One manager estimated that at least 50% of the tasks performed by Orbit Flight Design and real-time FDOs are identical. Both sets of users need to integrate trajectories, target maneuvers, and meet constraints on the mission, vehicle, or payload, and landing opportunities.

b. Tasks of Orbit Flight Design

Pre-flight preparation begins about two years before the flight and continues until a few weeks before the flight. During that time, many aspects of the flight are defined including the precise nature of the payload and the launch window. To plan for the various contingencies that may develop, flight designers are responsible for modeling a number of different flight scenarios. As a result of this modeling, flight design produces a large amount of data that eventually will feed into the real-time operation.

The Flight Design users create a standard script that will take a flight from beginning to end, and then change pieces of it as they finish their analysis. For example, they may perform a Monte Carlo type analysis where they iteratively change certain parameters. To implement the scripting or *runstream* capability, flight designers run multiple versions of the same program one after the other. They have to have the ability to iteratively run programs and change parameters until they achieve a desired result.

As one designer described his task, he may need to

> Modify individual elements inside of a file, whatever data type they are whether they're real, double precision, time, whatever. I ... spool particular elements, I ... extract from a huge data file, I ... extract the sixth element all the way down this data file, take off one element by itself. Then I ... take that array, and I ... shove it into a graphics processor, and I ... plot the thing.

In the course of their work, flight designers need to carry out numerous types of computations involving different data types and processes. To do so efficiently, they need many specialized computational tools available to them.

Currently, Orbit flight designers use the Flight Design System (FDS) to carry out their tasks. The system runs on a set of Perkin Elmer workstations.

c. Tasks of Flight Dynamics Officers (FDOs)

The Flight Dynamics Officers (FDOs) are responsible for all aspects of the orbiter's trajectory during missions. They are supported by back room personnel during all aspects of flight -- ascent, orbit, descent. They work in the MCC and report to the flight director.

During the ascent phase of flight, according to one flight dynamics officer,

> FDO's react. ...When you're a FDO and you're on console, you have only one scenario to look at, and that's the one you're currently in, and you really can't make a scenario.

Within that scenario, they coordinate several different areas, which include retargeting maneuvers when the shuttle has deviated slightly from the nominal trajectory, managing the vehicle's center of gravity (CG), and computing alternative deployment opportunities, the next primary landing site, and landing opportunities.

In real-time, users "need a product and they need it fast". A mission has a duration of several days, but the FDO tasks may need to be accomplished in a matter of minutes or less. Therefore, they need to know exactly how to minimize time on a task, and rely on an expert knowledge of both the subject matter and the tools at their disposal.

3. Requirements

a. Requirements of Orbit Flight Designers

From our discussions with flight designers, it is clear that they need a number of capabilities. Foremost among these are the following:

1. An ability to generate runstreams.
2. Multiple input streams stored in the user's home directory.
3. A strong emphasis on file processing including the ability to have complete control over changing data, moving files around, and deleting files.
4. Use a large number of applications.
5. Online help.

b. FDO Requirements

From our discussions with the FDOs, it is clear that they need a number of capabilities. Foremost among these are the following:

1. Quick turnaround time on tasks.
2. Initialization files for menu configurations.
3. Sharing data and views of data among multiple users.

We should emphasize that task requirements need not be the same as design specification requirements. Task requirements are driven by the demands of the task to be done. Design specification requirements, on the other hand, may be driven by a number of other factors including economic feasibility, software and hardware environmental constraints, and user

preferences. In the present case, for example, the guidelines baselined in the *TOAST Requirements Document Volume I* specified that applications should have the look and feel of the MOC and that minimal keystrokes should be required to accomplish a command. In our view, while these may be preferred by users, they are not necessitated by the demands of the tasks themselves.

c. Additions to TOAST to Support Flight Design

As a result of discussions with the flight designers, the following additions to TOAST will be required to support the requirements of flight design.

1. Multiple input streams stored in the user's home directory.
2. Indexed documentation with a roadmap to the system libraries.
3. More systematic approach to support including bug fixes, training, and consulting.
4. Online help for users and applications programmers.

d. Summary

With the additions to TOAST listed in section c, TOAST should be able to support the requirements of the FDOs and Flight Design.

B. Environmental Constraints

TOAST operates on Masscomp 6600 computers in a Unix development environment and real-time environment in the MCC. The real-time version of TOAST runs under the Workstation Executive (WEX). Unisys provides system administration and software support for the Masscomp computers. Hardware support is provided by Bendix Corporation. TOAST and its applications are managed by NASA and Rockwell personnel. The complexity of a multi-contract environment, with different vendors responsible for different aspects of the operation, results in an extremely difficult development environment (section C). Further problems result from high level centralized decision making which may not be responsive to developer needs. For example, as the base for X Window development, the TOAST developers were given one version of the X Window System , but should be using a newer release (X11R4), which corrects many of the performance problems of the earlier version while also providing additional tools that would facilitate TOAST under X implementation.

The evaluation team encountered two problems that are viewed as system constraints to TOAST software development.

1. Workstation Executive (WEX)

WEX use in the MCC TOAST environment imposes artificial barriers for configuration and system management. It compels use of programming practice that is not standard such as using an automatic program to modify a password file. In other areas, WEX constrains normal Unix activities such as spinning off tasks, which could be simplified and improved upon if the TOAST code was not limited.

2. System management and configuration problems on the TOAST Development Masscomps.

We noted several problems in this area that warrant immediate management attention and corrective action:

a. Our programmers encountered severe environmental difficulty while writing applications on the TOAST developers' Masscomps due to system instability. During one site visit, the systems crashed 3 times in 4 hours while our programmers were trying to write code.

b. Hardware problems were worked around rather than fixed. Our programmer patched bad blocks on a disk so that an 800 megabyte disk could be used. We observed several problems with bad disks and memory problems that the TOAST developers have to live with.

c. We observed inconsistently installed or poorly configured software including mail and X. Some FORTRAN libraries on some machines were not installed in the location that the FORTRAN compiler expected to find them.

d. Lack of regular backups or capability for users to perform their own backups.

C. Design philosophy

This section is a review of TOAST as an application manager, which consists of executive-based design and structured display control. Executive-based design concerns the use of software that provides a context consistent interface to system managed resources and devices (see section 2 below). Structured display control packages present data in a controlled format, accept and validate user input, and provide a consistent interface for the process of acquiring and displaying data (see section 3). The discussions place the TOAST Executive and Menu Handler within a continua of competing COTS products available today. We also examine the Menu-Application-Display (MAD) model for dividing application functionality (see section 4) and compare baselined user requirements with design implementation (see section 5).

1. Role of an Application Manager

An Application Manager provides an isolation layer between the underlying Unix operating system and the series of user programs. It provides two main services:

1. A common interface to operating system functions with semantics appropriate for C or FORTRAN. This is called an *executive*.

2. A structured input and output package that can be utilized by user application programs.

Figure 3 below illustrates this relationship.

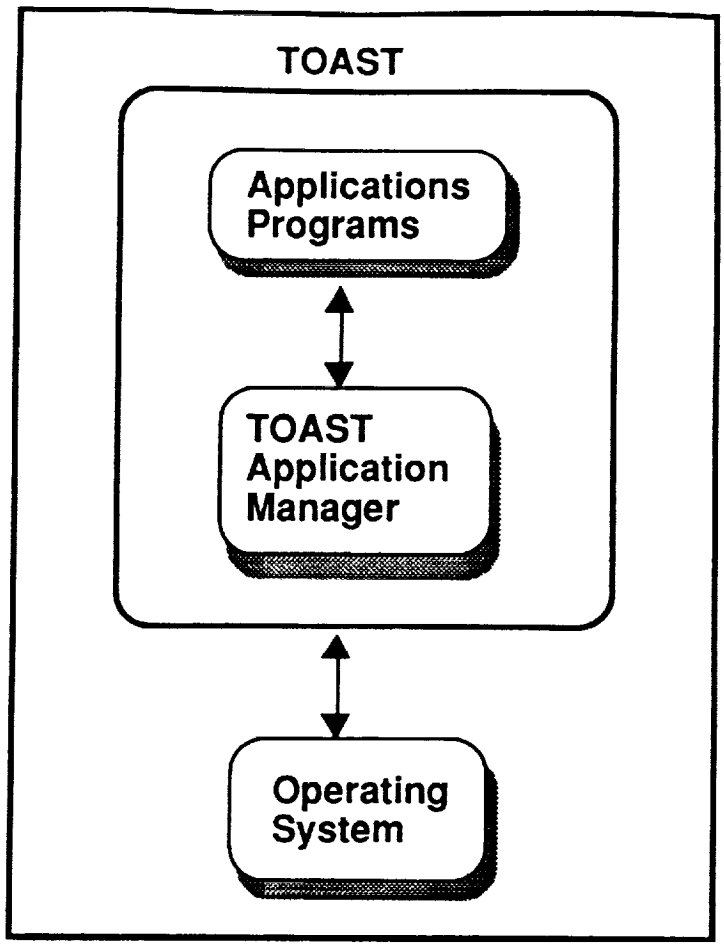


Figure 3: TOAST as Application Manager

2a. Executive-based Design

An executive is a set of functions that provide a context consistent interface to system managed resources and devices. Use of an executive reduces training time for programmers and permits previously implemented user programs to access services. In real-time, an executive interfaces to the operating system. The executive provides calls that are consistent with the conventions of a programming language and avoids the use of direct manipulation with machine registers or inline assembly code. In particular for TOAST, the Executive provides access to complex services from FORTRAN.

To compare the services of the TOAST Executive with COTS programs, we placed TOAST in a continuum of programs that provide a high level of application functionality and those that require a large amount of code to be written. At the highest end of the spectrum are experimental systems like ISIS (Cornell Theory Center) and Exodos (University of Oregon). Based on a high level operating system, these executives rearrange parameters to fit an underlying operating system convention. In these systems, an application program would use only a few lines of code to achieve a large amount of functionality. At the other extreme are systems like pSOS (Intel) and LynxOS (Lynx Computing Systems), which provide a minimum amount of functionality. These systems are based on low level operating systems (slightly above the device driver interaction level) and require application

programs to write a large amount of code to achieve reasonable functionality. Other programs are intermediate between the two extremes. TOAST falls into this middle range together with programs like the Unix *stdio* library (standard I/O) and SAS/FSP (Statistical Analysis System/Full Screen Product from the SAS Institute Inc.). This continuum is shown in Figure 4 below.

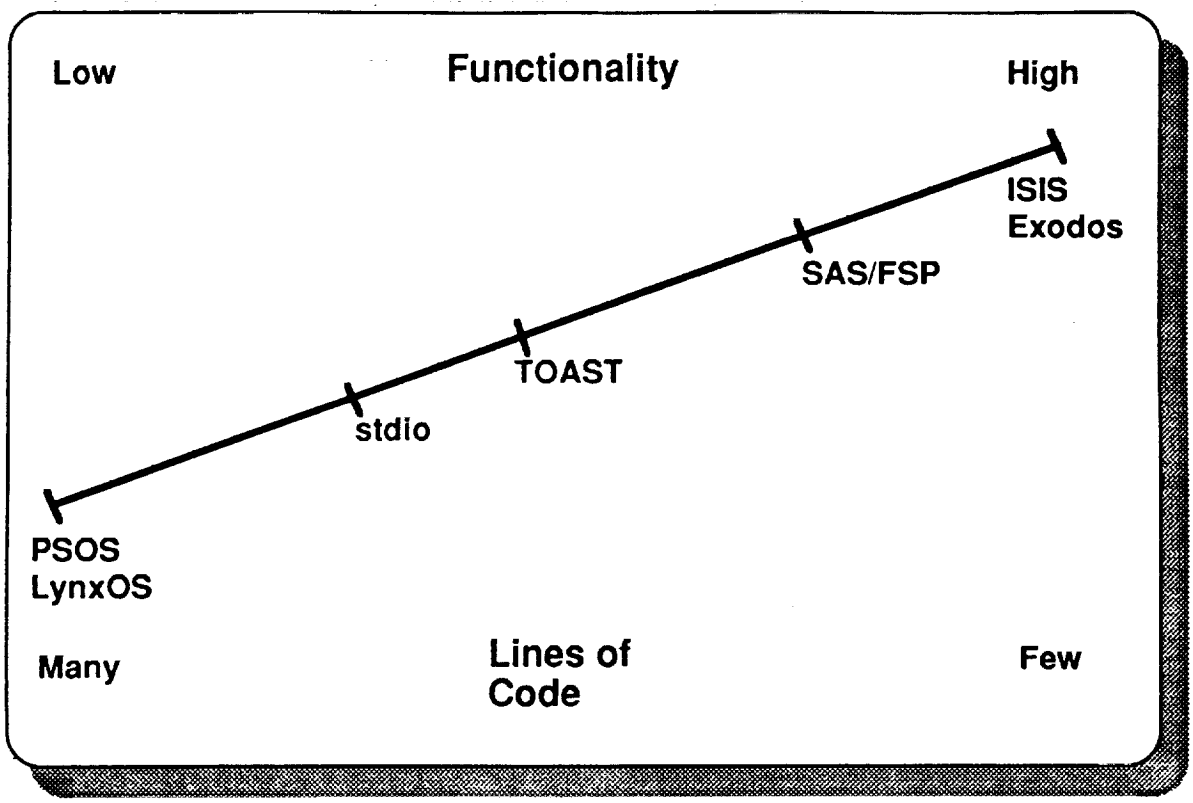


Figure 4: Continuum of System Executives

2b. Overall Evaluation of TOAST Executive

The current TOAST Executive is a reasonable approach.

3a. Structured Display Control

Structured display control packages present data in a controlled format, accept and validate user input, and provide a consistent interface for the process of acquiring and displaying data. Such a package must communicate directly with the operating system and allow user programs to invoke its services.

At the time that TOAST was written, COTS packages for structured display control under Unix did not exist, and a clearly accepted method for user interaction does not exist today. The *curses* package (standard Unix tool) provides some of the functionality, but does not

support input processing. Unix packages that provide full-screen input/output bypass the problem by using *curses* for output handling and writing their own input processing code.

3b. Continuum of User Interfaces

Structured display control systems range from artificial intelligence programs that will provide a DWIM ("do what I mean") interface to batch JCL systems. In between the two extremes are interactive interfaces to the operating system, prompt systems, text-based systems, and graphical user interface (GUI) packages . The range of services provided by these systems varies greatly. A diagram showing the types of systems and examples is shown in Figure 5. The current TOAST implementation and the planned TOAST under X version are placed within the continuum.

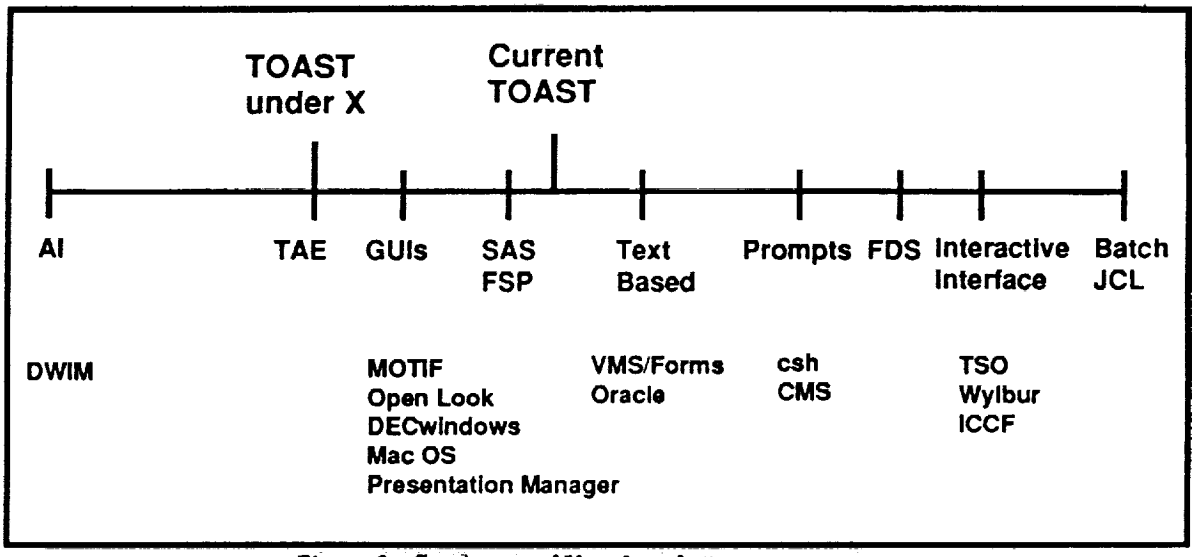


Figure 5: Continuum of User Interfaces

3c. Overall Evaluation of Menu Handler

To provide the structured display control services, the TOAST team developed Menu Handler. Similar to other packages, the TOAST Menu Handler program provides a structured approach to screen input and output and the capability of defining structures for displaying data and editing input. However, Menu Handler extends *curses* data types by supporting protected/unprotected fields and processing-required fields. Other than the *curses* package, no package equivalent to Menu Handler exists in the non-X Unix environment.

4a. Menu-Application-Display (MAD) Model

Having created the foundation for developing applications with the TOAST Executive and Menu Handler, the TOAST team adopted the Menu-Application-Display (MAD) model as a method for dividing program functionality into input, process, and output components. As shown in Figure 6 below, this model allows for replacement of input and output modules without necessitating changes to the application module.

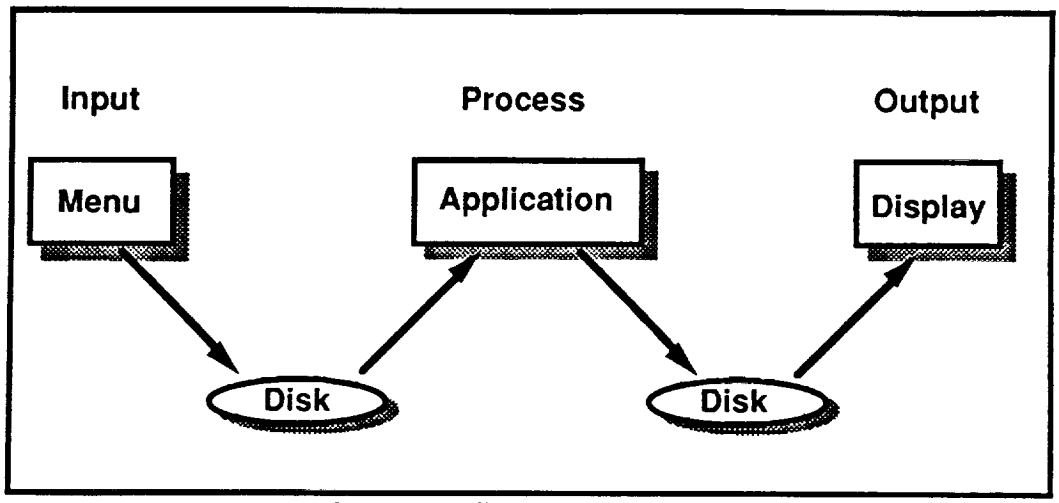


Figure 6: Menu-Application-Display Model

4b. Overall Evaluation of Menu-Application-Display (MAD) model

The MAD model permits independent verification and testing of each component module. It is flexible and portable, and will ease the planned TOAST migration into an X environment (section IX). The model provides the capability for using interfaces other than Menu Handler. Our conclusion is that use of this model supports standard software engineering practice and enhances the TOAST design.

5a. User Requirements versus Design Implementation

To compare user requirements against the implemented design, the evaluation team reviewed the *TOAST Requirements Document Volume I*, which was baselined on May 6, 1987. Pages 3 and 4 of that document list the guidelines for TOAST implementation:

> 1.4 Guidelines
>
> - The user interface is convenient, rapid, and user friendly. Nearly all TOAST commands are implemented via menu structure and function keys. Exceptions are clearly noted. A shorthand command input capability is also available. Menus have logical defaults. Cursor control is by arrow keys rather than Wordstar type controls.
>
> - Application formulations and capabilities default to MOC requirements. However, additional options are available in many cases. For example, the MOC Rendezvous Targeting Processor has restrictions which may never be fixed in the MOC, but which are removed in the MITS workstation.
>
> - LAN interfaces are via approved interface programs. The workstation software design does not preclude sending any valid MED commands to the MOC from the workstation. Near Real-Time Telemetry (NRT) commands are also supported.
>
> - Transportability of code is highly desirable. All programming is done in FORTRAN or C and all hardware and system dependent code is modularized for containment and is documented.
>
> - The system adheres to all security requirements. This includes requirements for converting the software from the black to the red machine.
>
> - Initialization files are supported. The user may create unique initialization files for analysis as well as mission support. Each of the Dual Operations sessions permits the initial user to select the appropriate initialization file from among those available. Logging on by flight ID alone forces the selection of the approved flight initialization file for that flight.
>
> - TOAST supports multiple terminals having displays controlled from one keyboard. The user is also protected from undesired output to his terminal. Conversely, any user is able to view data from any Session Data Area (SDA), but only the users of a specified SDA may write to that SDA.

5b. Overall Evaluation of User Requirements Implementation

The current and planned versions of TOAST implement the functional description in the baseline document. Planned releases, to be discussed in section IX, will implement Discrete Data Displays (DDDs), clocks, and automatic regeneration of displays for updated data. The team was not asked to address red versus black security issues.

Our overall evaluation is that the current and planned versions of TOAST implement the functions described in the baseline document.

In the following section, we present our technical analysis of the current TOAST implementation.

VI. Technical Analysis

In section VI, we present a technical analysis of TOAST including:

A. Fault tolerance
B. Portability
C. Comparison: Multi-level Menu System to COTS Application Manager
D. Comparison: Menu Handler to Text-based COTS Toolkits
E. Data Structures
F. Security

Data gathering was accomplished through reviewing code with TOAST developers, examining sample applications, running tests on the Masscomp systems, and performing outside research.

For each technical area, we present an overview of the area to be reviewed, summarize our evaluation strategy, discuss the current implementation in TOAST, list strengths and weaknesses of the implementation, present our recommendations, and summarize our findings.

A. Fault Tolerance

1. Task of Fault Tolerance

Fault tolerance measures the robustness of a system when subjected to unexpected system or configuration errors. Specifically, in evaluating fault tolerance capability of TOAST, we were concerned with how TOAST interacts with the Unix operating system to intercept and recover from problems.

2. Evaluation Strategy

Data for evaluating fault tolerance was gathered through experimentation with the TOAST environment on the TOAST developer's Masscomps and through reading code.
Some of the tests involved removing menu files, transposing characters in menu files, and changing bits in a TOAST executable to observe how the system responded.

3. Fault Tolerance in TOAST

The TOAST Executive and Menu Handler exhibited different behavior when subjected to serious errors.

The TOAST Executive provides a log entry if a process has terminated abnormally. As much of the context of the failure as can be determined superficially by portable Unix system calls is logged also. The process produces a core dump and exits.

Menu Handler does not cope with corrupt menu files other than refusing to initialize.

4. Strengths of Present Implementation

Based on our tests of the TOAST environment, we found that TOAST was not fault tolerant. However, the configuration management procedures in place (see section VIII. C) should prevent errors serious enough to cause Unix to terminate a process.

5. Weaknesses of Present Implementation

a. Parameter Verification

Many of the primitive routines deep within the Executive simply assume that their parameters are of the correct length and type. In general, good programmers will not make frequent mistakes of this type. However, the lack of argument checking can lead to memory overlays or allocation "leaks", which are very difficult to find and correct.

b. Menu Handler Recovery for Corrupt Menu Files

Menu Handler does not cope with corrupt menu files except by refusing to initialize. The process of loading all the requested menu definitions into in-core structures removes some of the dependence on correctness and presence of the menu definition files. However, it is fairly simple to corrupt the structure by attempting to load a menu definition containing a syntax error, as *load_menus* currently replaces any previously loaded set of menus with the ones loaded. If the load process fails, the structure is partially updated, leading to a non-deterministic state after a failed read.

c. Signal Handler Support in the Executive

The Executive provides no effective support for programs to establish signal handlers other than directly manipulating the underlying OS signal handling table via *signal()*. Admittedly, this is a difficult task for a multi-language toolkit, but the ability to detect, handle and recover from error conditions is an important feature that does not seem to be available in the current implementation, and will become yet more important as TOAST adds networking and IPC/RPC facilities in future releases.

6. Overall Evaluation of Strengths and Weaknesses

The TOAST Executive does not expand greatly on the standard Unix error handling for a serious process error. Parameter verification and recovery for corrupt menu files are serious problems that need to be addressed. Support for signal handlers in the Executive, while a useful feature, has the drawback of reducing portability.

7. Recommendation

All TOAST code should verify parameters.
Menu Handler should indicate the reason for failure.

8. Summary

If the enhancements discussed above are implemented, the Executive should be able to provide the facilities for user programs to deal gracefully with error conditions.

B. Portability

1. Task of Portability

A portable Unix application is a single version of a software package that will operate with minimal changes on any hardware platform that runs Unix.

2. Evaluation Strategy

To evaluate portability of the TOAST software, we reviewed the code and sample applications. We also ran the Unix utility, *lint*, on the TOAST software. *lint* is designed to detect features of C program files that are likely to be bugs, to be non-portable, or to be wasteful. It also performs strict type checking. Among the possible problems detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions with constant values. Function calls are checked for inconsistencies, such as calls to functions that return values in some places and not in others, functions called with varying numbers of arguments, function calls that pass arguments of a type other than the type the function expects to receive, functions whose values are not used, and calls to functions not returning values that use the non-existent return value of the function.

3. Portability in TOAST

According to the guidelines listed in the *TOAST Requirements Document Volume I*, "transportability of code is highly desirable". We found:

a. The Executive is portable.
b. Menu Handler will run on other Unix systems, but may not look right due to isolated terminal dependencies (color and fonts).

4. Strengths of Present Implementation

The Executive is portable.

5. Weaknesses of Present Implementation

Menu Handler will run on other Unix systems, but may not look right due to isolated terminal dependencies (color and fonts).

6. Recommendation

TOAST isolates WEX calls in the present Executive code. We recommend that this practice be extended to operating system dependencies as well.

7. Summary

Our overall finding is that the TOAST code is portable.

C. Comparison: Multi-Level Menu System to COTS Application Manager

1. Task of a Menu Application System

An application manager manipulates programs and data within an environment.

2. Evaluation Strategy

To evaluate the capabilities of the multi-level menu system, we compared it to the Visual shell from Berkeley, a text-based menu system.

3. Multi-level Menu System in TOAST

TOAST implements a tree structure menu system. The description of the menu system on page 12 of the *TOAST Requirements Document Volume I* is shown below.

> 2.3.1 Menus
>
> There are three levels of menus:
>
> Level 1 - This is the main menu whose choices represent a selection of processor categories.
>
> Level 2 - For each level 1 menu option, there is a level 2 menu containing all the processors available in that category.
>
> Level 3 - The level 3 menus are the input menus for the individual application programs. Each time the user calls up an input menu, the default values are displayed. The first time a menu is used, the defaults are zeroes and blanks. Some menu fields have non-zero initial defaults, and some menu fields have system defined, unchangeable defaults. Otherwise, the default values are the most recently saved values for that menu. Values for a menu are saved when the user presses either the [EXECUTE] or the [SAVE MENU] function key. These values are always stored in the user's SDA. When a level 3 menu is invoked, the user is placed in the input mode. In order to do anything else, the user must first exit the input mode. This is done via one of the system function keys.

4. Strengths of Present Implementation

The TOAST menu application is simple to operate and provides the ability to select an application rapidly without risk of selecting the wrong application. It provides an acceptable level of flexibility and modularity when compared to other text-based systems.

5. Weaknesses of Present Implementation

The tree structure of the TOAST application system is somewhat limiting. When the user needs to move between applications that are located in widely separated parts of the menu tree, it is somewhat tedious to move through the intervening levels of menus. In most cases within the current TOAST configuration, it is not too onerous because of the relatively small number of menu levels, but if the number of menu levels were expanded, it could become a severe liability.

6. Overall Evaluation of Strengths and Weaknesses

The menu structure works adequately for the current number of levels. However, if the number of levels expand, traversing the tree structure will become a problem.

7. Recommendation

A useful feature would be a "jump to the top of the menu tree" key to speed up motion through the tree. Additionally, each menu could be assigned a keyword that allows the user to jump directly to a particular menu by name, much as IBM's CICS allows invocation of a particular menu transaction by typing the transaction id assigned to that menu.

8. Summary

The TOAST program manager -- the multi-level menu system built on Menu Handler and the Executive services -- provides an acceptable level of flexibility and modularity in comparison to other text-based menu systems.

D. Comparison: Menu Handler to Text-based COTS Toolkits

1. Task of Forms Generation and Display Packages

Forms generation and display software addresses three categories of functions:

1. Functions to display data with various attributes such as inverse video or a "protected" attribute
2. Functions to read, interpret, and do some measure of validation on user input
3. Functions to manipulate areas of the screen, such as clearing the screen or setting up a scrolling region for list-oriented output.

Most forms packages also attempt to enforce a set of paradigms for the behavior of the user interface in terms of the behavior and appearance of screen objects such as function keys or menu selection, and the use of text and visual attributes such as color or highlighting.

2. Implementation in TOAST

Menu Handler provides forms generation and display for TOAST. According to the Introduction of the *TOAST Menu Handler Version 6.5 Programmer's Guide*,

> TOAST uses a generic user interface package (Menu Handler) for all data entry menus, option selection menus, function key definition displays, and tabular displays. The TOAST Menu Handler performs all of these functions and provides a common look and feel across all menus and displays. Additionally, Menu Handler provides those programs utilizing it with the maximum amount of hardware independence possible. Menu Handler provides only an extremely limited graphics capabilities: Greek text and Line Drawing text fonts.

As described in the *TOAST Menu Handler V6.5 User's Guide for the Interactive User*, Menu Handler implements CHOICE menus that present a list of options and FORM menus that resemble a questionnaire. A user can access more options or data than can be displayed on the screen simultaneously through the use of scrolling menus. Field types can be display only, user accessible/modifiable, or processing required. Data types permitted include string, enumerated, integer, real, latitude, longitude, Unix path, Unix file, time, and wind formats.

3. Evaluation Strategy

At present, no clearly superior textual interface package is supported on a wide range of Unix platforms and programming environments. Several approaches to the problem exist in commercial software, but none have the widespread acceptance within the Unix user community or the development community to constitute a significant segment of the market. Given the lack of a Unix standard, our evaluation included products from outside the Unix environment that provided equivalent functionality.

We compared Menu Handler to IBM's SAA applications programming interface, the VMS/FORMS package for DEC's VAX/VMS systems, and the RXFS package developed for IBM VM/CMS systems. SAA and VMS/FORMS are widely accepted within the industry as strategic choices for textual user interfaces; RXFS is a user-developed system that deals with many of the same issues as Menu Handler.

4. Strengths of Present Implementation

The data display and editing capabilities of Menu Handler are much better developed than the capabilities provided by the commercial packages. The IBM-based products came the closest to providing equivalent functionality by providing a number of fairly sophisticated numerical editing facilities and display attributes (given a terminal capable of 3279 DFT or extended datastream support), including programmed symbol sets and good color support. However, the enumerated field types and the flexibility of the Menu Handler date and time field types represent a facility that would be difficult to accomplish using either package. The DEC package fared rather poorly in this area, due to the somewhat limited support of special character sets provided by the VT100. The VT2xx and VT3xx terminals available from DEC alleviate some of these problems, but the number of special characters and attributes permitted is still limited. (Of some interest is the fact that both the IBM and DEC solutions require specific hardware devices, and in the case of the IBM solution, much of the data verification is in fact implemented by the hardware.)

In terms of generalized data input, Menu Handler provides a level of functionality comparable to that found in other packages. All packages provided a range of functions for protecting and unprotecting data, highlighting or dimming areas of the screen on a programmatic basis, as well as the use of multiple symbol sets and colors on the screen, subject to hardware constraints. In addition, Menu Handler provides a number of unique data types and capabilities, written to satisfy user needs, which other packages do not provide.

5. Weaknesses of Present Implementation

The screen control functions provided by Menu Handler are somewhat more limited than the other packages in that only clearing of the entire screen page is supported as a primitive, and it is difficult to establish an independently scrolling region without defining it as part of a displayed menu structure. VMS/FORMS stands out as the most developed package in this regard, as the page-oriented hardware structure imposed by the IBM solution makes asynchronous screen activity difficult.

6. Overall Evaluation of Strengths and Weaknesses

Menu Handler functions adequately. It excels in the display and input validation sections when compared with similar packages. The screen manipulation routines in Menu Handler are somewhat limited in comparison with the products evaluated, but sufficient functionality to remedy the defects can be easily added.

7. Recommendation

Functions to define an arbitrary scrolling region and partial screen erase capabilities (preferably arbitrary rectangular regions) would enhance the facilities of Menu Handler greatly.

8. Summary

Menu Handler held its own very well in comparison with two of the major vendors' efforts and a screen handling package written and enhanced by a member of the review team.

E. Data Structures

1. Task of Data Structures

A data structure organizes data elements into functional units.

2. Data Structures in TOAST

The TOAST database is a directory of flat files that include Executive applications, data lists, Executive menu definitions, session data areas, baseline data, user save areas, and position data.

The data structures in TOAST are implemented using flat files.

3. Evaluation Strategy

To evaluate data structures, the team compared flat file design of the current TOAST implementation to hierarchical database design.

4. Strengths of Present Implementation

We found the flat file approach is flexible, portable, and independent of emerging standards. It maximizes recoverability, and does not require a database administrator.

5. Weaknesses of Present Implementation

When compared to hierarchical database structures, flat files lose some functionality in terms of inheritance, database linking, record linking, and multiple table lookups. Inheritance (chaining similar data values in fields via pointers) reduces disk storage requirements. Linking and multiple table lookups reduce the number of queries into the index system. Use of these features promotes resource conservation by reducing requirements for disk storage, memory, and access time, and generally improves performance on a real-time system.

6. Recommendation

In general, we found that flat files are appropriate for current TOAST use. When TOAST migrates to X with a larger user base, a hierarchical database approach could be integrated into the TOAST Resource Manager.

7. Summary

In general, we found that flat files are appropriate for current TOAST use.

F. Security

1. Task of Security

The security of a Unix system is measured by system vulnerability to the commonly known weaknesses present in almost any software operating on a Unix platform. Most of these vulnerabilities center on how the system responds to attacks from outside the environment -- usually via network connections -- but internal issues such as password selection and encryption methods also play a role.

2. Evaluation Strategy

We evaluated the TOAST development system for security problems. We based our evaluation on the recent SRI International report, "Improving the Security of your Unix System" (David Curry, 1990). We used the checklist in that document to suggest possible Unix system weaknesses and then designed tests to determine TOAST vulnerability in those areas.

We presented our findings in briefings to NASA management and system security officials.

We did not investigate WEX security considerations.

3. Security in TOAST

The TOAST Executive provides a rudimentary access control mechanism in addition to the standard Unix system access control facilities, assisted and/or hindered by information provided by the WEX shell. TOAST security is based on information resident in an auxiliary file ($TOAST/access_list) containing a list of Unix userids and associated flight positions, flight numbers, cycles within a particular flight, and flags indicating that the indicated user may act as the "lead" for a particular flight and cycle. If this file does not exist, no access control to the TOAST system is performed.

All other access control is implemented by relying on the security of the underlying Unix system to prevent hostile users from gaining entry or damaging the system configuration.

4. Strengths of Present Implementation

We found security to be weak in the development environment.

5. Weaknesses of Present TOAST Implementation

While in general relying on the security of the underlying Unix system is a reasonable approach to data integrity and application access control, it unfortunately exposes the TOAST system to any weaknesses present in the underlying system, including null passwords, defects in system and network daemons, and other systematic efforts to circumvent the access control restrictions.

The internal requirement that TOAST run as a user process without *setuid()* privileges also effectively negates the Unix permission scheme, as all files are owned by the same Unix uid.

Given these two problems and access to the system supporting the TOAST Executive, a reasonably knowledgeable and persistent Unix user could infiltrate the TOAST system and gain unauthorized access to programs and data without excessive effort or time.

6. Weaknesses of the Environment

The Masscomp networking software combines 4.1 BSD and 4.2 BSD network semantics. This implementation includes many of the bugs exploited by recent network "virus" programs. Further discussion of these software flaws is inappropriate in a public medium, but the topic was discussed extensively in the TOAST security briefings.

7. Overall Evaluation of Security Strengths and Weaknesses

TOAST relies heavily on the Unix operating system to provide security. As currently implemented within the NASA environment, Unix operating systems are not secure.

8. Recommendation

Code is present within the Executive to generate Unix userids and uids dynamically if *setuid()* privileges are available at run time. This code generates a userid and uid dynamically upon login and uses the generated values for the duration of the terminal session. By generating these values dynamically -- not from /etc/passwd -- a hostile user gains little benefit from infiltrating the underlying system, as the information needed to impersonate a legitimate user of the system is simply not present outside the TOAST environment. Use of these generated uids also re-enables the Unix permission facilities, as each user now possesses a unique uid that can be checked by the Unix kernel. Enabling these facilities would be very beneficial to the stability and security of the TOAST environment.

9. Summary

Security is an environmental problem. TOAST can only be as secure as the environment in which it operates.

VII. Man/Machine Interface

The discussion of man/machine interface concentrates on the the applications programming environment at the request of the NASA project sponsors. Some general notes on Menu Handler's user interface are included, but a detailed examination of the user interfaces of the applications themselves was not performed.

A. Applications Programming Environment

1. Evaluation Strategy

To evaluate the TOAST applications programming environment, two members of the evaluation team wrote TOAST applications. Recognizing that programming teams consist of members with varying levels of experience, we used an entry-level programmer and a senior programmer to write the applications. The entry-level programmer converted an existing public domain roladex program in C to run as a TOAST application. The senior programmer wrote 5 original FORTRAN TOAST applications that displayed text, displayed a menu, displayed all fields, displayed a dynamically modifiable menu on the screen, and converted date fields to Julian dates.

To accomplish the task, both programmers read existing documentation, used the *vi* editor to design a Menu Handler form, defined the structure and return values for function keys, and wrote the do_menu loop. To evaluate the application writing effort, our programmers kept detailed notes on the application programming process with special attention to problems that they encountered. Other members of the evaluation team debriefed the programmers and summarized their findings.

The senior programmer took 2 hours to write the FORTRAN applications. The junior programmer, who was unfamiliar with the vi editor, took 16 hours to create a successful application. However, he believed that it would take him less than 8 hours to write a next application.

2. Strengths of the Programming Environment

a. Building Menu Handler applications is relatively easy. Menu Handler requires only a small number of functions to build an application as compared to the large number of complicated calls required for other packages.

b. The FORTRAN interface to TOAST provides reasonable services.

c. The more form-oriented the application is the better TOAST will service it. Menu Handler is especially geared to data entry applications requiring extensive type checking.

d. Use of color was very straightforward. It was straightforward to put text on the screen.

3. Weaknesses of the Programming Environment

a. Documentation was poorly organized.

b. Lack of a menu layout program is a serious drawback. One programmer had some colons missing in the menu definition that caused undesired results when the program ran. Such formatting details are better handled by a program.

4. Recommendation

A layout tool for menu design would speed the application programming process. To design the menus manually, the evaluation programmers used *vi* to draw the menu, and then used a *vi* function to get the x and y coordinates for each label. A layout editor that could be used to drag icons like boxes and buttons and create the menu definition file would be very helpful.

A document that lists the location of the include libraries and compilation order would be helpful. The information was available from the TOAST developers, but was not available in a written form that could be easily dispersed.

An index in Menu Handler documentation and a more detailed table of contents would be helpful. While all the information necessary for programming the applications was available, it was difficult to find in the current manuals.

B. User Interface Capabilities

Following the instructions of the NASA project sponsors, a detailed examination of user interface capabilities was not performed. Our general impression is that Menu Handler's user interface provides several useful capabilities. It protects users from common mistakes, traps common errors, and performs data validation well. It has several unique features that are not found in other similar programs including extended time and date fields and editable fields in scrolling regions within menus.

VIII. Software Engineering

The review of software engineering examines coding practices (section A), documentation, (section B), and revision practices (section C). Information was gathered through reviewing the code, reading available documentation, and conducting interviews.

Our investigation showed that the code is well written, the documentation needs work, and the formal revision process is thorough, but time consuming. Methods for tracking problems, changing software, and notifying users of system updates must be rethought to accommodate an expanding user base.

A. Coding Practices

1. Evaluation Strategy

The TOAST developers provided Box source listings for the TOAST software. Initially, several members of the evaluation team, working individually, reviewed the code for standard coding practices. They then met as a group to compare and combine their individual results.

2. Evaluation Criteria

The evaluation team applied standard criteria for evaluating internal software documentation and coding practice. The specific criteria invoked are listed below.

a. Readability and Layout
b. Use of a Comments Prologue in Subroutines
c. Identifier and File Naming Conventions
d. Extent of Structured Code Practices
e. Top down Design

(Software Engineering, Ian Sommerville, 1989.)

3. Implementation in TOAST Software

a. Readability

The code is easy to read and follow. The level of readability is greatly enhanced through the use of a consistent prologue.

b. Use of a Comments Prologue in Subroutines

The prologue covers the areas shown in Figure 7.

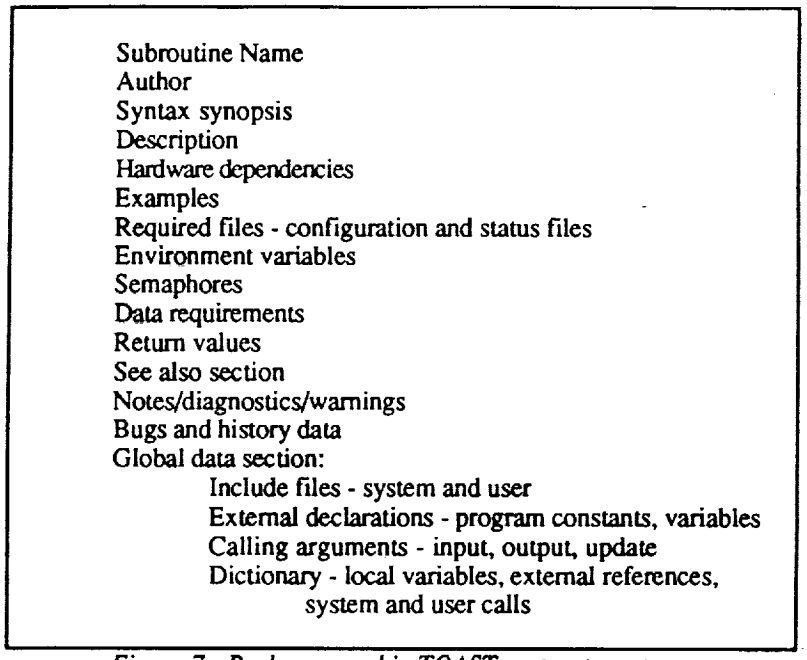


Figure 7: Prologue used in TOAST programs

The consistency in the prologue makes the code easier to follow. In some of the older routines, the programmers did not fill in all the fields. However, in several cases, the comment prologue is much longer than the body of the code for a particular routine.

c. Identifier and File Naming Conventions

Identifiers use the MH or TE prefix to identify constant files and variables relating to Menu Handler (MH) or the TOAST Executive (TE). Constants follow standard C programming practice with names in capital letters. In the use of local variables, many of the routines use the same names.

d. Extent of Structured Code Practices

Structured coding practices are enhanced through the use of the Box program. This program enforces a consistent approach to code development. Use of Box should be continued.

e. Top down Design

TOAST is extremely modular. The program may have started as a planned design, but many of the unused code portions indicate use of a bottom up approach as well.

4. Recommendation

a. General Recommendations

While the standard of programming is very high in the TOAST code, the areas of program history and data checking were identified for improvement.

The development team needs to establish a consistent practice of identifying in the history section the person who makes program changes. A date trail on the lines that are changed would be useful although difficult to implement with the current system tools. If the *emacs* editor becomes available as a systems tool, a template could be developed that would automatically leave an audit trail.

We must also point out a failure to test data rigorously before it is used internal to the Executive. When interviewed, the programmers stated that they rely on Menu Handler's data checking capabilities to trap problems and maintain good data. They use rigorous test procedures during the development stage to detect other problems and test data in the code where it is "important". We note that verifying data arguments returned from a function is a standard programming practice that should be employed where practical. If a corrupted array is passed into a routine, it can cause errors that may cause the program or TOAST to abort.

b. Specific Recommendations

A syntax codebook or a quick reference card for the Executive and Menu Handler would be useful.

A reference file of local variable names might be a useful tool for the future to help maintain consistency.

Code debugging will be aided when the flexible debug routine under development becomes available for production use. The ability to select different areas for debugging rather than relying on a compiler directive should speed the debugging and verification process.

5. Summary

The overall internal documentation and coding practices evident in the TOAST software is consistent with industry standards.

B. Documentation Review

1. Evaluation Strategy

The evaluation team reviewed copies of the Menu Handler V6.5 documentation and several documents that were in preliminary stages of work and were not appropriate for detailed comment. These documents included

1. TOAST Volume I: Executive - draft 11/27/89
2. TOAST Requirements Document, Volume 1 - section on Database Manipulation - draft 11/04/88
3. TOAST Requirements Document Volume I. Executive - baselined May 6, 1987
4. TOAST Executive Program Guide for TOAST Version 5.2 - 3/29/90

Specific evaluation comments focus on the Menu Handler documentation.

2. Evaluation Criteria

The evaluation team applied standard criteria for evaluating documentation. The specific criteria invoked are listed below.

a. Levels of documentation for differing skill levels
b. Ease of use
c. Task guide organization
d. Simple writing style
e. Use of examples
f. Meaningful and complete sample sessions
g. Use of summaries
h. Table of contents, index, glossary
i. List of error messages

(How to Write Computer Documentation for Users, Susan Grimm, 1987)

3. Implementation of Menu Handler Documentation

The Menu Handler Documentation is implemented as 3 manuals:

TOAST Menu Handler Version 6.5 User's Guide for the Interactive User
TOAST Menu Handler Version 6.5 User's Guide for the Menu Programmer
TOAST Menu Handler Version 6.5 Programmer's Guide

4. Strengths of Menu Handler Documentation

The documentation for Menu Handler V6.5 meets or surpasses several of the external documentation criteria. It provides 3 levels of information with a guide for end users, a user's guide for programmers, and a detailed programmer's guide. The writing style is clear and easy to understand. Diagrams are useful and help clarify the data structures.

5. Weaknesses of Menu Handler Documentation

Problem areas for the documentation set include the lack of an index and page numbers, the need for more examples, and lists of error messages. The absence of alternate paths for accessing the information makes it difficult for new applications programmers to find the information they need. This point was documented in the debriefing of two TOAST evaluation team members who wrote sample applications. All the information that they needed is in the documentation, it is just hard to access. Adding page numbers and an index and providing a more detailed table of contents will greatly improve the situation.

Another area for improvement is in the use of examples. Examples of subroutine calls as well as simple and more complex sample applications would aid the new applications programmer. The use of templates in the *User's Guide for the Menu Programmer* is a good start. However, a small application would also be very useful.

6. Recommendation

a. General Recommendations

The overall utility of a system depends heavily on its documentation. This is especially true in the case of products that have an expanding user community. We recommend that completing documentation external to the program source be viewed as a high priority item and that serious thought be given to providing online help capabilities.

b. Specific Recommendations

For the Menu Handler documentation,

- An index should be added to each manual.

- Examples of subroutine calls as well as simple and more complex sample applications should be added to the documentation.

- A reference card that lists all the Menu Handler calls and an appendix containing error messages should be created.

Although still in draft form, the Executive Program Guide gives an excellent overview of the TOAST Executive. The evaluation team would like to encourage the completion of this document and suggest creation of a similar guide for Menu Handler.

7. Summary

The programmers have demonstrated the ability to write good external documentation with the Menu Handler Documentation and the TOAST Executive Program Guide. A high priority should be assigned to completing the draft documents.

C. Revision Practices

1. Evaluation Strategy

To gather information on procedures and policies for revising software, we interviewed managers, users, and developers of TOAST. With management, we asked questions designed to look at maintenance quality control, the software change boards, the roles of developers and users in the software boards, review procedures for software and documentation, frequency of review, frequency of implementing changes, transition procedures, user acceptance testing, and procedures for centralized change management.

With users, we examined the process for submitting problems and bug reports, the support infrastructure for implementing changes and informing users about changes, procedures for tracking bugs, and tools for recording bugs.

With the developers, we reviewed software transition procedures, backups of old releases, backward compatibility issues, and use of source code tracking software.

2. Evaluation Criteria

We evaluated the revision procedures based on industry practice and our experience with providing services for a large user base.

3. The Revision Process for TOAST

a. Software Revision

Requests for changes to TOAST may be generated by users or developers. We found that most requests are made by the developers. Change requests (CR's) are made informally or formally. The informal approach involves contacting one of the developers. A more formal approach is submission of a discrepancy report (DR).

When a developer recognizes that code must be corrected, a DR is presented to the Orbit Design Panel (ODP), which approves all changes to the TOAST software. The necessary changes are incorporated into a new TOAST delivery. For each of the TOAST applications, a user is responsible for the test plan and verification. That application is tested and verified by that user when the application appears in a new delivery.

Test documents, internal testing, and test plans are kept by the TOAST development team. Both bug fixes and design changes are noted in separate history files for each program module.

A summary of all submitted DR's and CR's is made and can be reviewed by the appropriate group of the ODP. The summary includes the status of each request, the date that it was closed out, and the delivery that it will be a part of.

b. Software Control

Changes to the existing TOAST software must be approved by the Orbit Design Panel (ODP). Once the panel has approved proposed changes, the Configuration Control Board (CCB) must certify that the test plans generate the appropriate test results. This board must certify the changes to the software before it is used as flight software.

The ODP was originally the TOAST Working Group (TWG). As flight designers were included, the name was changed to be more inclusive. The ODP now consists of three subgroups: Flight Design, Real-time, and Executive.

c. Notification of Changes

When a change to the TOAST software is made, applications that are affected by the change also will need to be revised. This generates a "ripple" effect through TOAST and its applications. TOAST does not maintain backward compatibility with prior releases when changes are made, so it is very important that all affected software is pinpointed and that the responsible programmers are notified. Currently, no formal means exists to direct applications programmers to make changes in their code to conform to the changes in TOAST. The mechanism used for notification of system updates is word of mouth.

4. Problem Areas

The three part organization of the ODP minimizes the meeting time impact for the user segments, but creates a problem in that no one group is responsible for all of TOAST. Lack of a central coordinator causes problems in follow up during the implementation, delivery, and verification phases.

In addition, there is no electronically accessed centralized database for tracking change requests.

Lastly, word of mouth notification for pending software changes is clearly inadequate.

5. Recommendation

The TOAST project needs a central administrator to coordinate design changes, schedules, verification, and followup.

The TOAST project needs an online problem tracking system to log and track bugs and change requests.

All involved programmers should be notified of pending changes via electronic mail or a newsletter. As the user base expands in size, management of the notification process can become a very large problem.

6. Summary

Communications is the key to implementation of successful revision practices. The involvement of users and developers in the process is a good management practice that helps to ensure that the product meets user requirements and minimize code rewrite due to miscommunications. We found that the formal revision process is thorough, but time consuming. The process for reporting and correcting bugs and for making design changes will need to change as the user base expands.

IX. Future TOAST Migrations

The TOAST implementation team is working currently to integrate TOAST into the X Window System. They developed a set of working notes that appear as the *TOAST under X* class, dated January 1990. The evaluation team was asked to study the TOAST under X plans and recommend potential improvements.

Section A examines the plans for TOAST under X. It gives a background on the X Window System, discusses the planned implementations, examines problems, recommends some possible solutions, and provides an estimate of work involved in the migration process. Section B provides an assessment of the expandability of TOAST.

A. TOAST Under X

1. Evaluation Strategy

We reviewed the planned implementation as documented in the *TOAST under X* class and as presented in discussions with TOAST developers. No existing TOAST code employs the design features presented in this section.

2. The X Window System

The X Window System is a platform-independent windowing system developed by MIT as part of Project Athena. Industry and academia joined together to create the X Consortium, a vendor-independent research organization located at MIT that was created to develop the X standard.

The X window system offers several enhancements to both the software and the operating environment. This approach is:

a. Platform-independent user interface and user interaction paradigm.
b. Portable. Code can produce identical results on a wide range of platforms.
c. A flexible interface for displaying textual and graphic information.
d. A strong base for more structured graphical tools.

We will discuss each of these features below.

Platform-independent user interface and user interaction paradigm.

The X line protocol used to communicate from a client program implementing an application program to the X display server software controlling the actual hardware screen, keyboard and pointer device(s) is rigidly specified by the X Consortium. It builds an abstract model of a raster display with an arbitrarily fixed number of bit planes, a keyboard, and one or more pointer devices and provides a set of image primitives (point, line, polyline, etc.) and image transformation operations independent of the actual hardware or operating system software involved. Any client program employing the X protocol can interoperate directly with any X server software and accomplish identical operations. The X window system has become the de facto industry standard for graphical user interfaces (GUIs), and represents a significant direction in compatibility and interoperability with heterogeneous networks of workstations.

Portable code that can produce identical results on a wide range of platforms.

The X libraries have a standardized set of function bindings for the C language, with a limited FORTRAN binding under discussion (see section 4e). X sessions consist of the client generating a stream of imaging transactions on the abstract display device provided by the X window system and allowing the X server software in control of the real hardware to make the necessary translations to perform the correct action on the local hardware or to substitute the closest corresponding action (e.g., requesting color on a monochrome display device will result in gray-scale output if possible, otherwise the server will use white and black). The abstraction layer insulates the client application from needing any specific information about the hardware used to interact with it.

Flexible interface for displaying textual and graphic information.

X is a raster-based system with some structured operations for procedurally defined objects and compound objects such as polylines and geometric shapes. Graphics and text can be freely intermixed, as displayed text is a special case of high-speed block raster operations. Multiple fonts and symbol sets can be displayed on any X server device. Font glyphs are defined using a procedural language and can be generated at different resolutions and sizes to suit the displaying system.

Strong base for more structured graphical tools.

X provides a strong underlying layer for the construction of more application-oriented graphical objects and toolkits and is, in fact, the underlying transport of several commercially available implementations of the PHIGS and GKS. Programs currently using the Programmer's Hierarchical Interactive Graphics System (PHIGS) or the Graphics Kernel System (GKS) could be integrated easily as part of a transition methodology, or as a final goal using an implementation of PHIGS layered over X.

Each of these features provides an important enhancement to the current TOAST implementation. Below, we discuss the TOAST Development Team's plans for TOAST under X.

3. The Plan for TOAST under X

The TOAST software is workstation software that was developed for the trajectory flight controllers. It consists of an Executive, which provides an operating environment and coordinates user applications, and Applications, which are user programs. It provides the capability for several users to access TOAST simultaneously (multi-user), more than one process to execute simultaneously (multi-tasking), TOAST users to access multiple independent concurrent sessions (multi-session), and for users supporting more than one flight control position to simultaneously use TOAST (multi-position). Transparent cooperative processing (multi-machine), where users access resources across a network, is a future goal.

Executive Applications interact directly with users to provide requested services. The current version of TOAST implements the TOAST Application Manager, the database manager, session state, delogging, command line, and calculator. The X implementation of TOAST proposes implementing several features that are baselined in the *TOAST Requirements Document Volume I*, specifically:

- Clocks - system display of several time of day and user timer clocks
- Discrete Data Displays (DDDs) - event status lights
- Status displays - to display the status of the TOAST system processors, the user's session, and selected processors
- "Lead" user functions - provides mechanism for position "leads" to maintain baseline data and user access

Other system enhancements proposed in the plans for TOAST under X include:

- Active Window Display - identifies active TOAST windows and status
- Mouse help display - provides help on using the mouse in various windows
- TOAST Resource Manager - provides data file access coordination, resource sharing for applications, and event notification for processes.

The TOAST under X plan proposes migrating the MAD design model. By using this conceptual model, adding an X-based user interface to an existing program would be non-intrusive, as it would involve only recoding the input and output pieces of the applications. The code performing the actual computations need not know whether the older text-based interface or the X-based interface is in use.

Not all of the implementation plans are documented in the *Toast under X* class notes. Many of the implementation details being considered are still in the discussion stages. Our understanding of the implementation plans is thus largely drawn from extensive conversations with the TOAST developers. In the discussion that follows, we focus on implementation issues, which in our opinion, may present difficulties.

4. Discussion of Implementation Plans

The implementation plans present levels of difficulty that range from features that can be implemented using existing X tools to features that will require extensive effort. Our recommendations (section 5) for possible implementation follow the discussion of the difficulties.

a. Features that can be Implemented using Existing X tools

The TOAST under X documentation presents a number of features that should be fairly simple to implement using tools provided by the X programming community or modification of source code provided on the MIT X distribution tape. Figure 8 shows the features that can be easily implemented using existing tools:

Feature	X tool
Active window display	Icon Manager display in X11R4 *twm*
Mouse help display	*xman*
Calculator	*xcalc* with modifications for user-defined constants.
Log window	A short program using the AsciiText file browser widget
Status displays	A short program with a graphic indicator object
Session state indicator (public/private)	A short program with an on/off radio button indicator to toggle state

Figure 8: Features that can be Implemented using Existing X Tools

b. Clocks and DDDs

Two features will be very difficult to implement, especially in heterogeneous multi-machine networks: globally synchronized clocks and arbitrary event notification (DDDs). Essentially, these two aspects involve the development of network services that are not currently part of the TCP or OSI specification suite, and thus represent a significant development effort. While the research community is investigating these problems, production implementation as standard network services is still some time in the future. At this stage, the Network Time Protocol (University of Maryland) and Zephyr (MIT Project Athena) may offer possible choices, but further evaluation is required.

c. Single Window

The industry standard calls for the user interaction model to be a window per application managed by a standard window manager such as *twm* or *gwm*. The present TOAST under X plans suggest on operation mode in which the TOAST Executive create a single large window and perform its own window and screen management within the bounds of the base window created by the TOAST Executive.

d. Function Keys

Function key support as presented in the document is more of an aesthetic problem than a real technical setback. Implementing function keys as a separate window mapped by the Executive will generate significant flicker when a new application is brought to the front and its key set is activated.

e. TOAST Resource Manager

In general, we found the TOAST Resource Manager (TRM) specification to be a reasonable set of services with only one possible problem: the current event queuing system is susceptible to a runaway process flooding the queue with high priority requests and thus locking out lower priority status and query requests. This condition, and indeed, good performance of the Resource Manager in a high-volume multi-machine network, will require a modified algorithm to ensure that lockout is difficult or impossible. One possible solution is a multiple bucket time-slicing algorithm to ensure that for time quanta *t*, a fixed (but tunable) number of lower-priority requests are processed after each *n* high-priority requests. The current design does not allow lower-priority requests to change the state of a managed resource, thus removing the possibility of a race condition within a quanta.

An area that is not currently addressed within the TRM design is expansion to multiple machine queues for shared resources such as telemetry data acquisition devices or special instrumentation. The TRM specification provides no mechanism to transfer control of a locked resource to a backup copy in case of primary failure, nor any method of informing lock holders of the failure and the transfer. The solution is somewhat dependent on the arbitrary event protocol (AEP) used for the DDDs, thus the design of the AEP must take this problem into account.

5. Recommendation

a. Use X11R4 Software for X Window Implementation

Many tools are provided on the X distribution tape, and others are available from the X user community and the public domain. We strongly suggest that NASA obtain and use the source on the X11 release 4 tape as the basis for any further implementation, even though the current implementations provided by the current hardware vendors implement earlier releases of the X environment. In most cases, the source will work with minor modifications to accommodate minor changes in the X function bindings between releases.

b. Use an existing X window manager and *xpseudoroot*

The current design proposes a rewrite of tools such as a window manager and menu system that could be constructed using available X tools. A more consistent interaction paradigm is to implement each TOAST application's input and output sections as X clients that interact with a standard window manager, perhaps implemented in place of the current WEX window manager, or using a phantom X display generated by the *xpseudoroot* program. (A phantom display provides a simulated display that exhibits all the properties of an X server interacting with real hardware. This includes the ability to use a separate window manager.)

Much of the functionality described in the user interaction portion of the specification is essentially the function of a window manager program (i.e., moving/resizing windows, the current active window display, display object help, etc.). Several good window managers are already available within the X community and represent years of collective effort that cannot and should not be duplicated lightly. We feel that the X11 release 4 of *twm* (the Tabbed Window Manager) can be configured to perform the required functions with little or no enhancement to the window manager code.

Coexistence with WEX presents a somewhat more difficult problem. However, using *xpseudoroot* allows coexistence with WEX by configuring the standard environment to allow the WEX window manager to control the real display and allowing the TOAST Executive to create a number of phantom displays. Windows created by TOAST applications would appear within the window representing a phantom display and would be managed by an instantiation of *twm* configured to manage that phantom display.

d. Plan to take advantage of multiple windowing capabilities.

The specification describes moving a heavily terminal oriented design into a workstation environment without modifications to take better advantage of the facilities provided by a workstation. The current plan offers the user the ability to interact with only one application at a given time, which is a holdover from earlier implementations and hardware platforms. X provides straightforward multiple window handling capabilities, which the design should allow and, in fact, encourage.

e. Implement the function key display attached to the side or bottom of the application's main window.

This approach would allow a single screen update to occur, minimizing display flicker and transactions from the client to the X server. While somewhat different from the current implementation, the approach works well with the X paradigm and uses machine and screen resources efficiently.

f. Limit the use of FORTRAN to applications.

FORTRAN presents a serious problem to Unix systems. The language definition lacks many of the data types required to deal with the byte stream I/O model and extensive use of pointers required by the Unix I/O paradigm of "devices are special cases of files". While the FORTRAN 88 language committee has introduced a POINTER type into the language definition, we can expect a number of years to elapse before FORTRAN 88 is widely accepted.

Given this problem, interfacing FORTRAN routines with X -- a system that depends on pointers, functions with a variable number of arguments, and structure data types -- is difficult to implement in a portable manner. The implementation requires interface routines written in C to handle interactions with the window system and rework data and values into forms that can be processed by FORTRAN routines. Such an interface can be built, and in fact is the current method of calling Menu Handler services from FORTRAN programs, but it will become more difficult as the X environment progresses into object-oriented languages such as C++.

In the long term, the use of FORTRAN should be limited to the application portion of the M-A-D model with the menu and display portions of the application written in C to allow the best use of the window system facilities while retaining the large base of FORTRAN applications developed over time.

In the shorter term, NASA may wish to investigate the use of named pipes to communicate to C programs dealing with the user interface. FORTRAN has no problems coping with file processing, and the named pipe facility in the SVID provides the ability to treat a pipe to another user process as if it were a file.

6. Evaluation of Extent of Code Revision to Migrate TOAST to X

1. TOAST Executive

The TOAST Executive will not require extensive modification. Minimal changes will need to be made to allow for creation of a phantom display instead of modifying terminal parameters and status. Any dependence on the Unix stdin/stdout/stderr file handles should be removed. They cannot be implemented in the X environment.

2. Menu Handler

Menu Handler will need to be almost completely rewritten, retaining only the code to read and parse menu definition files and do field validation, which represents only 10% of the currently existing code. We estimate that an X expert would require roughly 1 year to rewrite Menu Handler to provide the current text-based functionality.

B. Expandability

Since TOAST is growing to meet the requirements of multiple user groups, it must expand to satisfy these needs. Advantages and constraints of the current version in relation to expandability are shown in Figure 9.

Advantages of Current Version

a. TOAST does not have constructs that inhibit portability to other Unix systems.
b. TOAST is not memory intensive.
c. TOAST uses text files for configuration.
d. Migration to X enforces standards compliance and increases portability.
e. The Menu-Application-Display model can address new capabilities independent of applications.

Constraints of Current Version

a. Design reliance on semaphores.
b. FORTRAN 77 interface.
c. Current Masscomp software license agreement limits the number of users to 8 per machine.
d. System V semantics for shared memory have problems in a multiprocessor environment.
e. Performance of the TOAST Resource Manager event queuing will constrain throughput.

Figure 9: Expandability Advantages and Constraints of the Current Version

Our overall assessment of expandability of the TOAST software is that the advantages of the current version outweigh the constraints. However, unless event queueing is redesigned, TOAST Resource Manager performance will impact expandability adversely (see section A4 above).

X. Industry Perspective

To provide recommendations for future TOAST directions, we researched emerging computing technologies. In this section, we present a brief discussion of the following areas.

A. Graphical User Interfaces (GUI)
B. Standard Graphics Packages
C. Database Systems
D. Distributed Authentication Systems
E. Network Services
F. Distributed Operating Systems

We will address their impact on TOAST in the section XI.

A. Graphical User Interfaces (GUI)

The industry is moving towards simplicity of user interaction and standardization within a potentially heterogeneous hardware and software environment. User interaction paradigms that cannot adapt to these two criteria are being abandoned in favor of more compatible approaches, represented by the selection of the X Window System as a basis for complex graphical user interfaces. Proprietary windowing and graphics systems are losing market share in the general purpose workstation and terminal market.

While the hardware-independent specification of X offers the industry a basic platform of screen, keyboard, and pointer handling operations, these operations are generally considered to be difficult to use directly in application programs due to the low level of the X library calls. In ordinary practice, the X library calls are combined into procedures that implement a more complex object (such as a slider or pushbutton) called a *widget,* which is then employed in applications program. Three major commercial libraries of widgets hold significant market shares: the Motif widgets from the Open Software Foundation, Open Look from Sun/Unix International, and the Project Athena widgets evolved as part of the Athena educational computing project at MIT. At this point, Motif and the Athena widgets seem to hold a significant advantage in terms of number of users. Motif was the first commercial widget package to provide a strong programming interface at a reasonable cost. The Athena widgets are provided with the X distribution tape at no cost.

B. Standard Graphics Packages

Graphics are becoming increasingly important to support design, computation, modeling, and simulation. The Programmer's Hierarchical Interactive Graphics System (PHIGS) represents the industry's current direction as a platform independent graphics library. PHIGS has ANSI standard bindings for C and FORTRAN, and includes significantly enhanced functionality when compared with GKS or GKS-3D. The PEX (PHIGS Extension to X) project plans to release a freely distributable PHIGS toolkit for X later this year, and several commercial implementations of PHIGS for X are available from IBM and DEC as well as other vendors.

C. Database Systems

In general, Unix is not a congenial platform for database systems due to the design of the underlying I/O code (emphasis on character by character I/O vs. block-oriented I/O). While most solutions in this area tend to be vendor-specific, Oracle (Oracle Corp.) , Ingres (Ingres Corp.) , and Unify (Unify Corp.) are used on a number of different platforms and represent reasonably strong database query engines.

Standard Query Language (SQL) is the common query language for almost all DB systems. Many systems combine SQL with a 'query by example' module to translate English queries to the equivalent SQL statements. As an enhancement to traditional querying, interfaces using artificial intelligence techniques such as Q&A from Symantec are coming into use.

D. Distributed Authentication Systems

One of the disadvantages of Unix is its inherently weak security mechanism and the relative difficulty of authenticating users requesting resources from a remote system in terms of identity and access control. The granularity of the Unix permission scheme is large, thus limiting access controls and data protection to relatively large groups based on uid and group membership.

Kerberos (developed as part of Project Athena at MIT) adds a rule-based permission scheme and a secure 'ticketing' method granting access to controlled resources. As an auxiliary function to the ticketing system, Kerberos maintains a centrally administered identity registry, allowing systems to be reasonably certain that users are who they claim to be.

Obviously such an extensive enhancement of the Unix permission structure is intrusive to the system configuration as delivered. Utility programs that deal with user authentication, user files, and network services must be replaced with modified versions that consult the Kerberos server for verification. Fortunately, the replacements have already been evaluated by the DoD and certified as completely functional replacements.

E. Network Services

Networking is an an integral part of computing in industry. Network services are required to provide such system services as distributed directories (Unix "yellow pages"), network remote procedure calls, and remote file access.

At this point in time, two protocols are competing for delivering network services - TCP/IP and OSI. TCP/IP is a mature protocol developed on the Internet over the past 20 years. OSI is a newer protocol that is under development by the International Standards Organization.

Currently, TCP/IP is the protocol of choice. The OSI protocol has problems in the areas of stability, performance, and behavior under stress. Specifically, the OSI specification is not stable at this time. In our opinion, it will not be mature for at least 5 years. Many important parts are still in draft status and are being constantly revised. No current OSI implementation available commercially can sustain the throughput of current TCP/IP implementations. Behavior under stress is not known since there are not enough OSI

implementations available to evaluate its performance under heavy stress in life-threatening situations. TCP/IP has been extensively stress-tested by the DoD.

F. Distributed Operating Systems

Distributed operating systems offer the advantage of scalable computer power, which reduces hardware costs, and direct control over the behavior of the operating system. They are an active area of research and development at the present time.

Few production distributed operating systems exist today. Amoeba, developed by Andrew Tannebaum, and Mach, developed at Carnegie-Mellon University, represent the current state of the art in both stability and flexibility. Mach is used on a wide variety of machines including the BBN Butterfly, the Unix frontend to the Connection Machine, the IBM/RT, and the Convex. At the present time, Amoeba runs on the Sun/3 and DEC VAXstation with other ports under consideration. Both distributing operating systems implement message passing, object orientation, and compact kernels.

Mach provides distributed services for memory, remote procedure calls, and network objects independent of location. It has extensions to Unix that support a network time protocol and arbitrary event protocols. However, Mach is not transparent to application programs, and distributed parts of the operating system are not invisible. Application programs must be written to use them. As a further consideration, the BSD 4.3 compatible version will be expensive as it requires both an AT&T and 4.3 source license.

Amoeba treats multiprocessor group as pools of resources. To the end user, it looks like a large mainframe system on local area networks (LANs) and wide area networks (WANs), with the entire matrix of processors available to all machines. For baseline data, the bullet data server can be used for data that does not change and can perform up to 50 times faster than a standard Unix system. Amoeba assumes very little of a processor -- CPU, memory, network connection. It is dynamically reconfigurable; resources can replicate and processes can migrate. In addition, the present version has an X port, and TCP/IP and OSI prototypes are implemented. As an added bonus, the software is free and requires no source license. However, the current implementation is not completely BSD 4.3 compatible.

XI. Recommendations for Future Directions for TOAST

Having briefly discussed some of the relevant trends in emerging computing technology, we will conclude this report with recommendations of how these technologies could enhance TOAST.

Our overall recommendation is:

> **Plan for a distributed operating environment with services, such as event notification, authentication and configuration management, database, and graphical user interfaces, provided via a high speed network.**

While we are not recommending specific products or presenting a design, we would like to point out where these features are available at the present time. Specifically,

1. Distributed Operating System

Amoeba has many features that could be useful to future TOAST implementations. The bullet data servers could be used with baseline data. The system's flexibility and transparency support redundant resources and configuration management.

Event notification exists in Mach, which has extensions to Unix that support a network time protocol and arbitrary event protocols. These protocols could be adapted for use in implementing clocks and DDDs when TOAST migrates to X.

2. Authentication and Configuration Management

Kerberos supports a multi-machine setup and can be administered remotely. It could be adapted to the TOAST model and would permit TOAST to avoid running as the superuser (root).

3. Database

Integrating a COTS database program into TOAST would provide an extra level of data verification in addition to that provided by Menu Handler and could certainly simplify the TOAST Resource Manager design by off-loading data access control to the database manager system. Changes to the database could be made automatically with automatic rollback in case of failure. In addition, the embedded information about the format of the stored data would allow use of database utilities to extract data from the database.

There are drawbacks to such an approach as well. Databases are complex subsystems which in almost every case require a full time administrator responsible only for database operations, including the overhead of maintaining access controls to data tables. In addition, the complexity of the file structures used by most DBMS' makes regular system backups an absolute necessity, as reconstruction of a damaged database is often impossible within a reasonable amount of time.

4. Graphical User Interface (GUI)

The GOSIP standard is discussing Motif as a possible windowing manager. TOAST should plan to take advantage of multiple windowing capabilities in future releases.

5. Networks

Changes in the area of network media are rapid. Today, TCP/IP is the recognized industry leader. In future years, OSI may well surpass today's standard.

XII. Further Reading

Future Issues

Series of articles on open visions for the 90s, *Unix Today!*, February 5, 1990.

Series of articles in *Unix Review* (vol 8. no. 1) on technology trends of the 90s.

Authentication Systems

MIT Project Athena, "Kerberos: A Distributed Authentification System for the Unix Environment", 1987. Available via anonymous ftp from athena-dist.mit.edu in directory pub/athena.

The C Language

Harbison, Samuel P. and Guy L. Steele Jr., A C Reference Manuals, Prentice-Hall, New Jersey, 1984.

Kernigan, Brian W. and Dennis M. Ritchie, The C Programming Language, Second Edition, Prentice-Hall, New Jersey, 1988.

Cooperative Processing

Altman, Ross, "Are You Ready for Cooperative Processing?", *Information Center*, April 1990, pp. 20-31. Indepth tutorial on cooperative processing and different styles for achieving it.

Gantz, John, "Are you ready for cooperative processing?", *Networking Management*, April 90, pp. 54-55. Short introduction to cooperative processing with a historical perspective.

Database

Date, C.J., An Introduction to Database Systems, Addison-Wesley Publishing Company, 1983.

Edelstein, Herb, "Distributed Databases", *DBMS*, September 1990, pp. 36-48. Addresses issues relating to distributed database technology.

Distributed Operating Systems

Boyes, D. and R. Collins, The Mach Papers, CMU Press, 1989.

Cornell Theory Center, "The ISIS Distributed Computing Environment", 1989.

The Mach Operating System, CMU technical reports 989-1143, Carnegie-Mellon University.

Lo, Virginia, Mark Vandewettering, George Rankin, Jeff Eaton, et. al., "EXODOS: A Distributed Operating System", Tech Report 79, University of Oregon, Department of Computer Science, 1988.

Mullender, S.J., G. van Rossum, A.S. Tanenbaum, R. van Renesse, H. van Staveren, "Amoeba--A Distributed Operating System for the 1990s", IEEE Computer Magazine, May 1990.

Tanenbaum, A.S., and S.J. Mullender, "An Introduction to Amoeba", from the Amoeba papers available from Addison-Wesley first quarter 91.

Wayner, Peter, "Distributed Applications with Vision", *Unix Review*, June 1990 (vol 8. no. 6), pp. 58-62.

Documentation

Grimm, Susan J., How to Write Computer Documentation for Users, Van Nostrand Reinhold Company, New York, 1987.

Katzin, Emanuel, How to Write a Really Good User's Manual, Van Nostrand Reinhold Company, New York, 1985.

Graphics

DeGroot, Marc, "Virtual Reality", *Unix Review*, August 1990 (vol 8. no. 8), pp. 32-36. Discusses an emerging computer technology that allows users to interact directly with objects in a 3D scene.

High Performance Computing/Parallel Processing

Hwang, Kai, and Douglas DeGroot, editors, Parallel Processing for Supercomputers and Artificial Intelligence, McGraw-Hill Publishing Company, 1989.

Rattner, Justin, "Micros at the Threshold", *Unix Review*, April 1990 (vol. 8. no. 4), pp. 36-40. Describes parallel processing technology built on microprocessors. Rattner is founder of Intel Scientific Computers, and is principal investigator for the Touchstone project, funded jointly with DARPA and Intel to develop a parallel supercomputer capable of performing 150 GFLOPs.

Reed, Daniel A. and Richard M. Fujimoto, Multicomputer Networks: Message Based Parallel Processing, MIT Press, 1987.

Wallach, Steve, "Supers Built to Fit", *Unix Review*, April 1990 (vol. 8. no. 4), pp. 45-50. Describes supercomputer architectures for the future and issues concerning their programming environments. Steve Wallach is one of the founders of Convex Computer Corporation.

Networking

Black, Uyless, Computer Networks: Protocols, Standards, and Interfaces, Prentice-Hall, Inc. 1987.

Case, J. and J. Davin, M. Fedor, M. Schoffstall, "Keeping It Simple", *Unix Review*, March 1990 (vol. 8. no. 3), pp. 60-66. This article is an introduction to SNMP (Simple Network Management Protocol).

Comer, Douglas E., Internetworking with TCP/IP: Principles, Protocols, and Architecture, Prentice-Hall, Inc., 1988.

Davidson, John, Introduction to TCP/IP, Springer-Verlag, 1988.

Harrison, Bradford T., "TCP/IP or OSI: Which Protocol Should You Speak?", *Dec Professional*, April 1990, pp. 38-46.

Gantz, John, "Advanced technology: Where is it leading us?", *Networking Management*, May 1990, pp. 34-48. Summary of new technologies with implications for networking.

McClain, Gary, "What is OSI and What Can You Expect It to Do for You?", *Information Center*, Weingarten Publications, June 12, 1990 (vol. VI, No. 6), pp. 12-17.

Stallings, William, Handbook of Computer-Communications Standards, Volume 1: The Open Systems Interconnection (OSI) Model and OSI-Related Standards, Howard W. Sams & Company, 1987.

Stallings, William, Handbook of Computer-Communications Standards, Volume 3: The TCP/IP Protocol Suite, Howard W. Sams & Company, 1989.

Portability

Levinger, Janet, "Portability's True Colors", *Unix Review*, vol. 8 no. 6, pp. 38-44.

Programming Tools

Kolstad, Rob, "Perl: The Super-Language", *Unix Review*, May 1990 (vol. 8. no. 5), pp. 30-40. Overview of the Practical Extraction and Report Language (PERL). Three-part article with subsequent parts appearing in June and July issues.

Wall, Larry, perl: An Improved Shell Programming Language for Unix Systems, available from via anonymous ftp to uunet.net under comp.sources.unix.

Security

Abrams, Marshall D. and Harold J. Podell, Computer and Network Security, Computer Society Press of the IEEE, 1987.

Arnold, Ken, "Setuid Security Blankets", *Unix Review*, May 1990 (vol. 8. no. 5), pp. 22-26. Describes security holes in Unix caused by setuid programs.

Russell, D.F., "How to Safeguard your Unix System", *Unix Today!*, April 2, 1990, pp. 38-41. Describes steps that will help guard against security breaches. Significance concerns the holes that are documented and easily accessible in this well-read journal.

Curry, David A., "Improving the Security of your Unix System", SRI International, April 1990.

Software Engineering

Benson, Scott E., "Give Me Your Tired, Your Poor...", *System Builder*, April/May 1990, pp. 27-29. Description of redevelopment engineering and project guidelines.

Hume, Andrew, "Backups: Can You Ever Be Too Safe?", *Unix Review*, May 1990 (vol. 8. no. 5), pp. 54-58. Describes current Unix backup schemes and future trends.

Martin, James, "Timebox Methodology", *System Builder*, April/May 1990, pp. 22-25. Description of timebox procedure for controlling software development projects.

Nurmia, Juhani, "Testing Your Testing Program", *System Builder*, April/May 1990, pp. 47-50. Short article on software engineering principles regarding testing programs and documentation.

Richartz, John, "Software Configuration Management Tools", *Unix Review*, May 1990 (vol. 8. no. 5), pp. 87-95. Compares CCC and Aide-de-Camp, 2 COTS packages for software configuration management.

Sommerville, Ian, Software Engineering, Addison-Wesley, 1989.

Standards

Dawson, Frank , and Fran Nielsen, "ODA and Document Interchange", *Unix Review*, March 1990 (vol. 8. no. 3), pp. 50-56. Tutorial on the office document architecture (ODA) international standard for passing compound documents between dissimilar document-processing systems.

Faden, Michael , "Open System Gets Euro Support", *Unix Today!*, June 25, 1990, pp. 46-50. Describes standards receiving commercial user backing in Europe.

Jackson, Kelly, "Countdown to GOSIP", *Communications Week*, July 23, 1990, pp. 1,19, 20. Description of the Government Open System Interconnect Profile.

Wagner, Mitch, "Standards Work Marches On", *Unix Today!*, March 5, 1990, pp. 50-51. Describes the work of several NIST and IEEE standards committees in the areas of conformance testing, profiling, X/Open branding, X-Window toolkits, and security. Defines many of the acronyms that you've often wondered about.

User Interface

Baecker, Ronald M., and William A.S. Buxton, Readings in Human-Computer Interaction: A Multidisciplinary Approach, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1987.

Bentley, Michael Brian, The Viewport Technician, Scott, Foresmann and Company, Glenview, Illinois, 1988.

Brand, Stewart, The Media Lab, Viking Penguin Inc., New York, 1987.

Heller, Dan, XView Programming Manual for Version 11 of the X Window System - An OPEN LOOK Toolkit for X11: Volume 7, O'Reilly & Associates, Inc., 1989.

IBM Corporation, SAA: Overview, GC26-4341, 1990.

Krill, Paul, "OSF/Motif Getting Nod from Most ISVs", *Unix Today!*, May 14, 1990, pp.26, 28. Provides a summary of graphics user interfaces for Unix.

Moran, Robert, "The SAA Shadow Play", *InformationWeek*, August 13, 1990, pp. 22-24.

Nelson, Ted, Computer Lib, Tempus Books of Microsoft Press, Redmond, Washington, 1987.

Quercia, Valerie and Tim O'Reilly, X Window System User's Guide, Volume Three, O'Reilly & Associates, Inc., California, 1989.

Shneiderman, Ben, Designing the User Interface: Strategies for Effective Human-Computer Interaction, Addison-Wesley Publishing Company, Reading, Massachusetts, 1987.

Shu, Nan C., Visual Programming, Van Nostrand Reinhold Company, New York, 1988.

Sun Microsystems, Inc., OPEN LOOK: Graphical User Interface Application Style Guidelines, Addison-Wesley Publishing Company, Inc., Massachusetts, 1990.

Sun Microsystems, Inc., OPEN LOOK Graphical User Interface Functional Specification, Addison-Wesley Publishing Company, Inc., Massachusetts, 1989.

Miscellaneous Articles

Erickson,Tom, "Choosing the Right Metaphor", *Apple Viewpoint*, Apple Computer, April 1, 1990, pp. 1-4. Discusses metaphors for designing user interfaces.

Schuman, Evan, "GM Issues Wish List", *Unix Today!*, March 19, 1990, pp. 1, 52. Describes the software and hardware requirements for EDS's workstation criteria. Of significance is a requirement for OSF/Motif, X-Window, PHIGS moving later to PEX, NFS, and POSIX.

XIII. Acknowledgements

The TOAST evaluation team would like to acknowledge the help of several people who provided support for the project.

Mike Evans, NASA contract supervisor and TOAST engineer

Diane Campbell and Ken Wallis, TOAST Developers, who answered numerous questions

Erick Rivas, a senior Computer Science student at UH Clear Lake, who provided programming support

Mark Grubbs, a senior Computer Science student at Rice University, who provided programming support

Julie Grubbs, who performed interview transcriptions

Melodee Schaller, who provided technical writing support

Dwayne Fontenot, a senior Computer Science student at Rice University, who developed the graphics for the evaluation presentations

And all the NASA and contract personnel who participated in the interviews.