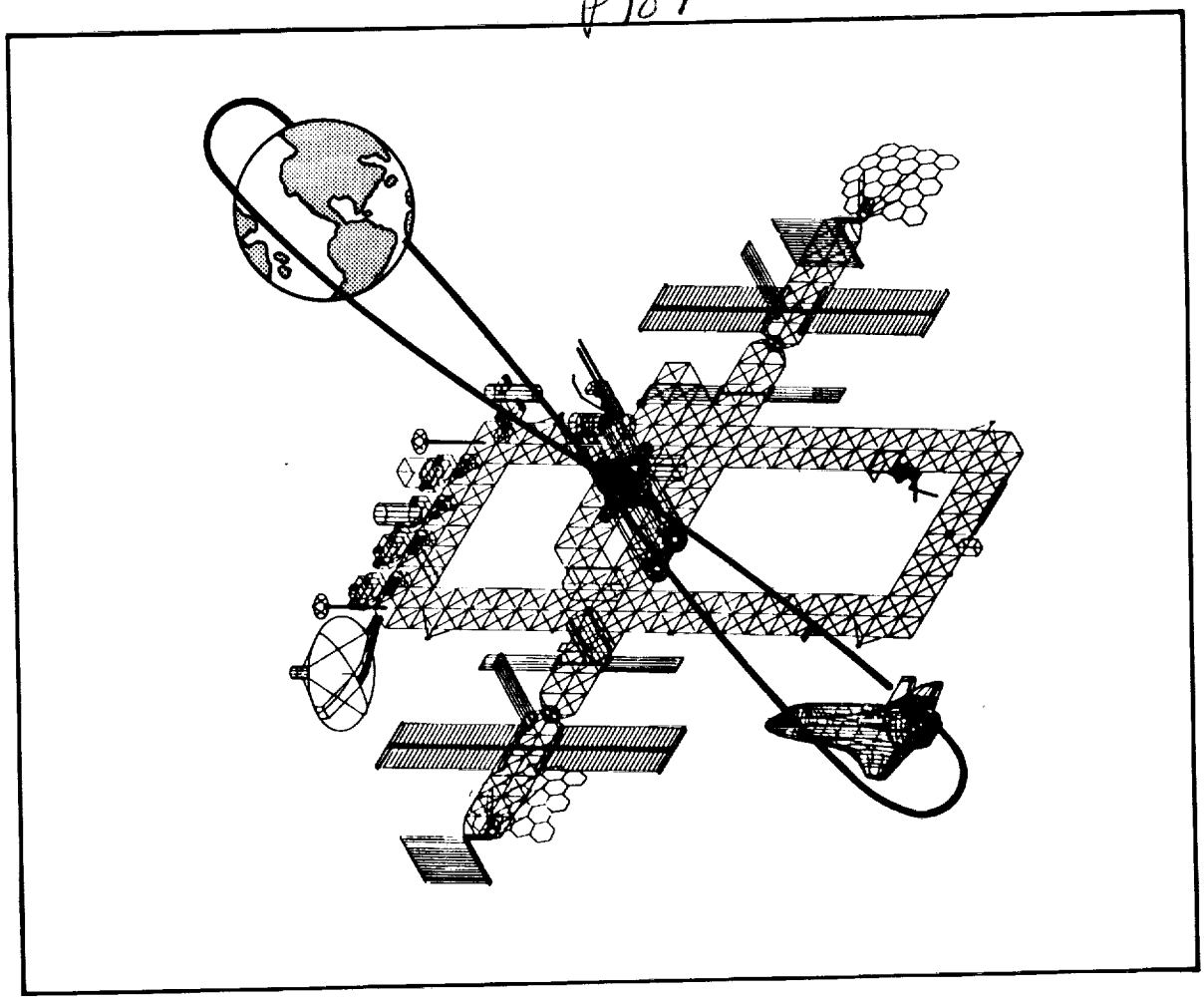
P187



RICIS '88 SYMPOSIUM

Omit to p13

RICIS SYMPOSIUM'88

Co-Sponsored by:

NASA/Johnson Space Center

and

University of Houston-Clear Lake

November 9-10, 1988

Houston, Texas

RICIS SYMPOSIUM '88

Steering Committee

Technical Co-Chairs:

A. Glen Houston, *Director - RICIS, University of Houston-Clear Lake*

Robert B. MacDonald, *Assistant for Research and Education - Mission Support Directorate, NASA/JSC*

Conference Coordinator:

Katherine Moser, *Coordinator - SEPEC, University of Houston-Clear Lake*

Members:

Glenn B. Freedman, *Director - SEPEC, University of Houston-Clear Lake*

Bryan I. Fugate, *Technical Manager - Software Technical Support, SoftwareTechnology Program, Microelectronics and Computer Corporation*

John R. Garman, *Associate Director - Mission Support Directorate, NASA/JSC*

Richard Kessinger, *Manager - Space Programs, SofTech, Inc.*

Everett Lyons, *Project Manager - Space Station Sofware Support Environment, Lockheed Engineering*

Charles W. McKay, *Director - SERC; High Technologies Laboratory, University of Houston Clear Lake*

James Raney, *SSE Project Manager, Mission Support Directorate, NASA/JSC*

University of Houston-Clear Lake
RICIS Steering Committee

Michael C. Gemignani, *Provost and Senior Vice President for Academic Affairs*
James T. Hale, *Vice President for Administration and Finance*
E. T. Dickerson, *Dean, School of Natural and Applied Sciences*
L. Todd Johnson, *Dean, School of Business and Public Administration*
Joan J. Michael, *Dean, School of Education*
Wayne C. Miller, *Dean, School of Humanities and Human Sciences*
David A. Hart, *Executive Director, University Computing*

Introduction

Welcome to RICIS SYMPOSIUM '88!

Considerable national interest is concentrated on enhancing productivity to help ensure a U. S. competitive advantage in the world marketplace. The technical community has realized the importance of software in meeting this goal for some time (15-20 years!). Over the past three to five years, the business community has come to understand and accept the importance of software. In fact, software has surpassed hardware as the key element to the success of many products, systems and businesses.

Despite this growing awareness of the importance of software, much work still needs to be done in addressing software development issues. Because an increasing number of people have recognized the need for a more disciplined approach to software development, "software engineering" is emerging as an important professional discipline. Unfortunately many remain unaware of modern software engineering methods and procedures, and too many organizations are still developing software in a haphazard fashion.

Given this perspective, plus the fact that the focal point of RICIS research is the NASA Space Station Program, "Integrated Environments for Large, Complex Systems" is an appropriate theme for RICIS SYMPOSIUM '88. Distinguished professionals from industry, government and academia have been invited to participate and present their views and experiences regarding research, education and future directions related to this topic.

Within RICIS, more than half of the research being conducted is in the area of Computer Systems and Software Engineering. The focus of this research is on the software development life-cycle for large, complex, distributed systems. Within the education and training component of RICIS, the primary emphasis has been to provide education and training for software professionals.

However, RICIS research has grown to the point that it is not feasible to cover the many on-going research activities in a single day-and-a-half conference. Hence we have elected to have a series of conferences, with each focusing on a specific technical area or topic of interest within RICIS. An overview of the accomplishments to date, research plans for the coming year, and upcoming conferences will be presented by the RICIS research area directors for each of the five RICIS research areas.

We hope you find RICIS SYPOSIUM '88 both informative and enjoyable!

A. Glen Houston
Technical Co-Chair

Robert B. MacDonald
Technical Co-Chair

Table of Contents

Integrated Computing Environments for Large, Complex Systems

Wednesday November 9,1988

12:30 - 1:00 **Welcomes and Introductions** Crystal Ballroom

A. Glen Houston
Conference Technical Co-Chair, Director RICIS
UH-Clear Lake

Robert B. MacDonald
Conference Technical Co-Chair
Assistant for Research and Education Mission Support Directorate
NASA/Johnson Space Center

Michael C. Gemignani
Senior Vice President and Provost
UH-Clear Lake

Joseph P. Loftus, Jr.
Assistant Director (PLANS)
Office of Director
NASA/Johnson Space Center

1:00 - 1:45 Crystal Ballroom

Keynote Address
Software Development Environments: Status and Trends

Larry E. Druffel

Software Engineering Environments offer significant opportunity for improved productivity. A collection of tools is not sufficient. To realize that opportunity, the environment must support a disciplined software engineering process and consistent methodology. The presentation will describe a current state of practice, expose some current trends and offer some considerations for future development.

1:45 - 2:00 Break Ballroom Deck

2:00 - 3:30

RICIS Research Review

Crystal Ballroom

A. Glen Houston, Director, Research Institute for Computing and Information Systems (RICIS), University of Houston-Clear Lake

Dr. Houston will introduce the directors of each project area within RICIS. Each project director will summarize his past year's research accomplishments.

Computer Systems and Software Engineering—Charles McKay, Director, High Technologies Laboratory and Software Engineering Research Center (SERC), UH-Clear Lake

Information Systems—Peter Bishop, Director, Space Business Research Center (SBRC), UH-Clear Lake

Mathematical and Statistical Analysis—A. Glen Houston

Artificial Intelligence and Expert Systems— Terry Feagin, Professor of Computer Science, UH-Clear Lake

Education and Training—Glenn B. Freedman, Director, Software Engineering Professional Education Center (SEPEC), UH-Clear Lake.

3:30 - 3:45 Break Ballroom Deck

3:45 - 5:15

Session I

Requirements Analysis Fundamentals

Crystal Ballroom

Session Chair: Bryan I. Fugate, Technical Manager-Software Technical Support, Software Technical Program, Microelectronics and Computer Technology Corporation

Moderator: Michael J. See, Head, Advanced Project Section Facility and Support Systems Division, Mission Operations Directorate

Requirements Analysis, Domain Knowledge and Design

Colin Potts

Requirements for software-intensive systems constantly evolve. A design architecture that is based on the structure of the functionality at one point in time is vulnerable to requirements changes. An alternative, more stable approach, domain modeling, should facilitate iterative design, encourage the reuse of design abstractions, and may enable us to treat traceability more formally. Such techniques are new with the expected benefits potential.

Mr. Potts is a member of the technical staff in MCE's software technical program, previously he lectured at Imperial College, London, and was Principal Investigator for two academic/industrial projects addressing requirements analysis and formal specifications techniques. His current research interest includes specification and design methods and hypertext applications in software engineering.

Knowledge-Based Requirements Analysis for Automating Software Development

Lawrence Markosian

A knowledge-based approach to representing and reasoning about requirements support the automation of software development. It shows how to formalize system requirements and apply domain-specific analysis techniques to validate the formalized requirements and simultaneously derive detailed functional specifications. General programming knowledge can then be applied to the functional specifications to yield executable code that meets the validated requirements.

As a research associate at Stanford University, Mr. Markosian specialized in computed assisted instruction of logic-oriented courses. At Systems Control Technology, he applied AI to DOD applications. As a founder of Reasoning Systems, Inc., Lawrence Markosian has applied software specifications synthesis technologies to communication, program translation, C^{3}I, equipment configuration and product planning and integration.

5:15 - 5:45 Break Ballroom Deck

5:45 - 6:30 Reception-Cash Bar Ballroom Deck

6:30 - 9:00 **Dinner and Speaker** Crystal Ballroom C

Integrated Software Support Environments: Some Lessons Learned

Frank Belz

TRW has developed and applied several software support environments on several dozen software projects, with varying degrees of success. This talk will summarize the major lessons learned which distinguish the more successful environment applications from the less successful ones. These lessons learned will be related to a new research and development program in environment technology being conducted by the Arcadia Consortium.

Thursday November 10, 1988

8:00 - 8:30 **Continental Breakfast** Ballroom Deck

8:30 - 10:30 **Session II** Crystal Ballroom

Some Lessons Space Station Software Support Environment

Session Co-Chair: Everett Lyons, Project Manager-Software Support Environment, Lockheed Engineering

Session Co-Chair: Jim Raney, SSE Project Manager, Mission Support Directorate NASA/Johnson Space Center

Learned From an Ada Conversion Project

Tim Porter

Mr. Porter will present lessons learned from the development of Ada software programs to support large command and control systems. The presentation will cover lessons learned about reusability, maintainability, portability, productivity and virtual interfaces from large projects. Special emphasis will be on the lessons learned from porting the SAVVAS tool to the IBM environment to support the development of NASA's software support environment.

Mr. Tim Porter, Deputy Division Manager for Science Applicators International Corporation (SAIC), was chartered with the funding of the company's future software engineering environment. He has 15 years experience designing and developing command and control systems, software productivity tools, database design and management systems. He has specialized in the application of Ada and relational database technology to support large command and control systems. Mr. Porter has also focused on the development of reusable software to improve programmer productivity.

Automating Software Quality Assurance

Gokul Bhaumik

Software quality assurance within an automated software development process control environment. New quality evaluation concepts utilize automated process management features of the system architecture for concurrent verification of design, the development activities and their attendant products for quality.

Mr. Gokul Bhaumik is Manager for Lockheed, Safety Reliability & Quality Assurance (SR&QA), Software Support Environment System Project. Mr. Bhaumik has over 20 years experience in software test and evaluation, software development, and software quality assurance assignments. In recent years, he has focused on the application of modern software engineering technologies and practices to the quality assurance process.

SSE Tool Interoperability

C. T. Shotton

How to make heterogeneous tools work together.

C. T. Shotton is Technical Director of Planning Research Corporation (PRC). Mr. Shotton has been concentrating on using grammar based technology to solve software interoperability problems for over four years.

10:30 - 10:45 Break Ballroom Deck

10:45 - 12:15

Session III

Crystal Ballroom

Developing Software Engineering for Competitive Advantage-Industry and Federal Government

Session Co-Chair: **John R. Garman,** Associate Director-Mission Support Directorate, NASA/Johnson Space Center

Session Co-Chair: **Richard Kessinger,** Manager-Space Programs, Sof-Tech, Inc.

The Role of Software Engineering in the Space Station Program

Dana L. Hall

The Space Station Program is characterized by extensive application of software throughout its distributed flight and ground environment. Software represents both the key means by which complex functions and user services will be accomplished as well as a likely source of development, integration, and long term evolution problems. The leverage that software has on the Space Station Program is a Harbinger for all future programs and systems within the Agency. Software Engineering and the tools and processes that surround it are crucial elements of NASA's future.

Dr. Dana L. Hall is Deputy Director of Information Systems Services Group, Space Station Program Office at NASA Headquarters.

Experience in Applying Quality and Productivity Engineering into an Existing System

Jack Munson

Mr. Munson will present the problems Unisys experienced defining, developing, and implementing a system-wide quality engineering program, within the difficult environment of existing systems. This included a variety of different management systems and people from disparate cultural backgrounds. Also addressed are the results to date and the quality goals for the near future.

Jack Munson is Vice President and General Manager for Unisys Houston Operations. Mr. Munson was in charge of the Unisys activity which, in conjunction with Rockwell as prime, won the Space Transportation System Operations Contract (STSOC) in the Fall 1985. The major consolidation contract started in January 1986, with Unisys responsible for all existing ground based shuttle software at JSC—previously maintained by eleven different contractors.

Next Generation

Howard Yudkin

The synthesis process for incororating reuse and prototyping ideas into large software system developmentsuggest how the acquisition process might be changed to support the new development process.

12:15 - 1:30

Lunch

Crystal Ballroom C

1:30 - 3:00

Panel I

Crystal Ballroom

Software Engineering as an Engineering Discipline

The Panel will explore the emerging discipline of software engineering from a variety of perspectives: theoretical foundations, educational foundations, and future directions of the field. Panelists will address the nature of software engineering as an engineering discipline distinct from computer science and electrical engineering. Further, they will assess software engineering in relation to the development of education and training programs that support industry and government demands.

Panel Chair and Moderator: Glenn B. Freedman

Director, Software Engineering Professional Education Center (SEPEC)
University of Houston-Clear Lake

Dr. Freedman is responsible for university education and training programs in software engineering offered to NASA/JSC and the surrounding aerospace community. He is also an associate professor in the School of Education.

Panelist: John Brackett

Professor, College of Engineering, Boston University

Dr. Brackett has been a leading software industry executive and was a faculty member at the Wang Institute of Graduate Studies.

Panelist: Ed V. Berard

President, EVB Software Engineering, Inc.

Mr. Berard is recognized as one of the nation's leaders in software engineering and Ada education and training. In addition, he has pioneered the development of large libraries of reusable Ada components.

Panelist: Robert B. MacDonald

Assistant for Research and Education, Mission Support Directorate, NASA/JSC

Mr. MacDonald has been a strong advocator for the development of software engineering as a rigorous engineering discipline. He has recently been instrumental in providing leadership at NASA for implementation of a comprehensive employee development program in software engineering.

Panelist: Norman Gibbs

Director of Education, Software Engineering Institute, Carnegie Mellon University

Dr. Gibbs has numerous professional affiliations and leaderships in software engineering and computer science education. He received his Ph. D. in Computer Science from Purdue University.

3:00 - 3:15

Break

Ballroom Deck

3:15 - 4:45

Panel II
Computer-Aided Software Engineering Environments for Real-Time Systems

Crystal Ballroom

Large, complex, distributed systems with operational requirements to support non-stop and mission and safety critical (MASC) components pose life cycle challenges that can not be safely or cost effectively supported with the traditional models, methodologies, and tools that sometimes suffice for smaller and simpler applications. Furthermore, these challenges require an integrated approach across three environments (host, integration, and target) to acceptably reduce and control risks. This session will concentrate upon some of the most crucial issues in each of the three environments.

Panel Chair and Moderator: Charles W. McKay

Director, Software Engineering Research Center, High Technologies Laboratory, University of Houston-Clear Lake

Dr. McKay, Professor of Computer Systems Design at UH-Clear Lake, will address the development of comprehensive software engineering environments, with emphasis on large, real-time Ada systems.

Panelist: Miguel A. Carrio, Jr.

Manager, Advanced Technology Programs, Teledyne Brown Engineering

Mr. Carrio will address modeling, method and tools appropriate for the first two phases of the life cycle: systems requirements analysis and the partitioning and allocation of these requirements among software, hardware, and operational interfaces.

Panelist: E. Douglas Jensen

Director, Research and Development, CONCURRENT

Mr. Jensen will address critical support issues in the kernel and library of the run time support environment of the target processors.

4:45 - 5:15

Closing Remarks and Wrap Up

Crystal Ballroom

Software Development Environments: Status and Trends

Larry E. Druffel

(NOTES)

RICIS Research Review

Charles W. McKay
Peter Bishiop
A.Glen Houston
Terry Feagin
Glenn B. Freedman

Computer Systems and Software Engineeing

Dr. Charles W. McKay

Computer Systems and Software Engineering

Charles W. McKay
SERC @ UHCL

The High Technologies Laboratory (HTL) was established in the fall of 1982 at the University of Houston Clear Lake. Research conducted at the High Tech Lab is focused upon computer systems and software engineering. There is a strong emphasis on the interrelationship of these areas of technology and the United States' space program. In January of 1987, NASA Headquarters announced the formation of its first research center dedicated to software engineering. Operated under the High Tech Lab, the Software Engineering Research Center (SERC) was formed at the University of Houston Clear Lake. The High Tech Lab/Software Engineering Research Center promotes cooperative research among government, industry, and academia to advance the edge-of-knowledge and the state-of-the-practice in key topics of computer systems and software engineering which are critical for NASA. The center also recommends appropriate actions, guidelines, standards, and policies to NASA in matters pertinent to the center's research. Results of the research conducted at the High Tech Lab/Software Engineering Research Center have given direction to many decisions made by NASA concerning the Space Station Program.

Current research involves the investigation of computer systems and software engineering concepts, principles, models, methodologies, tools, and environments. The relationship of this research to large, complex, non-stop, distributed systems is emphasized. Work also continues in the areas of reusability, data management systems, security, distributed systems, and the Ada programming language and programming environments. Some members of the High Tech/Software Engineering Research Center Team are principal members of the ARTEWG (Ada Run Time Environment Working Group), which was founded as an international task force to address the issues of standardizing the interface to the Ada Run Time Support Environment. Team members currently chair the Distributed Ada Task Force and the Subgroup responsible for evolving the Catalog of Interface Features and Options.

This year the High Tech Lab/Software Engineering Research Center worked on a major project on reusability with Martin Marietta Energy Systems, with support from STARS, AIRMICS, DOE and six other universities. This project involved developing a conceptual model for reusable Ada software that spanned the requirements across host, integration, and target environments. A reusability guidebook is to be published later this year with contributions from all participating organizations. It is entitled Guidelines Document for Ada Reuse and Metrics. The High Tech Lab/SERC has participated with MountainNET, Inc. on a related project on reusability which is jointly sponsored by NASA, AJPO, and DOC. This has led to the establishment of an Ada Technology Transfer Network, known as AdaNET. AdaNET is intended to be used

as a repository of reusable products and processes across the life cycle of Ada based projects. The repository will be accessible to public and private organizations for potential use in the non classified community. Ford Aerospace is also working with the High Tech Lab/SERC on a project in reusability. Ford is developing tools and procedures for support of a reusable software library.

SofTech, Inc. has worked with the High Tech Lab/Software Engineering Research Center on many projects. Several of these have been related to NASA's Space Station Program and the use of Ada. Emphasis on software engineering, systems integration and verification, and Information System technology has been prevalent in the center's research. Studies have been conducted to understand the important evolving Ada standards, guidelines and policies. When necessary and appropriate, the center has sought to influence these standards to reflect the best interest of the Space Station Program. Research in the area of multilevel security has been conducted to discover ways to enhance the safety of life and property in the Space Station Program. The need for automatic verification tools for the Space Station Program has also been addressed. Another area of research which has been investigated has been Ada support software, particularly in the areas of its effective use in embedded computer systems and testing and verification of flight software. The implications of the use of Ada for expert and knowledge based systems have also been studied.

Guidelines for extending the CAIS (Common Ada Interface Set) as a baseline for the System Interface Set of the Space Station Program Software Support Environment were investigated through the High Tech Lab/SERC, with support from SofTech and Rockwell. Honeywell, GHG Corporation, and the High Tech Lab/SERC have participated in research in the areas of run time environments. Together this team has worked to implement a baseline model with guidelines and tools to support the distribution of entities within Ada programs with tailorable run time environments. A part of the work has produced demonstrations of distributed Ada and of run time instruments for performance monitoring and command based interactions with the integration environment. The work continues to advance toward a bare machine prototype.

Research in the areas of object based information management systems has been conducted in conjunction with IBM. This project has focused upon identifying the key problems and promising approaches associated with the development and support of such systems.

The High Tech Lab/SERC and Inference Corporation studied the issues and approaches for developing tool support for integrating Ada and artificial intelligence. The project is intended to result in an Ada-based, expert system generation toolset.

For the next five years, the principle thrust of the center's

research will focus upon a new generation of integrated systems software. The PCEE (Portable Common Execution Environment) project is intended to provide a common execution environment for Ada applications software and users. The principal domain of interest is large, complex, distributed computing systems with Mission and Safety Critical (MASC) components which require non-stop operation.

The integrated systems software is to be built in Ada, and supported by a heterogeneous collection of bare machines. The goal is to provide systems software which is tailorable to the needs of a variety of applications, while insuring that performance, fault tolerance, security, extensibility and the requirements for non stop operation are satisfied. The intent of the object based approach is to create an appropriate run time kernel with catalogs of interface features and options. These features and options allow tailoring of system software interfaces to the specific requirements of each application.

By designing the underlying implementation as a set of integrated modules, unnecessary redundancy and conflict among the various subsystems can be minimized while support for performance, robustness, and security can be enhanced. Furthermore, the complementary features and options of the underlying subsystems can be selected for their ability to support MASC components in non-stop, distributed, and embedded applications rather than a more benign, general purpose programming environment. Examples of the types of applications which would benefit from a PCEE include the FAA's next generation of air traffic control software, the Space Station Program, the next generation of C^3I systems, and the next generation of process control and flexible manufacturing systems. The High Tech Lab/SERC is currently working on a project with GHG Corporation to investigate the use of Ada in distributed and fault tolerant real-time applications. The PCEE is being proposed as a standard interface for this project also.

The High Tech Lab/Software Engineering Research Center strives to advance the edge-of-knowledge and the state-of-the-practice in computer and information technologies. Working together with dedicated researchers from government, industry, and academia, the center continues to make important contributions to some of the most critical research areas of today.

RESEARCH REVIEW FOR INFORMATION MANAGEMENT

Peter C. Bishop, Ph.D.
Space Business Research Center
University of Houston-Clear Lake

Research Review for Information Management

The goal for RICIS research in information management is to apply currently available technology to existing problems in information management. Research projects include the Space Business Research Center (SBRC), Management Information and Decision Support Environment (MIDSE), and investigation of visual interface technology. Several additional projects issued reports. New projects include: 1) the AdaNET project to develop a technology transfer network for software engineering and the Ada programming language; and 2) work on designing a communication system for the Space Station Project Office at JSC. The central aim of all projects is to use information technology to help people work more productively.

RICIS instituted the research review process during the institute's inaugural symposium in 1987. The review is an opportunity for RICIS area coordinators like myself to summarize and report our results in specific research areas. As you know, RICIS coordinates research contracts for JSC and - more important - synthesizes and analyzes the results for strategic advantage.

I am responsible for RICIS research in information management. Researchers in this field study ways that information technology can be used to increase productivity. Our goal is to apply currently available technology to existing problems in information management. JSC, like any other large organization, is a wonderful proving ground for such applications.

Space Business Research Center

Today, I would like to share some of the results from the prototype operation of the Space Business Research Center. The Center emerged from the Space Market Model Project, initiated in August 1986. The Project was designed to determine the types of information businesses need to make decisions about commercial prospects of space.

Last year, I reported the results of Phase I. Quite simply, the Center located a lot of information about space, especially from the media and the scientific and technical communities. We did not find as much information about business activity in space,

did not find as much information about business activity in space, but we did find people who wanted it.

Figure 1 shows an analysis of the groups we interviewed to gauge their need for additional space business information, and their ability to pay for such data. We concluded that the two groups most likely to use additional information were the service businesses and government agencies that facilitate the commercialization of space.

Figure 1

Comparison of
Information Needs by Sector

	NEED	INTERNAL CAPABILITY	RESOURCES
Business Service	High	Low	High
Aerospace	High	High	High
Entrepreneur	High	Low	Low
Government			
-general	Low	High	High
-space commercialization	High	Low	High

During Phase II, which started in September 1987, the Center distributed space business information to any business person or U.S. government official who requested it. The Center had approximately 30 clients monthly and handled more than 400 requests for information during the ensuing year. The shear volume of information requests confirmed one of the results from Phase I--that the business community wanted additional data about space.

Figure 2 shows the Center's client-profile. As expected, the business service sector accounted for 50% of the requests. The next figure indicates the types of information clients requested. Directory information, 38% of all requests, was the type more frequently sought. The business community wants to know who is active in space business, and who the potential clients and suppliers are. Economic statistics on the space industry in general and on its respective markets were also requested routinely.

The Center is still taking requests for information, although

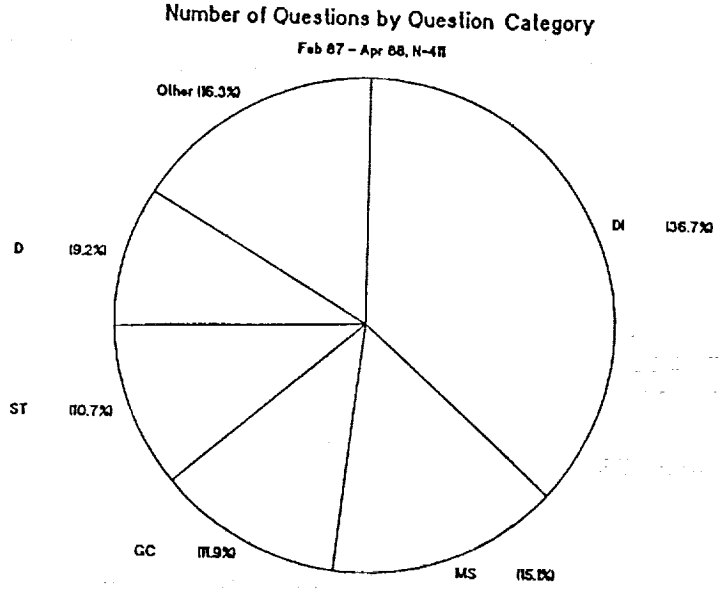
their frequency has diminished. On July 1, the Center began its Phase III operation and started to charge, although at a subsidized rate, for research. The rate of requests, predictably, is lower than it was when the information was free. However we still receive requests steadily, confirming that businesses not only need information about space, but are willing to pay for it.

Figure 2

Number of Contacts by Client Category
February 1987 - April 1988

	Number	Percent
Academic	35	13%
Business Service	138	50%
Government	47	17%
Information Companies	14	5%
Large Aerospace	10	4%
Media	6	2%
Miscellaneous	7	3%
Small Aerospace	17	6%
TOTAL	274	100%

Figure 3



Number of Questions by Question Category
February 1987 - April 1988

Category	Number
Directories	151
Market Studies	62
Government Contracting	49
Space Technology	44
Documents	38
Law and Policy	22
Economic	15
Education	7
Miscellaneous	23
Total	411

The Center has established two other methods to disseminate information during Phase III. The first is a publication program, launched in August with release of the booklet Space Business 1988, an economic profile and executive summary of the space industry today. The public's response to the publication has been very gratifying - the Center sold more than 100 copies in the first month following its release.

The second method of information dissemination is the use of seminars for the purpose of education. The Financial Aspects of the Space Industry was the 1st seminar which the Center co-sponsored with the Houston Chapter of the Texas Society of Certified Public Accountants. The seminar was very successful. It attracted forty Houston business professionals who learned some of the more technical aspects of financing and controlling space ventures from people who are space business veterans.

The Center is poised to become an autonomous and self-supporting research center for space commercialization. In addition to offering research, publications, and seminars, the Center also plans to provide an on-line retrieval service for information on launch histories, space transportation vehicles, and satellites. Proposals to start this prototype service and to continue our other services are currently under review by NASA. To date, the Space Business Research Center has successfully used current technology to productively disseminate information to businesses.

Management Information and Decision Support Environment

RICIS has supported two other research projects to help JSC manage information. The first is the Management Information and Decision Support Environment (MIDSE). MIDSE is a research prototype of an information environment that will enable JSC managers and employees to more efficiently access information in JSC's databases.

A 1986 JSC report, "The Strategic Plan for Information Systems", identified a key problem. While the operational databases at JSC were well developed, managers and other employees could not retrieve information quickly or easily. The report recommended that access needed to be improved.

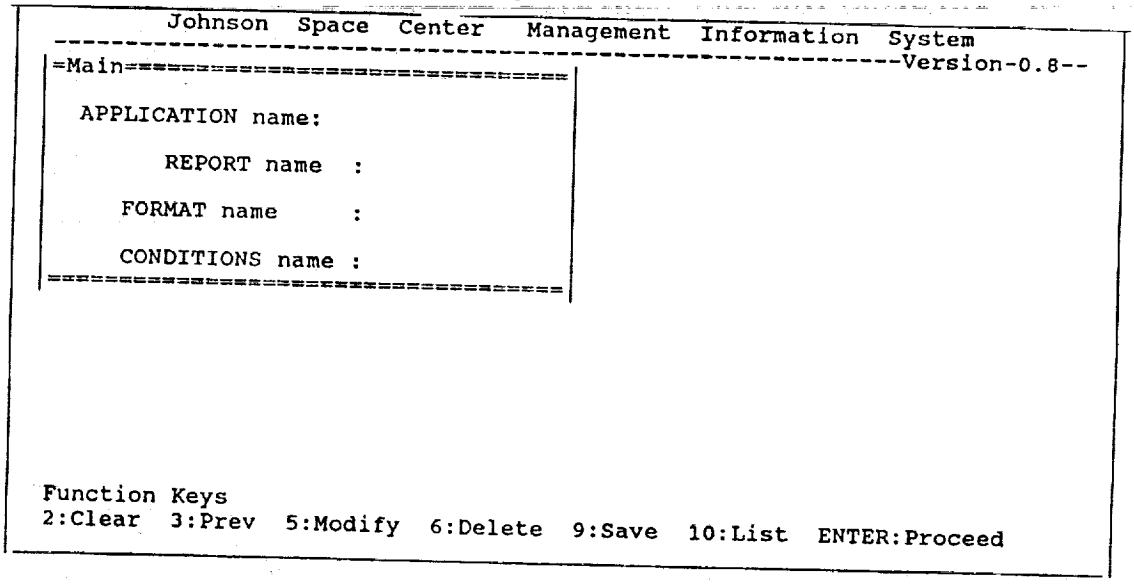
MIDSE is the RICIS response to that need. Briefly, the information environment hinges on a common user interface for all NOMAD2 databases on the JSC Center Information Network (CIN). Figure 4 is the first screen of that interface, the Johnson Space Center Management Information System (JSCMIS). The interface uses the new mainframe window technology available with the NOMAD2 programming language. The interface also operates according to the best principles for the human use of computers, specifically:

the new mainframe window technology available with the NOMAD2 programming language. The interface also operates according to the best principles for the human use of computers, specifically:

- users can select input parameters in the order that best suits their job needs;
- parameters are backed up, not only with information to help use the interface, but also the database itself;
- the interface keeps track of details and presents them on request;
- users can see the results of their selections almost immediately, and modify those selections as they wish; and
- users can save their work at any time and retrieve it later.

The interface currently can produce reports from a special edition of the JSC Personnel database. Work is now underway to add JSC's financial systems to that environment. The interface eventually will be used with all of JSC 's NOMAD2 databases. Information on other DBMS systems will be ported over to a special NOMAD database or interfaces to be constructed in the DBMS's own 4GL and have the same operational characteristics.

Figure 4



Visual Interface

A second RICIS research prototype is helping with the information management of JSC's photograph's and film footage archives. JSC is the repository for all still, film, video and

collection mount exponentially as its size increases. Dr. Mark Rorvig at the University of Texas at Austin has worked the past year to design an interface specifically suited to this and similarly large repositories of visual material.

The standard strategy to catalog and retrieve images depends on the use of words. Captions and/or keywords are assigned to each image. A user who wants a particular image enters one or more words (linguistic search terms), and the system retrieves associated images.

The system's weak point is that linguistic symbols (words) do not always match visual symbols (images). Not only is it difficult for a cataloguer to use a consistent set of words to describe all images, it is also difficult for a user to select the right words to establish search parameters.

Dr. Rorvig's approach to the problem includes representation of the image itself as a search tool. In this way, linguistic terms can be associated with their images, and searches can be more thorough and more precise.

The visual interface has potential application in any organization that maintains collections of images, including museums, news organizations, publication houses and government agencies. This is another example where existing technology, intelligently applied, can solve pressing problems for those who must manage both information and human resources to obtain high productivity. (See Reference 1.)

Reports

There isn't sufficient time to describe all the current major research projects in information management at this same level of detail. There are several significant reports, however, which I would like to mention.

Last year, Dr. Chris Dede reported the preliminary results of his technology forecast for new knowledge-based documentation systems in the Space Station era. His report has been published and is now available for distribution. (See Reference 2.) The report also was the basis for a national RICIS conference on hypertext and hypermedia, co-hosted by Dr. Dede in September.

Dr. Robert Hodgin completed his survey of the computing capabilities of JSC and the Clear Lake areas, and released his report last Spring. (See Reference 3.) The report contains a brief summary of the computer hardware and software used locally, and a comprehensive list of the aerospace contractors who contributed to this census.

Dr. Robert Mayer of Texas A&M continues his work to develop a formal methodology for software requirements analysis. We

received an interim report from his research team last Spring. (See Reference 4.)

Finally, the Center for Space and Advanced Technology has submitted the drafts of two reports in the final phase of their RICIS activity on the commercialization of the U.S. International Space Station. The first report analyzes the forces and factors that will promote or inhibit space station commercialization. The second report scrutinizes the potential for biotechnology in space. The biotechnology study includes the results of an industry survey indicating that the commercial potential for research in space is higher than anticipated. (See References 5 and 6.)

The final reports of these studies are available through the RICIS office or through the Space Business Research Center. Also, their principal investigators will be happy to discuss their projects in detail with individuals who would like additional information.

Next Steps

Most of these projects will continue into 1989. I expect to report additional results at this symposium next Fall. In addition, the Space Business Research Center is working on two new projects in information management.

The first of these is the AdaNET project to develop a technology transfer network for software engineering and the Ada programming language. AdaNET is funded by the NASA Technology Utilization Office with the assistance of other government agencies. Most of the development work is being conducted by MountainNet, Inc., a West Virginia based firm. The network effort has been undertaken to transfer knowledge, experience, and artifacts - from government projects which have used software engineering principles and/or the Ada language - to the private sector. A central goal of the research is to advance an understanding that software reusability is the theoretical foundation for the next generation of software repositories. AdaNET is a main RICIS project, and I expect you will hear a great deal more about it in the coming months.

Another information management project was started last August for JSC's Space Station Project Office. The project entails designing a communication systems to let office managers communicate the status of their work to each other and to the project manager effectively and efficiently. This deceptively simple requirement, however, has become a problem of enormous magnitude in today's world of large scale projects spanning long time frames. The Space Business Research Center is working with the Department of Decision and Information Science at the University of Houston to help build the system.

Conclusion

All these projects have the same central aim - to use information technology to help people work more productively. Individual projects change as we make progress in many ways. Last year, we discussed goals. This year, we report results. However, much remains to be done. Next year, I expect to report even more of our accomplishments.

REFERENCES

1. Rorvig, Mark E., Research in Image Management and Access: Year Two, August 1988.

2. Dede, Christopher J., Factors Shaping the Evolution of Electronic Documentation Systems, January 1988.

3. Hodgin, Robert F. and P. Bishop, Clear Lake Area Computer Capability Census and Directory, 1988.

4. Mayer, Robert, Methodologies for Integrated Information Management Systems, 1988.

5. The Center for Space and Advanced Technology, Space Station Industrialization, 1988.

6. The Center for Space and Advanced Technology, Space and Biotechnology: An Industry Profile, August 1988.

RICIS RESEARCH REVIEW OF

ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS

submitted by

Dr. Terry Feagin

ABSTRACT

The paper summarizes the research accomplishments of the past year for the artificial intelligence and expert systems areas. Most projects have been underway for only a short time, but overall progress within the areas has been steady and worthwhile. Several projects have already attained their major objectives.

RICIS RESEARCH REVIEW OF

ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS

This past year's research accomplishments in the area of artificial intelligence and expert systems are summarized below.

The first project (Communications and Tracking Expert Systems Study) is being conducted by faculty at the University of Houston-Clear Lake and involves the development of expert, automated software for the support of fault detection, isolation, and recovery from failures in the communications and tracking system on the space station. As a result of this work, a very fast method for isolating single-point and two-point failures in the system has been developed. Simulators for testing the software have been developed and used to evaluate the system. Distributed expert systems have also been studied and developed for work in this area.

Another project (Computer Graphics Testbed to Simulate and Test Vision Systems for Space Applications) is underway at Rice University. During the first year of work, this project has seen the development of a preliminary graphics testbed and physical models have been constructed to permit comparison with the graphics models. A graphics testbed has been developed using a Sun workstation to permit control of the light source direction with gray shading or Gouraud shading. Physical models include hexagonal cylinders and attachment devices. Evaluation is underway as to the suitability of 3M VDL vision systems.

Another project has culminated at the University of Michigan in 1988. The project involved path planning for robotic equipment, including foreign code encapsulation and automating the process.

Another project, underway at Massachusetts Institute of Technology, concerns the development and application of fuzzy sets and related theories to failure detection and control in space systems.

A project currently being carried out at Rice University involves the demonstration of a 3D vision algorithm for space applications. The research concerns developing object recognition algorithms that are insensitive to object orientation and distance.

At Yale University, the use of the T programming language on the Cray X/MP is being investigated. The language, a superset of Scheme which is a dialect of LISP, is being ported to the Cray with an eye toward making use of parallel computation.

Another project, carried out by Lincom Corporation, involves research and development for onboard navigation and ground-based expert/trainer system.

Object-oriented programming systems (OOPS) and frame representation schemes using Ada are being studied by Softech and Brown University. The possibilities for merging OOPS and Ada has been pursued. The merits and demerits of OOPS as a way of addressing the applicability of OOPS to various programming tasks has been investigated.

A University of Lowell study is underway regarding a unified robotics control system using a parallel CLIPS environment. The goals include identification of enabling and enhancing technologies for space operations, and the application of emerging technologies to problems in space and planetary exploration, with particular attention to ways of increasing computational speed via parallel processing and expert systems.

The last project, underway at Rice University, involves the development of algorithms and software for the recognition and location of single unoccluded objects based on fusion data from a single camera (intensity image) and from a laser range imaging device (range image). The algorithm discriminates the objects from the clutter and obstacles in the field of view.

Most of these projects have been in progress over only a year or two and most results are only of a preliminary nature. Nevertheless, it is apparent that the overall progress within this area of research has been steady and worthwhile. Several projects have attained major objectives already and new results are forthcoming.

Mathematical and Statistical Analysis

Dr. A. Glen Houston

MATHEMATICAL AND STATISTICAL ANALYSIS

Research Goal

The goal of the mathematical and statistical analysis component of RICIS is to research, develop and evaluate mathematical and statistical techniques for aerospace technology applications. Specific research areas of interest include modeling, simulation, experiment design, reliability assessment and numerical analysis.

Research Activities

To date, there has been only one research activity in mathematical and statistical analysis. This research activity is entitled "Space Station Momentum Management and Attitude Control" and referred to as MS.1. This research is sponsored by the Guidance and Navigation Branch of the Mission Planning and Analysis Division within the Mission Support Directorate at NASA/JSC. The NASA technical monitor is David Geller, an aerospace engineer in the Guidance Analysis Section. The UH-Clear Lake technical representative is Dr. Terry Feagin, professor of computer science.

The research is being done at the University of Texas under the direction of Dr. Bong Wie, an assistant professor in the Department of Aerospace Engineering and Engineering Mechanics. Co-investigators include Dr. Jason L. Speyer and Dr. David G. Hull, professors from the same department. Two graduate research assistants also support this activity.

The research effort was initiated February 1, 1987, for a planned three-year period.

The objective of this study is to develop robust fault-tolerant adaptive control techniques for the Space Station. The Space Station Program needs a control system which must accomodate a modular construction of the Space Station and provide attitude stabilization over a wide range of geometry and mass distribution, which will occur during the initial assembly and follow-on configuration growth stages. The study focuses on developing fundamental concepts as well as advanced analytical techniques for designing a robust adaptive control system.

The major accomplishment to date has been the development of momentum and attitude control equations for the Space Station's guidance, navigation and control computer system. In particular, the momentum and attitude control equations have been modified to accomodate both small and large station pitch torque equilibrium attitudes (TEA's). The new control equations utilize quaternion feedback. The modifications were required since the space station navigation computer is planned to provide attitude information in quaternion form, and the space station may be required to maintain large pitch TEA's.

As a result of this investigation, a significant new feature has also been added to the controller. It now has the capability to seek a dynamic TEA. This is a time-varying torque equilibrium attitude that virtually eliminates control moment gyro (CMG) momentum occilations. The new feature plus the above modifications greatly enhance the overall usefulness and flexibility of the controller.

Other accomplishments include the development of algorithms for identifying station mass properties including bending modes and frequencies.

In the coming year, plans are to continue the study and development of adaptable control equations and identification algorithms for the Space Station GN&C flight computer; attempt to define robustness measures that are meaningful for Space Station momentum management control systems; investigate techniques for establishing the inertia bounds in which the station will remain stable; and determine the optimum gain selection technique for the momentum management controller.

Future Plans

For some time, we have been attempting to initiate a research activity related to software reliability modeling. During the past year, exploratory conversations were held with members of JSC's Safety, Reliability and Quality Assurance office. These discussions have now led to a potential relationship between UH-Clear Lake and the Safety Group within the Boeing Aerospace Company.

The plan is for UH-Clear Lake to lead a project to establish a quantitative risk analysis methodology for the software portions of Boeing's computer-aided user-oriented system evaluation (CAUSE) hardware and software analysis model. The intent is to automatically generate detailed fault trees from which single and multiple failure points of systems (due to software) can be identified and analyzed as a part of the risk analysis process.

In support of this project, a workshop is planned to be held in the coming year. Experts in the field will be invited to participate and discuss the issues in modeling software reliability.

Research Review for Software Engineering Education and Training

Glenn B. Freedman, Ph.D.
Software Engineering Professional Education Center
University of Houston - Clear Lake

EDUCATION AND TRAINING

The education and training component of RICIS supports the range of activities sponsored by the cooperative agreement. In particular, this component emphasizes research in technology transfer, information transfer, and dissemination of research in computing and information systems. During the past year the level of research activity in education and training increased, as the overall level of effort in the cooperative agreement increased.

Highlighting the year's activities was the formation of the Software Engineering Professional Education Center (SEPEC). SEPEC assists all RICIS researchers and research sponsors through its conferences, seminars, and technology transfer activities. SEPEC also assists all RICIS components by coordinating cooperative programs and affiliations with various NASA branches, other university groups, and industry. These cooperative relationships link organizations concerned with educational innovation, research and development in software engineering and other computer related areas.

In review, there were seven research activities in the education and training area that were begun or completed during the past year. In this research overview, the activities will be reviewed in the temporal order in which they were funded. For each activity, the title, major points, status and deliverables are as follows:

1. ADA TRAINING SYSTEM (CBATS)

Completed by SofTech, Inc., the Computer Based Ada Training System (CBATS) was developed through funding with the U.S. Navy to meet the need for an Ada reference system that could be on-line and easy to use.

CBATS used hypertext technology to link the Ada reference manual, syntax examples, Ada and software engineering information, and commentary in a straightforward, useful manner.

2. SOFTWARE ENGINEERING AND ADA TRANSACTION COURSE DEVELOPMENT

A coordinated approach bringing together NASA management, UHCL, and SofTech resulted in a field-tested, three day course entitled "Software Engineering: Concepts and Implementation Strategies." The purpose of the course is to provide managers with the software engineering background and Ada technology information necessary to assist with transition to Ada and its related software environments.

Completed in October, 1988, the course is now available.

3. RESEARCH IN INTELLIGENT TUTORING SYSTEMS FOR KNOWLEDGE POOR DOMAINS

This project is sponsored by the US Air Force Human Resource Laboratory at Brooks Air Force Base. Working with the Artificial Intelligence section and flight training branch at NASA/JSC, as well as UHCL, this project will provide a prototype tutoring system for the Mission Operations Directorate to use in training personnel to use the flight control panel, a task that requires coordinating a knowledge base, with automatic motor skills, auditory and visual overload, and a detailed understanding of flight data. The work for this project is being conducted at Southwest Research Institute in San Antonio.

4. SOFTWARE ENGINEERING EDUCATION AND TRAINING IMPLEMENTATION RESEARCH

This research activity will result in new training tapes for NASA, featuring the latest available information on design strategies in support of Ada use. In addition, the activities supports maintenance of an education and training database for courses, resources and services. The activity fosters the dissemination of research information on software engineering. Sponsored by the Mission Support Directorate, this activity focuses primarily on space station applications, intended for the deliverables.

5. PROTOTYPING CAPABILITIES FOR MISSION OPERATIONS DIRECTORATE

This activity provided access to the Mission Operations Directorate to use the facilities of the Advanced Technology Center for training and research purposes. The activity is on-going.

6. HYPERMEDIA TOOLS FOR BUILDING TECHNICAL TRAINING SYSTEMS

This activity, sponsored by the US Air Force's Human Resource Branch Intelligent Systems, investigates advanced knowledge transfer technologies and their application to future training systems. The research is to be conducted by Dr. Christopher Dede and will be directed both to Air Force requirements and those of NASA's space station training offices.

As the project has just begun, results will not be available until late 1989.

7. MICROCOMPUTER INTELLIGENCE FOR TECHNICAL TRAINING -PHASE II

This project features a continuation of work developed by Search Technology for the US Air Force and NASA/Mission Operations Directorate. The project will build on the initial product of a rule-based system to teach shuttle fuel cell understanding through sophisticated simulations of malfunctions. The second phase will extend the capabilities of the system and add an authoring system. The second phase has just begun.

In summary, the education and training component of RICIS assists all other components through dissemination of information, research on innovative systems, and support for advanced technology in education and training settings.

Requirements Analysis Fundamentals

omit

Session Chair: **Bryan I. Fugate**

Moderator: **Michael J. See**

Speakers

Colin Potts

Lawrence Markosian

Requirements Analysis, Domain Knowledge, and Design

Colin Potts

MCC Software Technology Program

Requirements Analysis, Domain Knowledge, and Design

Colin Potts

MCC Software Technology Program

Abstract: Two improvements to current requirements analysis practices are suggested: domain modeling, and the systematic application of analysis heuristics. Domain modeling is the representation of relevant application knowledge prior to requirements specification. Artificial intelligence techniques may eventually be applicable for domain modeling. In the short term, however, restricted domain modeling techniques, such as that in JSD, will still be of practical benefit. Analysis heuristics are standard patterns of reasoning about the requirements. They usually generate questions of clarification or issues relating to completeness. Analysis heuristics can be represented and therefore systematically applied in an *issue-based* framework. This is illustrated by an issue-based analysis of JSD's domain modeling and fucntional specification heuristics. They are discussed in the context of the preliminary design of simple embedded systems.

Introduction

Requirements elicitation and requirements analysis are activities that take place at the *procurement interface* (Potts, 1988a) between customers and developers. Much has been written about the costliness and seeming inevitability of errors made during these activities. These errors are breakdowns in the supposedly shared understanding of the system among customers and developers. The problems are made worse in many large system development projects by two factors: changing and conflicting requirements, and the lack of application domain knowledge in the development organization (Curtis, Iscoe and Krasner, 1988). In this paper, two improvements to current requirements analysis practice are suggested: domain modeling, and the systematic application of analysis heuristics.

The Role of Domain Knowledge

In most development settings, requirements change frequently throughout the system's liftetime, especially during its early design phases. Designs that are structurally based on the functional requirements are particularly sensitive to changing requirements. As requirements are revised or new requirements added the logical basis for the design architecture may be lost. To make a design less sensitive

to changing requirements, one could base the design on anticipated as well as existing requirements. Unfortunately, it is hard to anticipate requirements.

An alternative is to base designs on a model of the application domain. A *domain model* contains the objects, relationships, and concepts that are considered important to the users. It is likely to be more stable than the requirements, because the domain evolves more slowly and less significantly during the lifetime of of a system than its requirements.

There are other advantages to formulating a domain model before specifying the requirements in detail (Bruns and Potts, 1988): terms and concepts defined in a domain model can be used with less risk of misunderstanding; domain knowledge may be shared across projects; the activity of domain modeling acts as a goal-directed familiarization activity prior to specification; and, the existence of a domain model may improve the training of new developers and maintainers.

The use of domain modeling in software design has been proposed before. Greenspan, Mylopoulos and Borgida (1982) give a good overview of the domain modeling approach to software development, and why it was felt to be necessary in the TAXIS information systems design project:

> In considering the development of a variety of information systems we have found it necessary to become initmately familiar with a wide range of subject matters: medical knowledge, hospital procedures, available therapies (drugs, surgery, etc.), legal responsibilities to government, and so on. We believe that this kind of real world knowledge needs to be captured in a formal requirements specification.

A valid domain model can only be formulated by people with sufficient knowledge of the application domain, but this runs up against the relative absence of domain expertise in most development projects. One possible long-term solution is to apply artificial intelligence techniques to requirements analysis. Intelligent requirements analysis tools are envisaged that will be able to use deep knowledge of the application domain to question the analyst about possible inconsistencies or gaps in the stated requirements. Some promising research is underway (Fickas, 1987; Rich, Waters and Reubenstein, 1987), and ultimately the problems of requirements analysis may be addressed by these means. A recent review of domain modeling approaches (Bruns and Potts, 1988), however, concluded that the most effective applications of domain modeling in industrial system development to date have incorporated restricted modeling primitives, modest goals, and a systematic method. Examples of such restricted domain modeling approaches include JSD (Jackson, 1983) and Booch's (1987) object-oriented Ada design method.

Systematic Requirements Analysis

A second major problem with requirements elicitation and analysis practices is their unsystematic nature. To some degree this is unavoidable; requirements elicitation and analysis are inherently open-ended. Much can be done, however; for example by using diagrammatic techniques such as CORE (Mullery, 1985) to increase confidence in requirements consistency. Another place where greater control could be exercised is the interface from requirements to design. Better practical techniques are needed to 'graft' requirements onto design specifications, particularly in view of the inevitability of changes to the requirements once the design is underway. If this could be done, it would have the bonus of improving traceability: that is, the ability to be able to demonstrate that every requirement is satisfied by the design.

Making requirements analysis more systematic is also a goal of the AI research mentioned above. But again, a short-term partial solution is available. Standard requirements analysis heuristics and checks can be represented in a systematic and machine supported (though not mechanizable) framework: the *issue base* (Conklin and Begeman, 1988; Potts, 1988b).

JSD as an illustration

JSD (Jackson, 1983) serves as a good example of a system development method that attempts to fulfill both short-term objectives. It has a restricted form of domain modeling (Bruns and Potts, 1988), and its many heuristics can be cast in an issue-based framework (Potts, 1988b).

In the remainder of this paper, JSD will be used to illustrate domain modeling and issue-based heuristics. First, its design philosophy and its restricted form of domain modeling are discussed. Next, the systematic, issue-based nature of its heuristics is illustrated. Heuristics in two phases of the method are discussed: the heuristics that determine what should be included in the domain model, and the heuristics that guide the way functional requirements are introduced into the model or an existing design. Figure 1 shows the procurement interace that is implicitly assumed in JSD and by the quotation from Greenspan *et al.*, cited above. The figure indicates those processes and transitions that are illustrated by JSD in this paper. (Because the flow of information is predominantly from the cus-

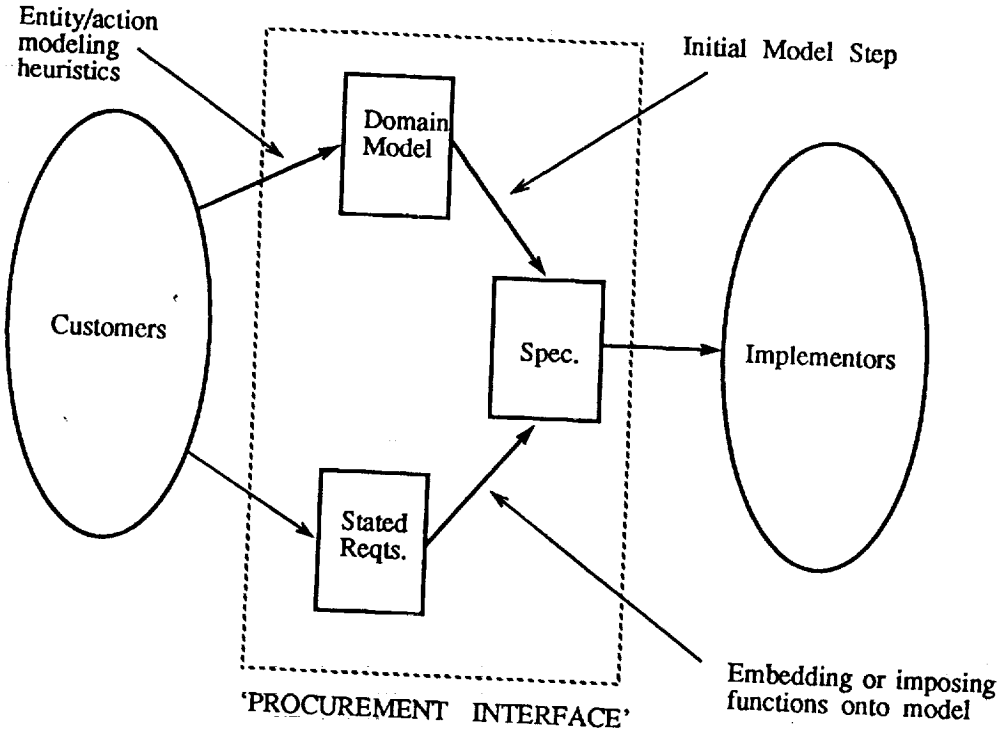


Figure 1: The 'procurement interface' when domain modeling precedes specification. JSD activities and heuristics are shown.

tomers to the implementors, only forward flows are shown. This does not imply that feedback does not occur.)

JSD is only an illustration of how systems can be specified and designed when greater emphasis is given to domain modeling and systematic analysis. JSD is not universally applicable, nor is it perfect where it is applicable.

The Development Method JSD

Although Jackson does not describe JSD this way, JSD consists of two major phases, *specification* and *implementation*, each of which involves a series of steps. The *strategy* of JSD, the major steps and their rationale, is described in this section, and the *tactics*, or modeling heuristics locally applicable within the steps, are illustrated later. The product of the specification phase is an operational specification, which describes the desired functionality. It consists of a set of concurrent processes which communicate asynchronously. The goal of the subsequent implementation phase is to sequentialize the specification until an efficiently implementable amount of concurrency remains. Only the specification phase of JSD will be described will be described further.

The JSD specification phase follows the formula:

System = Model + Function

A system comprises a model of its environment and mechanisms to accomplish its functionality. For example, a patient monitoring system in a hospital must include a model of patients and their vital signs, etc., and operations that perform the required functions, such as sounding alarms when a patient's vital signs fall outside a safe range. A model is created by first analyzing the entities and events of the relevant part of the real world (in the case of a patient monitoring system, the real world is an intensive care unit). The resulting model contains a set of regular expressions, each one specifying the lifecycle of a real-world entity in terms of the actions it performs or suffers. An example is given as a structure diagram in Figure 2. A PATIENT is ADMITted, MONITORed and DE-

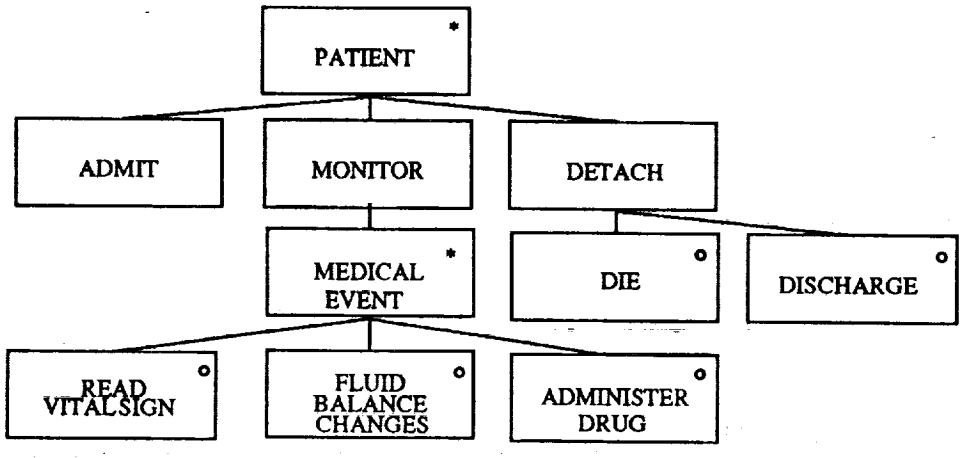


Figure 2: A structure diagram showing the lifecycle of the BED entity of an intensive care unit.

TACHed in that order. The MONITOR part of the lifecycle is itself an iteration of MEDICAL EVENTs, which may be a READ VITAL SIGNS action, a FLUID BALANCE CHANGE, or an ADMINISTER DRUG action, and so forth. The content of a JSD domain is restricted to only those entities and their properties that the system needs to model to execute successfully. Other domain knowledge that may be useful to the developers is excluded. For example, a patient monitoring system needs to model the patient, because it is required to produce statistical summaries of a patient's vital signs and to warn the nurse if any of them becomes threatening. It does not need to model a nurse's legal responsibilities or hospital policy — however important this information may be — because the system itself is not required to act directly on this knowledge. Thus, a JSD model is more restricted in scope than a domain model might be.

Next, an 'initial model' is constructed, in which (to simplify slightly) every entity is connected to a monitoring process inside the system boundary (see Figure 3).

The functional component of the specification is not produced in JSD until after the initial model is complete. Jackson does not discuss the elicitation of requirements in any detail, but it is clear from his examples that it is essential to have some skeletal requirements documented so that an analyst can judge which entities are relevant to the system and must be modeled.

Once the requirements have been elicited and documented — however that is done — desired functions can be introduced into the initial model to create the full system specification. Simple reporting functions can be accomplished by augmenting existing monitoring processes. These functions are said to be *embedded* in the model. More complex functions usually require one or more processes to be introduced that realise the function by combining information modeled by several monitoring processes. These functions are said to be *imposed* on the model. Figure 3 shows the resulting system specification diagram for the patient monitoring system. There are two alarm functions. One is set off whenever the PATIENT entity becomes disconnected from the system (this function might compare the cur-

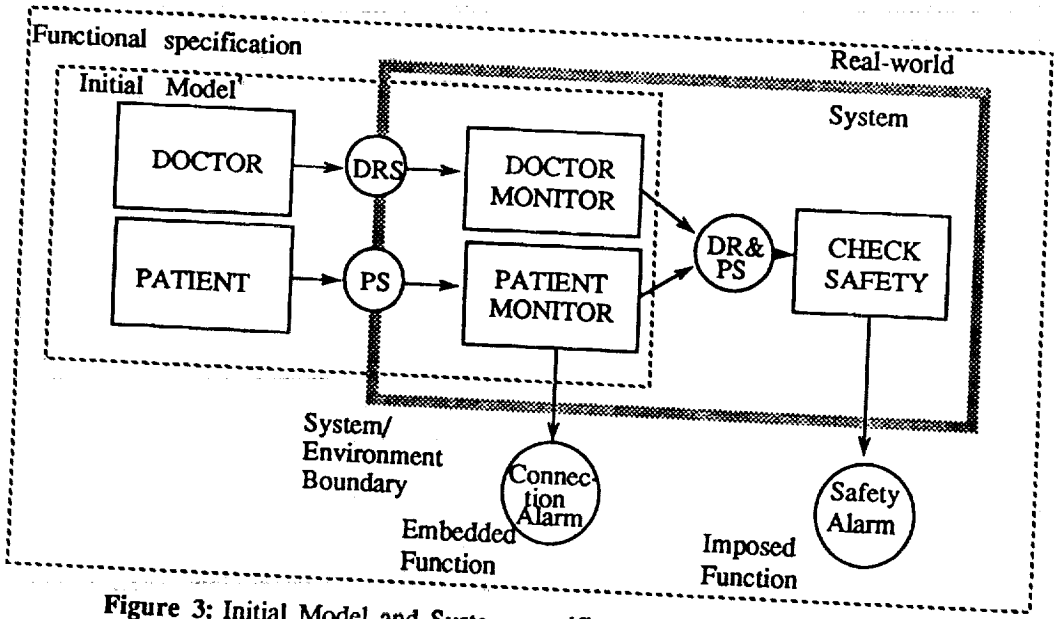


Figure 3: Initial Model and System specification diagram for Function Step.

rent PS value with constants symptomatic of sensor disconnection or failure). This function can be embedded in MONITOR PATIENT. The other alarm is activated when the patient's vital signs fall outside a range defined by the DOCTOR in the input DRS. Since this function requires integration of multiple inputs from the environment, it is an imposed function and requires the introduction of the CHECK SAFETY PROCESS.

Issue-Based Domain Modeling and Specification

Before illustrating some of JSD's modeling and specification heuristics, the issue-based framework that is used to represent it will be introduced.

The Issue-Based Framework

In the *issue-based* framework (Potts, 1988b), there are only five kinds of entities: *artifacts* (method-specific design documents), *steps* (revision, refinement or elaboration operations), *issues*, *positions*, and *arguments*. Artifacts (e.g. data flow diagrams) and steps (e.g. top-down decomposition) are a standard component of all design methods, but the representation of reasoning and rationale in terms of issues, positions, and arguments is less familiar.

The representation stems from Rittel's work in using his IBIS ('issue-based information systems') model to support the discussion of policy and design alternatives in architectural planning (see Conklin and Begeman, 1988 for a summary). *Issues* pose questions about some focus of concern. For example, 'How can the heartbeat sensor fail in such a way that the patient's vital signs appear threatening?' is a domain-related issue. A *position* is a candidate response to an issue, such as 'Heartbeat sensor failure mode F gives rise to apparently threatening vital signs'. An argument may support or object to a position. For example, an argument that supports the above position might be 'Failure mode F apparently quadruples the heart rate because of a masking error', whereas an argument that opposed the position might be 'Failure mode F can always be detected independently by self-test procedure P'. Issues, positions or arguments may spawn sub-issues. For example, a sub-issue raised by the last argument could be 'Can self-test procedure P be run sufficiently fast whenever the patient's heartrate exceeds X beats per minute so that it can be ascertained within a therapeutically safe interval whether Failure mode F has arisen?'

Issue-Based Reasoning in the Early Stages of JSD

Figure 4 illustrates a small part of an issue-based representation of the heuristics of JSD. Ignore for the moment the nested boxes, and consider only the heavy outer boxes. These represent the five basic entity classes of the representation. From the names of the relations in Figure 3 it can be seen that steps modify artifacts, issues review artifacts, positions respond to issues, and so on. Within the heavy outer boxes we see that — for example — JSD's issues form a taxonomy of classes. Although positions in general respond to issues, we see that only some JSD positions are valid responses to particular issues. For example, the position sub-class not entity! is a valid response to issues of sub-class entity?, which address the question whether a candidate entity is an entity in the model according to JSD's modeling criteria. Furthermore, arguments of type not individuatable can support this type of position, whereas arguments of type not OMB (for 'outside the model boundary') cannot. The legality of relationships is inherited. For example, any issue of type EA check, including enti-

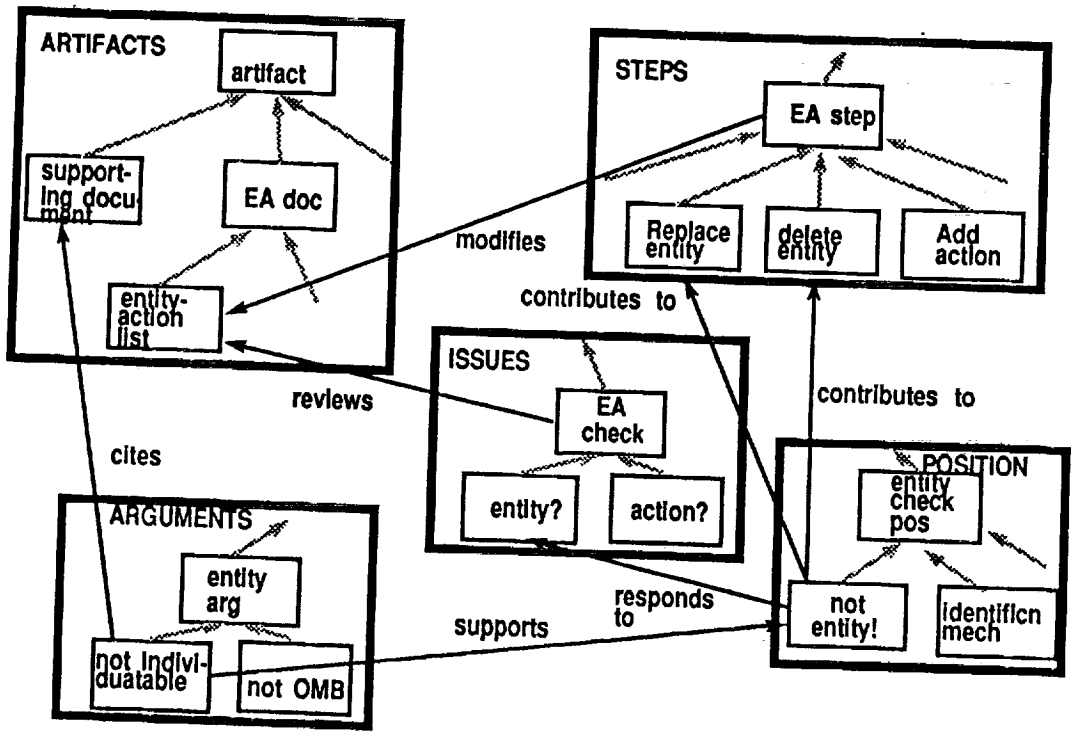


Figure 4: Part of the Entity Class Hierarchies for JSD with relationships.

ty? and action? issues, can be raised to review the JSD entity-action list document.

Example

Figure 5 shows some of the JSD sub-classes from Figure 4 instantiated for a specific domain modeling situation. It is taken from Jackson's example of an elevator system. The issue in question is whether the elevator system needs to model the PASSENGER. This issue is raised to review the entity-action list (a JSD document that includes much of the JSD domain model information). This issue is of a type that should always be raised concerning candidate entities in the model. Because the issue is a yes/no question, there are two positions attached to it: the selected position that PASSENGER is not an entity, and the position that it is. To save space, only the selected position is shown in the diagram. Several arguments can be made for or against excluding an entity from a JSD model. One of them, of type 'not individuatable', is shown. This states that PASSENGER cannot be an entity type in the model, because the system has no way of keeping track of individual passengers. It must be able to respond to requests, of course, but it has no way of telling, for example, which passenger issued a request, or whether the passenger who issued a downward request at one floor is the same passenger who requested to get off at the ground floor.

The point of this example is to show that this kind of questioning and reasoning occurs in all JSD model developments. The details change, but the presence of entities in the model has to be questioned, there are standard criteria for including or excluding entities, and so on. The issue-base, a frag-

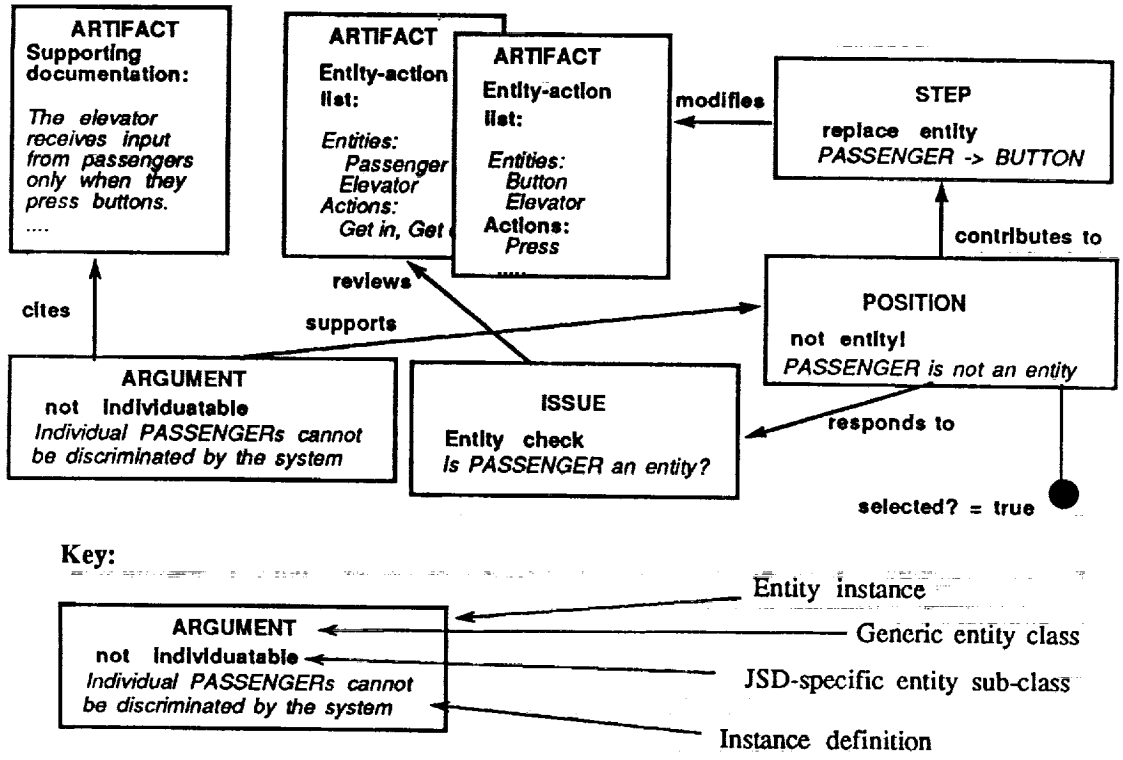


Figure 5: Simple (PASSENGER replacement) design episode

ment of which is shown here, is a network structured template.

The preceding illustration is atypically simple and the reasoning is restricted to a small part of the model. The topic is the content of the entity-action list, and the evaluation of a small number of modeling criteria are sufficient for the exploration of the alternatives. Most decisions are more complex than that: they range more broadly and necessitate more open-ended reasoning. Potts (1988b) contains a detailed example, also from Jackson's elevator system, of the reasoning behind the way the scheduling function is imposed on the initial model. This involves several interconnected analyses, artifacts and sub-issues.

As an analogous, though abbreviated account, consider the introduction of the CHECK SAFETY process of the patient monitoring system specification (see Figure 3). The reason for introducing CHECK SAFETY, explained in terms of the issue-based framework, is as follows: an issue is raised concerning the requirement that a safety alarm should be sounded whenever the patient's vital signs fall outside the doctor's specified bounds. The issue is how the requirement should be supported: by embedding inside DOCTOR MONITOR, or (more likely) in PATIENT MONITOR, or by the addition of a new imposed function process. Many subsidiary issues will be raised at the same time: for example, how are the bounds to be specified? how often should the comparison be made? are the bounds all single valued or can the doctor specify logical combinations of different vital signs? etc. The usual reason in JSD for introducing a process to support a required function is that it must combine inputs from several monitor processes or other function processes. This is the case here. The algo-

rithmic structure of the process will directly reflect the structure of the merged data stream DR&PS, so another issue is raised concerning its structure. Some of these details sound like a detailed design questions, but really concern the frequency and ordering of events in the world and the acceptability of system responses to the inputs it receives. To determine the structure of the DR&PS data stream and the degree of fairness required of CHECK SAFETY several detailed questions need to be answered. For example, how frequently should the CHECK SAFETY process check for new criteria from the doctor? Should new criteria take effect immediately? What does 'immediately' mean here? How long a delay is acceptable between the doctor entering a new criterion and its taking effect in the comparison process of CHECK SAFEFY? These issues address the requirements directly, but they might not be raised if the interconnected procedures of JSD did not trigger them.

Conclusions

In this paper it has been argued that domain modeling is more important early in the development process than functional specification, and that forms of domain modeling are feasible. It has also been argued that a more systematic approach to requirements analysis is feasible, and that an issue-based framework is useful for capturing the heuristics of a method. JSD illustrates both suggestions.

Other methods than JSD also illustrate the role of domain modeling and systematic process in specification and design. Object-oriented methods like Booch's (1987) seem different from JSD, but are based on a similar principle: that a design should be derived from a domain model. Even some structured analysis methods (e.g. Hatley and Pirbhai, 1987; McMenamin and Palmer, 1984) have moved away from top-down functional decomposition toward an 'outside-in' event-driven methodology. In all these cases, not only does the analyst strive to understand and model the system's environment before specifying the system's functionality, but the form and content of the resulting 'domain model' have a major impact on the subsequent design process.

Domain models can be expressed with richer semantics than JSD's. Gist (Balzer, Goldman and Wile, 1982) introduces relationships and provides several kinds of semantic constraints (as opposed to event ordering restrictions or cardinality constraints). In RML (Greenspan, 1984), some of the meaning associated with relationships can be described, and a general inheritance mechanism is introduced. In the Requirements Apprentice (Rich et al., 1987), domain knowledge is organized in frame-like clichés. In KATE (Fickas, 1987) constraints can be weakened into normative policies, and domain behavior can be described in typical usage scenarios.

Some informal development methods provide the analyst with many analysis heuristics — CORE (Mullery, 1985) and JSD are especially noteworthy in this respect, although other methods like Booch's or Hatley and Pirbhai's (1987) version of structured analysis for real-time systems are also quite prescriptive. The issue-based framework was not designed specifically with JSD in mind, and may be equally suitable for other prescriptive methods.

The specialization mechanism allows the simple generic model to be customized for a prescriptive development method, or any application domain in which the modeling issues are well-understood. Although it is conceivable that intelligent tool support could be developed with a knowedge-base that corresponded to that of the issue-based framework, more modest goals seem more promising in the short-term. The issue-based analysis of a method or domain could be used as a manually imposed discipline when using the general-purpose issue exploration tool gIBIS (Conklin and Begeman,

1988).

It must be stressed that JSD has not been the topic of this paper. It is only an illustration of the ways in which informal development methods that are based on sound design principles can lead to a more systematic approach to the earliest phases of system development. They can do this in two ways: by encouraging the explication of application knowledge in a formal description of the domain, and by providing the analyst with a structured network of checks and heuristics.

Acknowledgements

Some of the content of this paper is derived from a review of domain modeling approaches written jointly with Glenn Bruns, who also commented extensively on an earlier draft.

References

Balzer, R., N.M. Goldman and D.S. Wile, 'Operational specification as the basis for rapid prototyping' *ACM SIGSOFT Software Eng. Notes, 7*(5): December, 1982.

Booch, G. *Software Engineering with Ada*, Benjamin Cummings, 2nd Edition, 1987.

Bruns, G. and C.Potts, *Domain Modeling Approaches to Software Development*, MCC Technical Report, STP-186-88, June, 1988

Conklin, J. and M. Begeman, 'gIBIS: a hypertext tool for exploratory policy discussion', *ACM Trans. on Office Info. Sys.*, October, 1988.

Curtis, B., H. Krasner and N. Iscoe 'A field study of the software design process for large systems', *Comm. ACM, 31*(11), November, 1988

Fickas, S. 'Automating the analysis process: an example' *Proc. 4th Int. Workshop on Software Specification and Design*, IEEE Comp. Soc. Press, 1987.

Greenspan, S.J., *Requirements Modeling: A knowledge representation approach to software requirements definition*, Univ. Toronto, Technical Report CSRG-155, March 1984.

Greenspan, S.J., J. Mylopoulos and A. Borgida, 'Capturing more world knowledge in the requirements specification' *Proc. 6th Int. Conf. Software Eng.*, IEEE Comp. Soc. Press, 1982.

Hatley, D.J. and I.A. Pirbhai, *Strategies for Real-Time System Specification*, Dorset House, 1987.

Jackson, M.A., *System Development*, Prentice/Hall, 1983.

McMenamin, S.M., and J.E. Palmer, *Essential Systems Analysis*, Yourdon Press, 1984.

Mullery, G.P., 'Acquisition - Environment' in M.W. Alford, J.P. Ansart, G. Hommel, L. Lamport, B. Liskov, G.P. Mullery and F.B. Schneider (eds.) *Distributed Systems: Methods and tools for specification - an advanced course*, Springer-Verlag, LNCS 190, 1985.

Potts, C., 'The other interface: specifying and visualizing computer systems' in T.R.G. Green, G.C. Van der Veer and D. Murray (eds.), *Working with Computers: Theory versus outcome*, Academic Press, 1988(a).

Potts, C., *A Generic Model for Representing Design Methods*, MCC Technical Report, STP-312-88, 1988(b)

Rich, C., R.C. Waters and H. Reubenstein, 'Toward a requirements apprentice', *Proc. 4th Int. Workshop on Software Specification and Design*, IEEE Comp. Soc. Press, 1987.

Knowledge-based Requirements Analysis for Automating Software Development

Lawrence Z. Markosian

Reasoning Systems, Inc.
1801 Page Mill Road
Palo Alto, CA 94304

Knowledge-based Requirements Analysis for Automating Software Development[1]

Lawrence Z. Markosian

Reasoning Systems, Inc.
1801 Page Mill Road
Palo Alto, CA 94304

Abstract. We present a new software development paradigm that automates the derivation of implementations from requirements. In this paradigm, informally-stated requirements are expressed in a domain-specific requirements specification language. This language is machine-understable and requirements expressed in it are captured in a knowledge base. Once the requirements are captured, more detailed specifications and eventually implementations are derived by the system using *transformational synthesis*. A key characteristic of the process is that the required human intervention is in the form of providing problem- and domain-specific engineering knowledge, *not* in writing detailed implementations. We describe a prototype system that applies the paradigm in the realm of communication engineering: the prototype automatically generates implementations of buffers following analysis of the requirements on each buffer.

Introduction. Our goal is to increase software development productivity by automating the development process. We attack several weaknesses in current software development models:

- lack of formal connection between requirements and code,
- emphasis on manual production of code, an error-prone process, and
- inability to reuse previously-developed code.

Our approach is to provide domain-specific *requirements specification languages* that allow machine-capture and machine-understanding of requirements in a particular domain. Next we provide *very high level compilers* that automate the generation of more detailed specifications and implementations (code) from the requirements specifications. These compilers are also domain-specific and embody knowledge about how to generate specifications and implementations in particular engineering domains. Thus the compilation process occurs in a knowledge base, and every step in the generation of code from requirements is explicitly represented in this knowledge base. The software development environment that we propose tracks design and implementation decisions made by the user as well as those made by the system itself. Because the *process* as well

[1] The work reported in this paper has been partially supported by the Naval Oceans System Center under U.S. Navy contract N00039-86-C-0221. The views and conclusions expressed in this paper are those of the author and should not be interpreted as representing the policies of the U.S. Government or any agency thereof.

as the *products* of software development are machine-captured, the entire development history is available for analysis, and future applications in the same domain can be derived in part by a replay of earlier derivations.

Our prototype development environment is for the realm of radio communications. The underlying software development environment, on which the communication-specific automated environment is based, is REFINE™. REFINE is a specification-oriented, general-purpose software development environment. REFINE has many features that support the development of domain-specific languages, the capture of domain-specific requirements and programming knowledge, the synthesis of executable code from specifications, and recording and replay of the derivation process.

Related work. Our work is a continuation of research in program synthesis and automatic programming initiated by Green *et al.* [1]. Rich and Waters [2] discuss this work and provide an extensive collection of literature on the domain of program development by transformational synthesis. Barstow has writtren extensively on domain-specific automatic programming systems in [3] and the state of the art in transformational programming in [4]. Kelly and Nonnenmann [5] have developed a knowledge-based approach to the synthesis of communication protocol specifications from informal scenarios of typical system operation.

Enabling technologies. Our knowledge-based approach to generating programs from requirements is based on *transformational synthesis*. In this process, requirements that have been captured in the knowledge base are analyzed and refined into detailed specifications and then into executable code. At intermediate stages in the process there may be parts of the requirements that have been refined down to the code level, and parts that have been only partly refined or elaborated but not fully implemented. Each refinement step is incremental, with a specific feature selected for refinement at each stage. The computational model used is that of applying *transformation rules* that take a partially detailed piece of the program structure and map it into a more detailed structure to which further transformation rules may be applied. For example, transformation rules refine sets into lower-level data types such as arrays, lists or hash tables. The refinement is guided by the engineering and programming knowledge base. Each application of a transformation rule represents a design decision, and these design decisions are recorded in a derivation tree that enables the user to return to an earlier stage in the refinement in order to play out alternatives to previous design decisions. Transformational synthesis has been applied successfully to program generation (REFINE) and to automatic generation of mission plans for robotic vehicles. In these applications the synthesis process procedes entirely automatically. In the prototype discussed in this paper, the process is interactive, with the user supplying engineering decisions when the system's knowledge base is inadequate to identify a refinement.

An important feature of the transformational synthesis process is that transformation rules are *correctness preserving*. Each rule that is applied to a partially refined set of requirements preserves faithfulness of the partial refinement to the requirements. Correctness-preservation is a property of transformation rules that has to be ensured when the development environment itself is built. But it needs to be ensured only this one time instead of each time a new application is built.

We contrast the representation of knowledge about requirements, specifications and programs with traditional approaches, such as using a subroutine library to generate different implementations. As we will indicate later in this paper, the use of transformational rules yields an exponential productivity gain over the use of subroutines—knowledge that is represented explicitly is more far generally applicable than knowledge encoded in the form of procedures.

The REFINE system. The basic REFINE system [6] that underlies our prototype provides a general-purpose *specification language*. The REFINE specification language includes first-order logic and set-theoretic data types such as sets, sequences, mappings and relations. Thus it is possible to write purely declarative functional specifications of system behavior in the REFINE language. In addition, the language is wide-spectrum and supports a range of programming styles, including object-oriented, rule-based and procedural. The REFINE compiler automatically generates implementations of these specifications. For example, the compiler generates low-level data type implementations (lists, arrays, hashtables, etc.) of abstract sets, as well as the appropriate implementations of set-theoretic operations such as union, intersection, membership, equality tests among sets, etc. Customer experience over the 4-year history of REFINE use shows that there is an order-of-magnitude productivity gain writing REFINE specifications instead of programs in a 3rd generation language such as Ada, Lisp or C. A large part of this productivity gain is attributable to the fact that specifications express primarily functionality while programs express both functionality and detailed implementations. The REFINE system is discussed in [7].

Because most of the programming knowledge in REFINE is general-purpose, we believed an even greater productivity gain could be achieved by customizing the system to a particular application area. A key component of REFINE that allows this customizing to be done is the language definition subsystem. This subsystem allows a user to develop an extension to the basic REFINE language, or an entirely distinct language. The language definition subsystem includes a grammar specification system, a parser generator and a printer generator.

The REFINE compiler can also be extended by a user to generate implementations of the extended specification language. An extended compiler will perform semantic analysis of specifications and apply transformation rules to generate implementations.

All of these capabilities are illustrated in the application to communication systems that we describe below.

The problem domain. Our prototype domain-specific requriements analysis and synthesis environment was developed for communications engineering. Specifically it addresses the synthesis of different kinds of buffers to meet different requirements. Before illustrating the operation of the prototype, we discuss requirements analysis issues regarding buffers and the impact of the requirements analysis on implementation.

The purpose of a buffer is to mediate data flow between asynchronous processes, or between a synchronous and an asynchronous process. Hence every time a data flow between such processes is explicitly required or discovered in a system design, it could trigger the need for a buffer. For our discussion we make the simplifying assumption that the buffer feeds a synchronous process. We list several general properties that one needs to know for any use of a buffer:

- the buffer's input data rate,
- its functional behavior and
- the desired type of implementation.

The requirement for a buffrer at a particular point in the development of the design may be represented in a suitable system as a logic assertion:

"If process P1 transmits data to process P2 and P1 is asynchronous and P2 is synchronous then a buffer is required between processes P1 and P2."

Similar assertions specify requirements for buffers under other conditions.

Engineers will expect certain parts of the buffer design process to be similar in all cases and other parts to depend on the particular context in which the buffer is used.

The similarities among implementations can be summarized as follows:

- a data store of some capacity is required;
- buffer behavior is generally (but not always) that of a FIFO queue;
- we may assume data can be removed from the buffer at a constant rate, because the output is assumed to be synchronous;
- we may assume that the mean input rate (over suitable intervals) is equal to the output rate;
- overflow control may be required;
- underflow control may be required.

Differences among implementations will include different values of parameters in the above buffer attributes (e.g., the actual capacity of the buffer); also, we expect

- different specifications of functional behavior for different requirements, and
- different implementations in different environments.

Input data rate requirements analysis. We first consider three possible characterizations of input data rate:

- approximately constant,
- normally distributed and
- bursty.

Alternative implementations of the buffer store are appropriate for the different characteristics of input data rate:

- approximately constant: fixed-length array, length dependent on:
  - — input rate distribution,
  - — output data rate and
  - — requirement on avoiding saturation.[2]

[2] A model for determining buffer length is given in Lynch [9]

- normal distribution: dynamically-allocated structure such as a list
- bursty: a combination of array and list.

Overflow control requirements analysis. Next we consider possible requirements on buffer behavior under overflow or near-overflow conditions. Possible choices of overflow behavior include:

- input process blocks,
- no blocking; oldest data lost,
- no blocking; most recent data lost,
- feedback to an input filter control,
- feedback to an input aperture control,
- feedback to an input sampling control and
- adaptively changing the buffer store size.

Each of the types of overflow control is appropriate in some situations. For example, in a textbook specification of a buffer as a FIFO queue, the input process typically is represented with a guarded command that waits (blocks) until space is available. In a radar system, a buffer receiving plot data may overwrite the oldest plot data. In a mouse click handler on a workstation, the buffer may drop the most recent mouse-click data because the display handler cannot keep up with rapid clicks and thus the most recent clicks are unlikely to be meaningful. In digital transmission of analog signals, feedback to control the sampler, aperture or filter may be appropriate.

Underflow requirements analysis. Underflow control is often necessary to insure that there are no transmission gaps, which could cause loss of synchronization and an eventual loss of data. Possible ways of handling underflow include:

- output process blocks
- output process extracts a "null" data item
- feedback to an input filter control, etc.

The discussion thus far should provide an idea of the kind of requirements a communications engineer must analyze and the resulting design decisions that must be made in building a system to meet these requirements.

Buffer analysis requirements and synthesis in REFINE.

We indicate how knowledge about buffers is represented in our REFINE-based prototype, and how the requirements analysis and code synthesis process proceeds. (A more detailed discussion of the process can be found in Ladkin *et al.* [8].)

- Buffers are an object class in the knowledge base with attributes that include:

— a partial specification, common to all buffers and therefore associated with each instance of the class

— input data rate requirement

— output data rate requirement

— the partial implementation of the buffer, in its partially refined form

Figure 1 shows some of the attributes of the buffer object class and the possible values of these attributes. The order in which the attributes appear reflect approximately the range from highest abstraction level (requirements) to lowest (coded implementation).

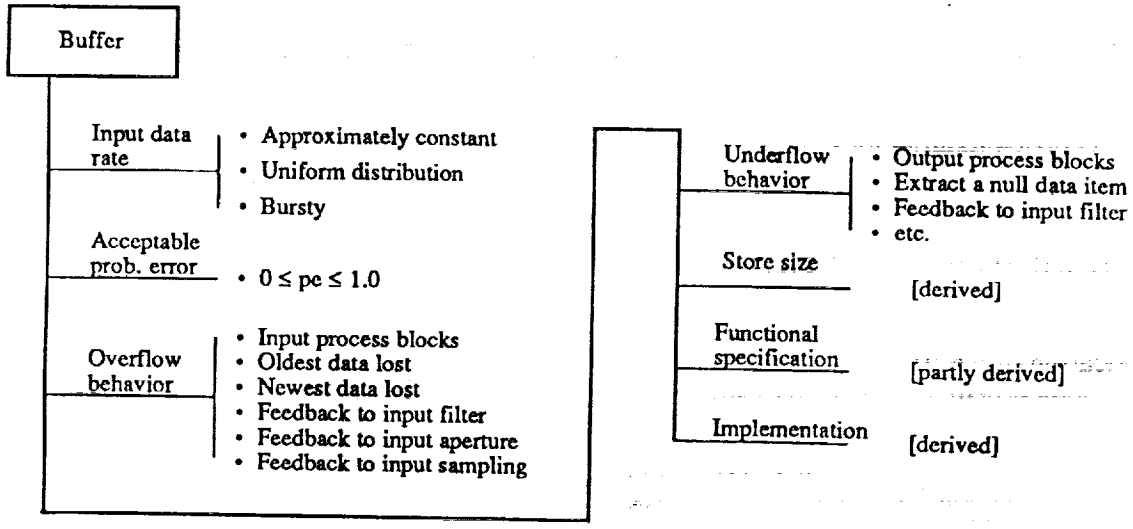


Figure 1: Buffer object class, attributes and possible values for attributes

When the need for a particular buffer is recognized by the system, an instance of the buffer object class is generated. The *values* of the input data rate, output data rate, and partial implementation attributes may at this point in the development be undefined. They will become defined, and may change, as the system draws conclusions about the use of the particular buffer.

Note that in particular a knolwedge base object representing a buffer contains the partial implementation of the buffer as an attribute. It is this feature of our representation, along with the retention of previous KB state via the context mechanism, that enables backtracking to previous design stages, and reimplementation of the buffer with a different design decision.

Here is an example of an abstract buffer instance in the application-specific language developed for this prototype:

```
the-buffer BUFFER-1 with-input-data-rate 9600 baud
   with-discipline FIFO with-underflow-action
   EXTRACT-NULL has-element-data-type CHARACTER
```

Some of the attributes of the buffer have been defined—for example, the input data rate. The value

(9600 baud) may have been provided as part of the original problem requirements or it may have been derived by the system or supplied interactively by the user. This specification is *abstract* because not all the attributes of the buffer have been defined—for example, the capacity of the buffer store has not yet been defined, nor is there yet a partial implementation.

- Other communication system components, such as data compression, noise immunization and encryption devices, and channels are also represented using object classes.

- Assuming the requirements have been formally specified, they are also represented as objects in the knowledge base. Each requirement is represented in the form of an annotated abstract syntax tree. These trees are annotated further as requirements analysis proceeds.

An example of a partial requirement on the larger communication system in which this particular buffer is embedded is the following:

```
The-comm-system CS-1 with transmitter ALPHA-3
   communicates ALPHANUMERIC data and
   has SPORADIC traffic distribution
   with nominal rate 50K baud and peak rate 65K baud.
```

This is a fairly high-level requirement and says nothing about implementation details except that a transmitter of type ALPHA-3 is to be used. The language in which this requirement is expressed is a domain-specific requirements language that was defined using the REFINE language definition subsystem. Its expressiveness and syntax is intended to be comparable to the informal language in which engineers state system requirements. The parser in the prototype system generates an abstract syntax tree. The abstract syntax tree is annotated with several attributes of the communication system that do not appear in the original requirement. Values for these attributes were derived by the system as analysis proceeded. One of these attributes is the code representing the complete implementation of the communication system.

- Logic assertions relate buffer properties to system properties such as input data rate, and these properties are maintained throughout the synthesis by the logic constraint maintenance facility in REFINE.

An example of such an assertion is that the buffer capacity is to be computed using a particular mathematical model from the input and output data rates and the allowable error rate due to loss of data. A model for computing buffer capacity can be found in Lynch [9].

- Transformation rules represent possible refinements of a buffer

Most of the transformation rules in the prototype determine values of undefined buffer attributes from the context in which the buffer is used. One simple, high-level rule derives the discipline of the buffer (FIFO, LIFO, etc.) from the fact that it is being used as the transmitter's output buffer. All the interactive rules in our prototype—those that require information to be provided by a human—are of this high-level type: they require either additional information about the context to be provided, on the basis of which the system can draw a needed conclusion; or they specifically ask the user to supply some value for an attribute of a buffer. In general, the prototype will attempt to derive needed buffer characteristics by backward chaining to the original requirements or consequences of the initial requirements. If the requirements are insufficient to supply the answer, or if the prototype's rulebase is inadequate, the user will be asked to supply a value.

Other transformation rules are more complex and determine details of the buffer specification and implementation. An example is given in Figure 2. This transformation rule derives part of the buffer's detailed functional specification (the buffer store). The specification to be derived depends on, among other things, the type of the data to be stored and the required overflow action. These rules, which are fired after the high-level attributes have been determined, are entirely automatic and do not require user intervention.

Thus the user supplies engineering knowledge and guidance, while the system handles programming details. In particular, the underlying REFINE system is capable of deriving implementations automatically from complete functional specifications, and thus the user concentrates on higher-level, application-oriented tasks rather on coding.

```
Rule Make-Data-Structure-for-buffer-contents (a: DATA-BUFFER)

A = '##r comm-grammar
       the-buffer @n
         with-overflow-action @o-a
         element-data-type @d-t
         implementing-data-structure @undefined'

and defined?(d-t)

and impl-ds = new-var(build-symbol-name([n, 'data]),
                      (if o-a = 'flush-newest-require-reset then
                       'tuple(integer, seq(@(copy-expr(dt)),
                              boolean)' else
                       'tuple(integer, seq(@(copy-expr(dt)))' ))
-->

variable?(impl-ds) = true

and initial-value(impl-ds) =

   (if o-a = 'flush-newest-require-reset then '<0, [], true>' else
      '<0, []>')

and scope(impl-ds) = 'global

and bufer-data-structure(a) = impl-ds

and lisp-code(impl-ds) = undefined

and lisp-initialization(impl-ds) = undefined)
```

Figure 2: Transformation rule to construct buffer store

Note that, in general, if a buffer has n high-level properties and each of these attributes have m possible refinements, then representing each refinement as a transformation rule requires roughly

$n * m$ transformation rules. On the other hand, use of subroutines to represent this knowledge would require on the order of m^n subroutines. Thus representing engineering knowledge about buffers in the form of transformation rules represents an exponential decrease in the amount of code that needs to be written, and a corresponding increase in reusability.

Summary and Conclusion. We have described and illustrated a new approach to system development that uses the requirements analysis process to construct detailed specifications and implementations. The paradigm is based on the use of transformation rules. These transformation rules are correctness-preserving. Transformation rules are a highly reusable knowledge representation technique compared to subroutines in a standard programming language. When it is necessary to make a refinement step but the development environment is unable to proceed automatically, the possible choices are presented to the user in a high-level formulation. Thus the user provides high-level specifications and requirements, and the development environment assumes the burden of developing the detailed program code. Because the system builder is allowed to concentrate on the application problem, primarily requirements analysis and specification, we conclude that our approach holds the potential for very large increases in productivity in software development, and that domain experts—engineers—can replace programmers as the primary developers of software systems.

References.

1. Green, C. C., Luckham, D., Balzer, R. Cheatham, T. and Rich, C., "Report on a Knowledge-Based Software Assistant," *RADC Report RADC-TR-195,* Rome Air Development Center, New York, also *Kestrel Institute Technical Report KES.U.83.2,* Kestrel Institute, Palo Alto, CA, 1983, also reprinted in Rich and Waters [2]

2. Rich, C. and Waters, R. C., eds., *Artificial Intelligence and Software Engineering,* Morgan Kaufman, Los Altos, CA, 1986

3. Barstow, D. R., "Domain-Specific Automatic Programming," *IEEE Transactions on Software Engineering* SE-11, 11 (November, 1985)

4. Barstow, D. R., "Artificial Intelligence and Software Engineering," *Proceedings of the 9th International Conference on Software Engineering,* Monterey, CA, March 1987, 200-211, IEEE Computer Society Press, 1987

5. Kelly, V. and Nonnenmann, U., "Inferring Formal Software Specifications from Episodic Descriptions," *Proceedings AAAI-87 Sixth National Conference on Artificial Intelligence,* July, 1987, 127-132

6. *The REFINE User's Guide.* Reasoning Systems, Palo Alto, CA, 1987

7. Linden, T. and Markosian, L. Z., "Transformational Synthesis Using REFINE™," in Richer, M., and Yazdani, M. (eds), *Artificial Intelligence: Tools and Techniques,* Ablex Publishing, Norwood, NJ (to appear)

8. Ladkin, P., Markosian, L. Z., and Sterrett, A., "System Development by Domain-Specific Synthesis," *Proceedings of the Third International Conference on Applications of Artificial Intelligence in Engineering,* August, 1988, Palo Alato, CA

9. Lynch, T. J., *Data Compression Techniques and Applications,* Lifetime Learning, 1985

Dinner Speaker

Intergrated Software Support Environments: Some Lessons Learned

omit

Frank Belz

(NOTES)

Space Station Software Support Environments

Session Co-Chair: **Everett Lyons**

Session Co-Chair: **Jim Rainy**

Speakers

Tim Porter

Paul Babick

Gokul Bhaumik

Herb Krasner

C. L. Carmody

C. T. Shotton

Lessons Learned from an Ada Conversion Project

Tim Porter and Paul Babick

Abstract.

Recognizing the importance of building software with the robustness to accommodate rapid advances in technology, developers have focused on methods which preserve both past and future investment in software, and in providing an advanced software engineering environment that frees the engineer, to the extent possible, from the more routine aspects of software design and development. The software engineer is permitted to concentrate on the creative aspects of problem resolution. Standard languages such as Ada maximize portability across hardware and operating systems. Standard interfaces which enhance portability and permit the incorporation of new technology as it becomes available have been developed. Software design and development techniques which maximize portability receive increasing emphasis. Experience gained in porting an Ada application between two widely varying environments is evaluated in light of current practices to maximize software portability.

Introduction.

The Software Automated Verification and Validation System (SAVVAS) is an automated tool used to manage and track software requirements during development projects. Developed in Ada on a Digital Equipment Corporation (DEC) VAX/VMS environment, SAVVAS has been ported to the IBM 3090/VM environment and will be delivered to NASA to support software development in the Space Station Software Support Environment (NASA SSE). For the purposes of this paper SAVVAS functionality is immaterial but can be characterized as an information management tool, with a few relatively simple embedded algorithms, and consisting of approximately 25,000 Ada lines of code (LOC). SAVVAS depends on the services of a database management system (DBMS) and was originally designed to use the INGRES relational DBMS. It was subsequently modified to use the ORACLE relational DBMS in the VAX/VMS environment, and uses ORACLE in the IBM environment. Figure 1 illustrates the SAVVAS architecture.

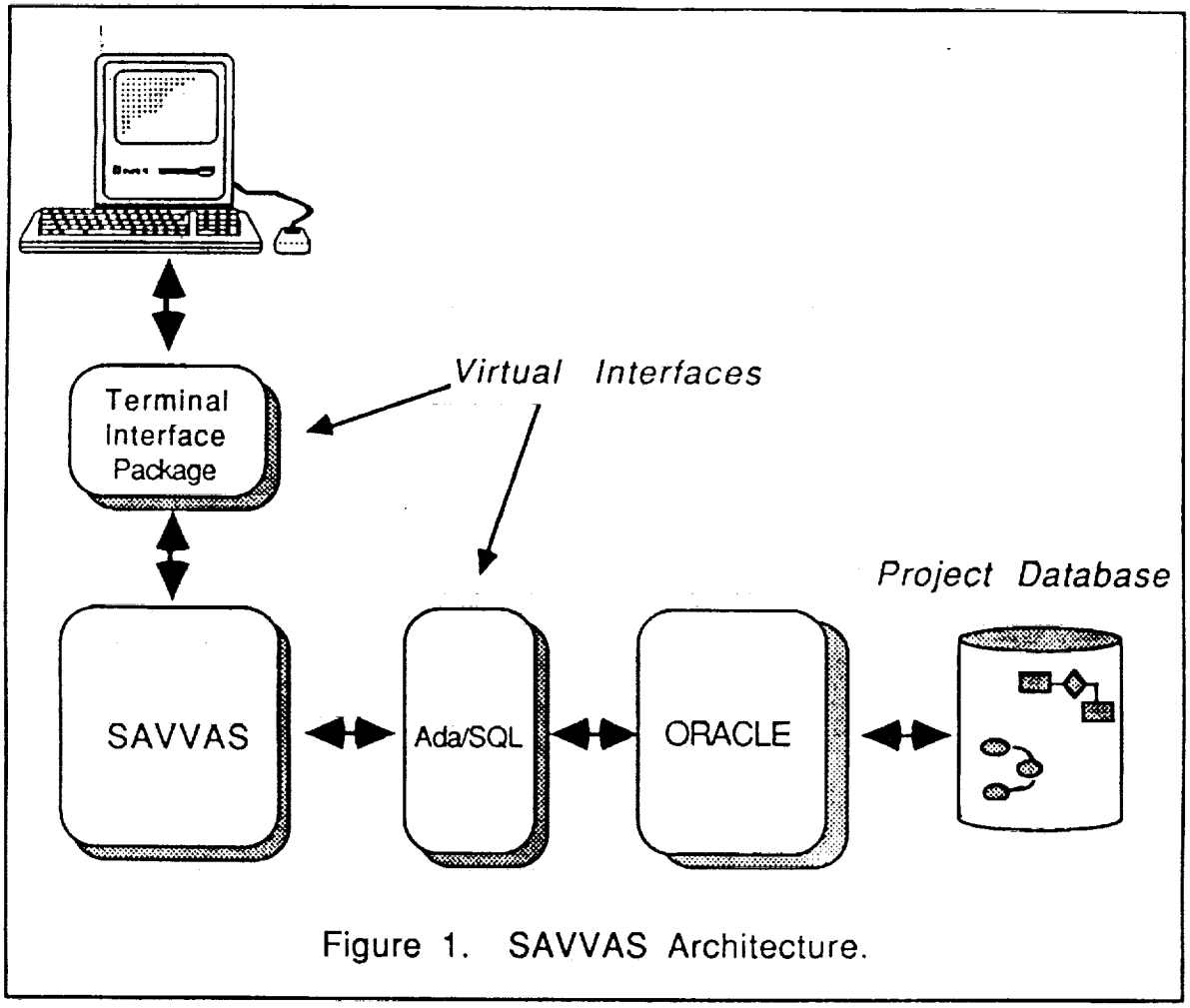


Figure 1. SAVVAS Architecture.

Software Portability.

The degree of software portability is defined as the relative ease with which source code can be moved between alternative hardware, compilers, operating systems, and other external interfaces. High degrees of portability are desirable in order to protect software investment, prolong product life, and promote software reuse. As a result new technological innovations can be easily introduced. Various measures can be taken to improve software portability. However such measures may also adversely impact software performance in several ways.

Applications written in Ada have been reputed to be highly portable. While the Ada language has been standardized (MIL-STD-1815A) and extensive compiler validation tests have been developed to evaluate the degree of standard compliance by Ada compilers, high degrees of portability can be difficult to achieve unless

accompanied by the use of other more important portability enhancing methods and techniques. These include the isolation of non-portable source code, constraints on the use of certain language features, and standard or virtual interfaces. Each of these is reviewed in the context of the SAVVAS port and in light of previous experience.

Isolation of Non-Portable Code.

Rarely is it possible to totally eliminate all non-portable code from application programs. Even simple operations such as cursor positioning on a terminal or text display require the transmission of special control sequences to the display device. System calls to the operating system are usually unique to the operating system. Some operating systems will accept leading or trailing blanks in filenames. Others will not. The cost associated with porting software can be minimized by isolating identified classes of such software to specific Ada packages. This makes the task of finding and correcting non-portable source code much simpler when porting a software application. A trivial example is the set of routines required for terminal input and output. The package specification contains procedure and function declarations with specific machine-dependent control sequences isolated to the package body. Figure 2 illustrates a simple terminal interface package. This package also illustrates another of Ada's advantages in that by isolating machine dependent code to the package body, as illustrated in figure 1, only the package body must be recompiled before the program is re-linked. No other program units become obsolete by virtue of changes to a package body, thus minimizing recompilation time and development costs. In many applications only a few such packages may be required to isolate all known non-portable source code. Experience has also shown that identifying all non-portable source code can be very difficult, and undetected occurrances often result in bizarre program errors that are even more difficult to correct. Clearly lessons learned in this area should be reflected in the organization's software standards and procedures.

```
package SIMPLE_TERMINAL_INTERFACE is

   procedure GO_TO_POSITION (X, Y: in INTEGER);

   procedure DISPLAY_TEXT (MESSAGE: in STRING);

end SIMPLE_TERMINAL_INTERFACE;

with TEXT_IO; use TEXT_IO;
package body SIMPLE_TERMINAL_INTERFACE is

   procedure GO_TO_POSITION (X, Y: in INTEGER) is
   begin
        -- Send the appropriate code sequence to the terminal.
        -- These are different for varying terminal types.
   end GO_TO_POSITION;

   procedure DISPLAY_TEXT (MESSAGE: in STRING) is
   begin
        -- Send the message to the terminal
        -- including any required code sequences.
   end DISPLAY_TEXT;

end SIMPLE_TERMINAL_INTERFACE;
```

Figure 2. Simple Terminal Interface Package.

With respect to terminal interface, the mechanisms available on the IBM 3090/VM system are radically different from those on VAX/VMS systems. The block oriented nature of IBM terminals required significant changes, and in some cases wholesale rewrite, of the human interface modules. This is not surprising in spite of the fact that SAVVAS has a very simple menu-driven interface. IBM's Interactive System Productivity Facility (ISPF) was utilized to recreate SAVVAS menus. The object modules created for each screen were then linked into SAVVAS. Another important design feature which minimizes software porting costs is to use software layers of increasing abstraction. This simplifies conversion to

vastly different environments by permitting the introduction of alternatives at various levels of abstraction depending on the degree of product deviation.

Contraints on the Use of Certain Language Features.

The SAVVAS port as well as previous experiences demonstrates that some Ada language features are less portable than others. For example, previous experience has demonstrated that programs which rely on tasking and which have severe timing constraints may be much less efficient (and may not even work) when ported to other perhaps less optimized, but validated, compilers. Validation of an Ada compiler provides no guarantees with respect to run time performance. There can also be wide variance in the implementation of Ada pragmas. Pragmas are essentially compiler directives. For example the Ada pragma "interface" is used to provide direction to the compiler in the linking of object modules external to the Ada environment, e. g. assembly or other foreign language developed modules. Many compilers provide pragmas that are unique to only a single vendors compiler. Software which relies on such features is clearly less portable than programs which do not.

SAVVAS stores project data in a relational database. This eases data manipulation and report production. It also means that SAVVAS is dependent on some DBMS provided features and is linked to vendor supplied C-language routines. Originally, SAVVAS was developed as a stand-alone tool for the VAX/VMS environment to support U.S. Navy software development projects. The original implementation relied on a pragma unique to the DEC Ada compiler called "pragma IMPORT_VALUED_PROCEDURE" to import object modules outside of the DEC Ada environment such as the DBMS access modules. For most relational DBMSs these access routines are written in the C programming language. The purpose of "pragma IMPORT_VALUED_PROCEDURE" is to specify parameter types and passing mechanisms for linkage to external modules. This DEC-supplied pragma is non-standard but, as stated previously, is allowed under compiler validation rules. It represents but one of many approaches to defining the parameter passing mechanisms required in linking external modules. The Alsys compiler used in the IBM environment employs a very different approach. Needless to say, extensive rework was required because of the rather sophisticated database interface requirements. This extra effort is attributable solely to the use of non-standard language features.

This problem was compounded by the fact that the DEC pragma permits parameter declaration "out of order." Since such out of order parameters in many cases compiled successfully, erroneous programs resulted that were very difficult to debug.

Ada's exception handling is a powerful feature designed to assist in the construction of fault tolerant software. Typically the software designer identifies potential categories of software failure creating specially named exceptions and providing procedures for software recovery in the event of failure. The exception handling feature also provides a pre-defined "others" category of exceptions to be used for unanticipated exceptional conditions. Deeply nested exception handlers each of which has a catch-all "when others" path make for bullet proof programs that won't fail catastrophically. They also make programs extremely difficult to debug since it is virtually impossible to determine at what level the software failed. The SAVVAS experience indicates that the payoff in decreased test and debug time usually exceeds the cost of additional care in the design of exception handlers.

Virtual Interfaces.

The intent of a virtual interface is to isolate application software from perturbations in the external environment. Virtual interfaces have been developed which provide interfaces from Ada application programs to relational database management systems, human interface systems, graphics display devices, and even operating systems. These interfaces are sets of standard calls providing basic facilities for accessing external capabilities. If these interfaces are robust enough, applications are buffered from changes in the external environment. For example technology advances such as a new database machine may be easily integrated into applications without incurring undue software maintenance costs. In a sense virtual interfaces are an extension of the idea that non-portable code should be isolated. In effect virtual interfaces standardize well known classes of non-portable source code. They therefore enhance software portability and protect software investment.

Ada/SQL is a proposed virtual interface to relational DBMSs. It provides standardized native language (Ada) access to relational databases using SQL-like syntax. SQL, or Structured Query Language, is an American National Standards Institute (ANSI) approved

standard for database access. The relative merits of Ada/SQL over other database access approaches, such as embedded SQL or the module approach, are still being debated. It is however a relatively mature virtual interface to relational databases. Applications written using Ada/SQL can easily be interfaced to any relational database management system. Porting to new environments is especially easy once the "standard" non-portable components of the interface have been developed for alternative DBMSs since these modules can simply be plugged into an application and a new DBMS swapped in. The database itself must still be created and populated using the new DBMS but application software remains unchanged. It is significant that during the SAVVAS port no changes were required to the Ada/SQL virtual interface. The source code which implements this interface is identical on both IBM and VAX systems. This capability is critical to large systems representing investments of many millions of dollars. Such systems cannot afford to be "locked into" specific vendor products by relying on the vendor supplied interface procedures.

In both the SAVVAS conversion from INGRES to ORACLE and the port to the IBM 3090, several problems in the database interface area occurred. Some of these required significant amounts of effort to resolve. However all eventually resolved to compiler limitations/bugs or unidentified non-portable software modules. For example, the original software design of SAVVAS assumed that any user disk space could have write access by more than one user - true for the VAX but not so for the IBM. Assumptions such as this are obvious to the "monday morning quarterback" but are sometimes difficult to detect in practice. Assumptions about the underlying environment are often quite subtle and can permeate an entire software design.

Another important evolving interface standard is the X Window System. X evolved from research at the Massachusetts Institute of Technology into a network-oriented windowing system. While not yet an ANSI standard, X has been adopted by an increasing number of computer manufactures, including DEC, DG, Apollo, SUN, and IBM. X was not employed on SAVVAS, but an Ada-X Binding has been developed and employed in numerous software development projects. It has proven to be a versatile, comprehensive interface package. The Ada-X Binding is a formalized virtual interface for the construction of human interfaces and would be used in place of

developer created terminal interface packages such as the simple example included above.

Standards for graphic displays are also important. Several have been proposed such as the Graphics Kernal Standard (GKS). An Ada-GKS Binding has been developed to virtualize this interface for Ada applications.

Standard operating system interfaces such as POSIX have also gained considerable momentum. Other candidates include the Common APSE Interface Set (CAIS) and the European-sponsored Portable Common Tools Environment (PCTE). Each of these are attempts to provide a standard set of operating system primitives. POSIX is receiving widespread support in industry, government and academia. Ada-POSIX bindings are being developed. Programs such as the NASA SSE are evaluating these alternatives in the hope of finding a suitable standard. A virtual interface to the operating system would eliminate many of the portability problems which result from subtle assumptions about the external environment such as the user disk space issue discussed above.

Conclusion.

Clearly a consistent software design methodology coupled with design and coding standards which enforce effective modularization and limit the use of less portable language features is essential to achieving a high degree of software portability. Standards and guidelines should be constantly reviewed and updated based on new insights and experiences.

Virtual interfaces such as Ada/SQL, the X Window System, POSIX and Ada-GKS significantly contribute to software portability, and in addition have significant productivity implications. These tools should be incorporated into Ada software libraries and made readily available to the software development staff. On the other hand they also clearly add to program performance overhead, and must be weighed in light of performance requirements.

The SAVVAS port has validated past experience and cautions with respect to achieving software portability and will be capitalized on in future tool development and integration efforts. The SAVVAS experience also highlights the importance of adequate training to take full advantage of an advanced programming language. Putting

programmers through familiarization courses will not normally result in Ada programs which evidence modern software engineering practices. Instead they will result in poorly written programs that are error-prone, difficult to debug, and costly to maintain. SAVVAS was originally developed by a team with mixed experience and education. The software engineers which conducted the port to the IBM environment could almost guess which programmers wrote which modules. This reemphasized the importance not only of adequate training but also of comprehensive standards and procedures, automated standard checkers, and thorough program walkthroughs and reviews led by experienced programmers.

Gokul Bhaumik

Modernization of Software Quality Assurance

NEED FOR MODERNIZATION OF QUALITY ASSURANCE FUNCTION

The evolution of the modern day programmer-analyst, in a sense, has followed a path similar to the Freemason of the middle ages. During the early days of the computer technology, any understanding of how to build a computerized system set the individual apart and was licensed for personal success. It has been said, however, that often, systems were built much like the Wright Brothers designing airplanes: build the whole thing, somehow, push it off a cliff, and if it flies, fine. If it crashes, start all over again. Of course, some designs were monuments to initiative and individual talent, but - like the builder of old planes, we have not yet fully come to grips with all the variables of system design. The need for results have out-raced the time needed for developing techniques to design, develop, and more importantly assuring the quality software systems or products. The above speech was given by John W. Luke, President, Infonet Division, Computer Science Corporation at the AIIE conference on software, Chicago, Illinois, on July 27, 1977.

NEED FOR QUALITY MANAGEMENT

The customers satisfaction depends not only on functional performance, it also depends on quality characteristics of software products. And it is this quality aspect of software products need to be examined which will pro vide a clear , well-defined framework for quality assurance functions to improve the life-cycle activities providing significant leverage on software quality.

We need to be aware of the thoughts expressed by many quality experts and they are:

- Quality cannot be added on. It means that unlike present day, traditional inspective type of control, it must be engineered from the very beginning of the software development process. The quality function must start at the same time when system conceptualization begins.
- The level of quality built into a program is a function of the quality attributes employed during the development process. Standards, practices, tools, and techniques are needed to define these attributes. If they do not exist, the quality process remains a subjective evaluation.
- Quality therefore, must be managed. It must be planned, it must be organized. It must be directed and it must be regulated or controlled.

The above thought provoking comments, therefore, lead us to the necessary definition of Software Quality Assurance function.

Definition: Software Quality Assurance is a formal, planned approach of actions designed to evaluate the degree of identifiable set of quality attributes present in all software systems and its attendant products.

To support the above definition, the architect of quality evaluation must plan and implement necessary tools, techniques, and methodologies in such a manner that brings to fruition another important advocacy advanced by many experts in the quality disciplines :

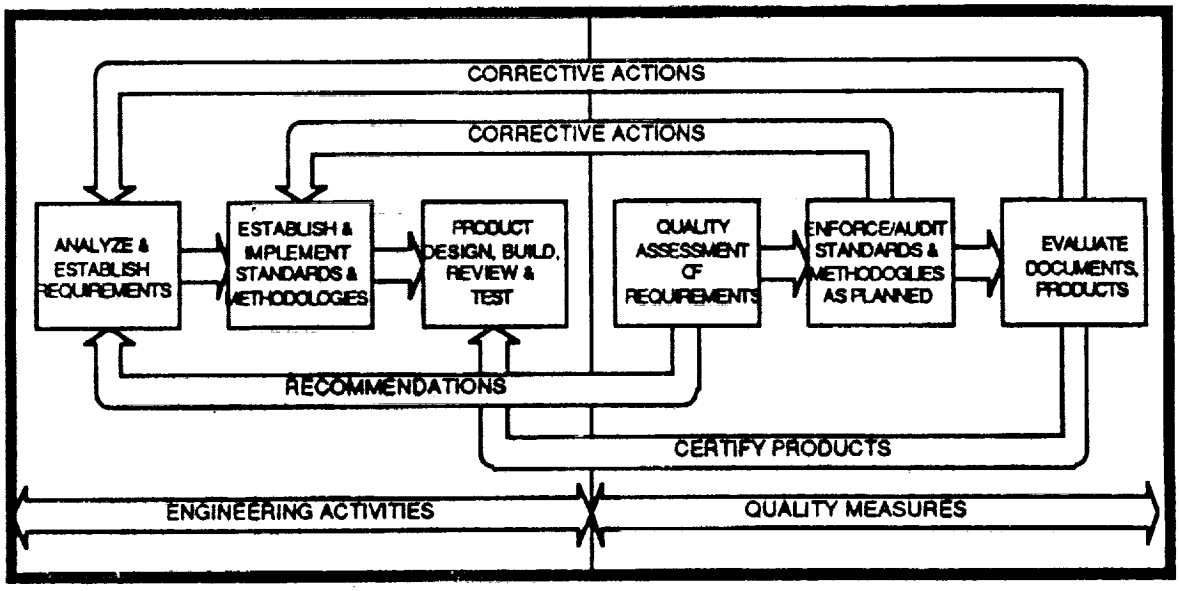
"A strictly orchestrated interdependency between the design and development processes or product and their concurrent verification measures for attributes relative to quality."

QUALITY MANAGEMENT ROLE

The Quality Management Role on any Software project must then be to:

- Monitor
- Regulate
- Evaluate

the Software Development Process and Products and recommend/initiate necessary corrective action(s) as depicted in the following figure.



QUALITY EVALUATION

For the purpose of Quality Evaluation, necessary criteria must be established for both the process and the products as well.

PROCESS EVALUATION CRITERIA

- Activities required by approved project plans are performed.
- Processes are compliant with the approved project plans.
- Tools, Techniques & Methodologies described in the project plans are utilized.
- Processes are adequate to meet the contractual requirements.
- Adequacy of configuration control system
- Adequacy of discrepancy reporting and corrective action system.

PRODUCT EVALUATION CRITERIA

- Compliance with contractual specification requirements
- Adherence to required format and documentation standards
- Technical Adequacy
- Consistency with indicated documents
- Traceability to indicated documents
- Appropriate degree of Quality attributes(factors), namely,

  Correctness, Efficiency, Flexibility,
  Integrity, Interoperability, Maintainability,
  Portability, Reliability, Safety, Reusability,
  Testability, etc

- Adequacy of test cases,and test procedures
- Completeness of testing
- Adequacy of retesting.

SR&QA TOOLS AND METRICS

A significant amount of the work done to evaluate quality is manual, tedious, and subject to human error. Tools are desirable in order to monitor development process, compliance to standards, change control etc. Automation aids should be used extensively to simplify many of these tasks to overcome the complexity and volume of products. Automation aids can be used to correlate and centralize the software requirements. Software design can be directly verified by software tools. A variety of design and code checkers, both static and dynamic should be used for detailed verification of resultant code. Traditional review/audit checklists can certainly be made computer aided. Some other tools are:

- Impact analysis tool
- Requirements traceability matrices
- Test specification tool
- Regression test identification tool

QUANTITATIVE EVALUATION OF SOFTWARE QUALITY UTILIZING METRICS

In order to evaluate quality quantitatively, quality of software must be defined in terms of measurable attributes of software and only then mechanism can be devised to measure it quantitatively.

Light and Fisher have defined software quality as "the composite of all related attributes which describe the degree of excellence of computer software".

General Electric study, sponsored by RADC, has refined the above notion of quality further into identifiable factors which are some conditions or characteristics that contribute to software quality.

Based upon the result of these studies, T. J. McCabe has defined the process of Quality Evaluation as the identification of important factors in a given environment, the specification of these factors and the measurement of the degree of their presence during and after implementation.

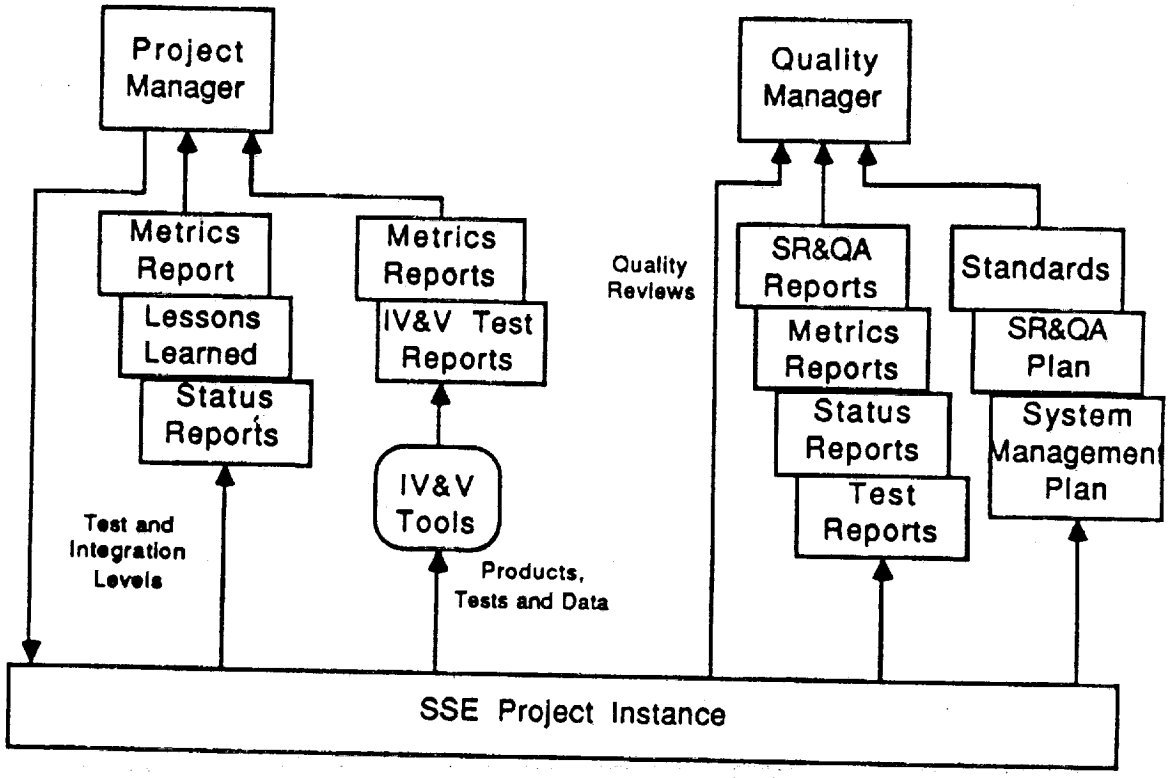
Units of metrics are defined as the ratio of actual occurrences or non-occurrences to the possible number of occurrences of certain software attributes.

Defining the actual metrics that are used to determine the quality of a specific product is beyond the scope of this paper.

WHAT LOCKHEED IS DOING TO AUTOMATE THE QUALITY EVALUATION PROCESS

Through the Space Station Freedom Software Support Environment(SSE) Project, we are called upon to meet a new challenge in orchestrating a quantum leap forward in software development productivity and methodologies.

The SSE architecture provides an important quality technology breakthrough by allowing the implementation of quality evaluation techniques in an automated fashion. This automated support results from the "product test control" features inherent with the SSE framework. The following figure describes the information flow in and out of a SSE System Project instance that supports quality management. The SSE instance not only ensures the application of quality criteria, it also maintains current status data on the progress of development and quality evaluation.



In summary, the SSE architecture, supporting the automated quality evaluation is in the process of bringing to fruition already stated advocacy advanced by the quality experts and that is:

"a strictly orchestrated interdependency between the design and development processes or products and their concurrent verification measures for quality."

Empirical Studies of Design Software: Implications for Software Engineering Environments

Herb Krasner,
Lockheed Software Technology Center

Empircal Studies of Software Design : Implications for Software Engineering Environments

Herb Krasner, Lockheed Software Technology Center

1. INTRODUCTION

The empirical studies team (Herb Krasner, Raymonde Guindon, Diane Walz, Neil Iscoe, Vincent Shen, Barbara Smith, Bill Curtis and Nancy Pennington) of MCC's Design Process Group conducted three studies in 1986-87 in order to gather data on professionals designing software systems in a range of situations. The first study (the Lift Experiment) used thinking aloud protocols in a controlled laboratory setting to study the cognitive processes of individual designers. The second study (the Object Server Project) involved the observation, videotaping and data collection of a design team of a medium-sized development project over several months in order to study team dynamics. The third study (the Field Study) involved interviews with the personnel from 19 large development projects in the MCC shareholders in order to study how the process of design is affected by organizational and project behavior. The focus of this report will be on key observations of design process (at several levels) and their implications for the design of environments.

2. OBSERVATIONS

In our study of individual, experienced designers working on the *lift problem* we observed: the differences in design strategies and solutions, the ways in which many levels of abstraction and detail are worked at the same time, the ways in which designers understand and elaborate requirements through explorations of their mental model of the problem environment, and the discovery-oriented nature of their problem solving. Furthermore we identified the main sources of process breakdown [see Guindon, Krasner and
Curtis, 1987] as: 1) lack of specialized design idioms, 2) lack of knowledge about design process and methods, 3) poor prioritization of issues leading to poor selection of alternative solutions, 4) difficulty in considering stated or inferred constraints during solution formation, 5) difficulty in keeping track of and returning to postponed subproblem solutions, 6) difficulty in keeping track of steps or test cases during evaluative simulation, 7) difficulty in expanding or merging subproblem solutions into the complete system solution, and 8) premature committment to an initial solution skeleton based on a priori criteria. Implications for the design of software tools to support individual professional designers were generated.

In our longitudinal design team study [Walz, Elam, Krasner and Curtis, 1987] we observed the processes of group disagreement about goals, processes, plans, issues and system design. We saw that problems can arise in the accomplishment of a group task when individual team members hold conflicting assumptions, goals, beliefs, etc. which are not surfaced and/or resolved. These conflicts can cause conflicting or incompatible system components. Team members attempting to integrate various individual efforts may find difficulties/incompatibilities due to the differences in these underlying beliefs. The process of design by a team is an information pooling task and therefore difficulties in communication can be expected. The identification and characterization of design "inflection points" can lead to more effective management of the divergence/convergence process in team design. Implications for software environments to support a high performance design team were developed.

In our field study of 19 large software projects, we identified the key problem areas spanning the boundaries between project, organization and external settings. We identified problems in: the acquisition and dissemination of sufficient application knowledge, the effect of requirements change and uncertainty, the artificial barriers to software technology transfer, the dynamics of design evolution and the special problems of government contract developments. We also identified and described project level phenominae related to multi-group interaction as [Krasner, Curtis and Iscoe, 1987]: 1) the typical communications breakdowns in large programming projects, 2) the cultural and environmental differences that create barriers to effective intergroup communications, and 3) the boundary spanning activities that coordinate five crucial topical networks of design information. Four types of communication breakdowns observed on the projects were characterized. Implications for software environments to support large projects of remotely distributed teams were developed.

3. CONCLUSIONS

Across these 3 studies we observed how some breakdowns occur, how some get solved, some get amplified, and how some new types occur as you go from individual to team to large projects and organizations. The mechanisms underlying the breakdowns at the individual level were the lack of knowledge about some important aspect of the design or limitations in human information processing and memory capacity. At the team and organizational levels these mechanisms were still important precursors of many

of the breakdowns observed, however, these mechanisms were augmented by interpersonal and organizational processes to create the breakdowns in a multiperson, multigroup design effort. We have identified interpersonal mechanisms providing synthesis and integration that allow teams to compensate for individual limitations. These are the communication mechanisms that provide for the coordination of mental models of the design and its process across a project staff. The processes of design integration and synthesis cannot be effectively translated into a software system unless cognitive coordination processes are effective.

4. The Lockheed STC Effort

The Lockheed STC Software Process Management Group is currently exploring process models of design that support at least the following components: the decomposition of system requirements/designs, the synthesis of relevant design idioms, the intelligent management of project resources, the coordination of models of the design and its process across project staff, the constraint-based exploration of requirments under changing/negotiated conditions and the capture/reuse of historical design rationale.

Tool Interoperability in SSE OI 2.0

C. L. Carmody
and
C. T. Shotton

Title: Tool Interoperability in SSE OI 2.0

Authors: C.L.Carmody and C.T.Shotton

Date: October, 1988

Abstract: This paper presents a review of the concept and implementation of tool interoperability in the Space Station Software Support Environment (SSE) OI 2.0. By first providing a description of SSE, the paper describes the problem at hand, that is; the nature of the SSE that gives rise to the requirement for interoperability - between SSE workstations and hence, between the tools which reside on the workstations. Specifically, word processor and graphic tool interoperability are discussed.

The concept for interoperability that is implemented in OI 2.0 is described, as is an overview of the implementation strategy. Some of the significant challenges that the development team had to overcome to bring about interoperability are described, perhaps as a checklist, or warning, to others who would bring about tool interoperability. Lastly, plans to expand tool interoperability to a third class of tools in OI 3.0 are described.

C-2.

Tool Interoperability in SSE OI 2.0

Background

The SSE System is an integrated system consisting of computer hardware, communication networks, SSE, and all other elements that support the life-cycle management of all Space Station Freedom Program (SSFP) operational software. The Space Freedom Station Software Support Environment (SSE) is an evolving collection of software, procedures, standards, hardware specifications, documentation, policy, and training materials which, when implemented in hardware and a computer network(s), provides the environment for the life-cycle management of SSFP software, including itself. The SSE System will provide a common environment for software support to the SSFP in geographically distributed and networked computer facilities, typically a host processor with attached user workstations. The near-term evolution of the SSE System, in operational increments one through four (OI 1.0 through OI 4.0), involves replacing the COTS from the Interim and Initial Systems (OI 1.0 and OI 2.0) with non-proprietary, SSE developed software which is more tightly integrated into an environment.

On October 10, 1988, the second operational release of the SSE System, OI 2.0, was delivered to NASA for the continued development of the SSE itself, and for use by SSFP developers, the Work Package Contractors, prior to the delivery and installation of their Software Production Facilities (SPFs). Delivery of the SPFs will constitute the next major operational increment of the SSE; OI 3.0. As of OI 2.0, the SSE System contains two host architecture types (VAX 8820 and IBM 3090), and three workstation types (Macintosh II, IBM P/S 2, and Apollo). The bulk of the functionality of OI 2.0 is provided by commercial off-the-shelf software (COTS), on both hosts and workstations, which supply initial SSE capabilities in process management support, software production support, document development and production support, office automation support and project management support. A major requirement of the SSE System is to provide equivalent functionality on each of the three workstation types, for that functionality allocated to the workstations, and on each host architecture, for that functionality allocated to the hosts.

Typically, a Space Station Freedom software development project will consist of several developers and testers working as a team. Each team member will do a major portion of his work on his own workstation (any one of the supported workstation types), and then place that work under configuration control on the host for integration with the work of others. In order for that integration to take place, whether it be the integration of code, of design products, or of document sections, there must be a concept and method for the interoperability of functionally equivalent tools.

In OI 2.0, a portion of the functionality that is allocated to the workstations is the preparation of document sections, both text and graphics. To satisfy that requirement, the workstations are equipped with the following word processing and graphics packages;

- Microsoft Word and MacDraw on the Macintosh II
- Microsoft Word and Gem Draw Plus on the IBM P/S 2
- Interleaf 3.0 on the Apollo (for both text and graphics)

Each word processing and graphic package outputs a different format; even Microsoft Word on the Macintosh II outputs a different format from Microsoft Word on the IBM P/S 2. In order to support the common text and graphic functionality (with distinct output formats), OI 2.0 contains a set of transformation procedures which will transform tool-specific format to a common interoperability format, and from that format back to tool-specific format. Planning Research Corporation (PRC), subcontractor on the SSE project, has developed a set of word-processing transformation procedures, to support a Text Interoperability Format (TIF), and a set of graphic transformation procedures, to support a Graphic Interoperability Format (GIF).

The Concept of Tool Interoperability

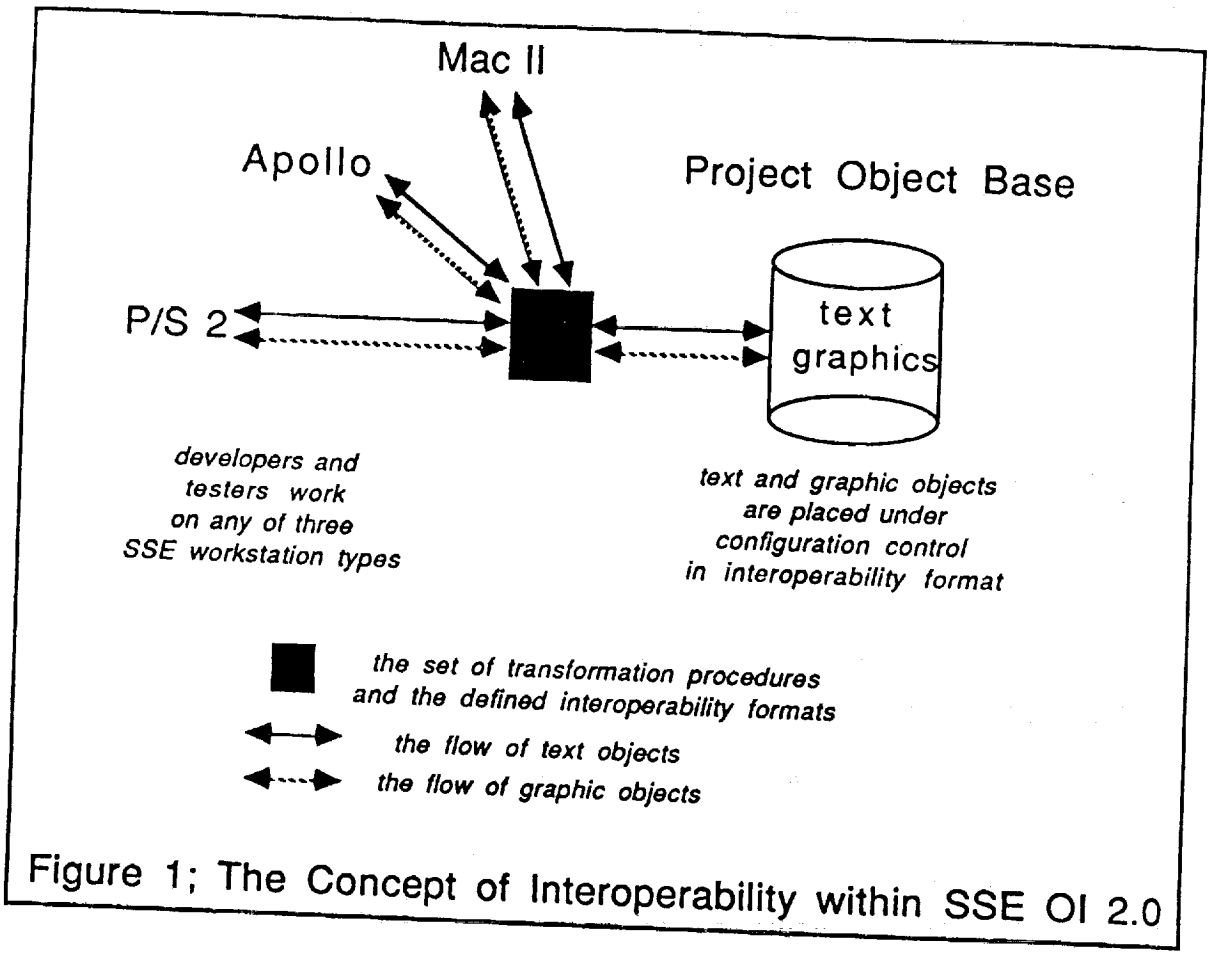


Figure 1; The Concept of Interoperability within SSE OI 2.0

As is indicated in Figure 1, The Concept of Interoperability within SSE OI 2.0, SSE users do a major portion of their work with tools that reside at the workstation. In this context (which is shown to be the development of text and graphics objects for later integration into a document), developers and testers are shown as the major SSE users; developers produce text and graphic objects using the baselined tools available on their workstations. When a developer is through with a given text or graphic object, he transforms it into the appropriate interoperability format, with one of a set of transformation procedures. The object, in interoperability format, is placed under configuration control on the host with the other objects for the given document. That object is now available for test, and testers may run both host-based tests and workstation-based tests. For example, a test of a graphic object may consist of bringing it down to the tester's workstation and making sure all words in the

figure are spelling correctly, and the figure itself conforms to project graphic standards. Using the transformation procedures, the tester does not need to have the same type of workstation as the developer; he doesn't even need to know on what kind of workstation the objects were originally developed. And, if an object needs rework, a different developer with a different type of workstation can do the rework. Ultimately, when all planned tests are passed, the objects are integrated into the final product on the host; in this case, a document. [SSE OI 2.0 also contains transformation procedures which will transform text and graphic objects from interoperability format into a merged document using the VAX host-based document processing tool, Scribe].

As shown in Figure 2, Text Interoperability Procedures, there are six text transformation procedures; three transform tool-specific output into text interoperability format, three transform from text interoperability format into a form acceptable by the SSE baselined wordprocessors.

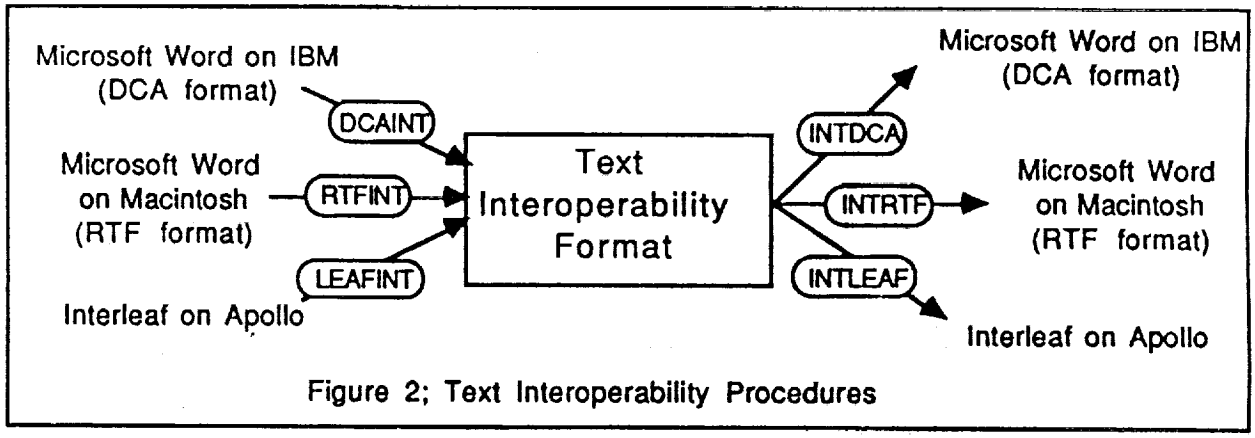


Figure 2; Text Interoperability Procedures

As shown in Figure 3, Graphic Interoperability Procedures, there are six graphic transformation procedures; three transform tool-specific output into graphic interoperability format, three transform from graphic interoperability format into a form acceptable by the SSE baselined graphic packages.

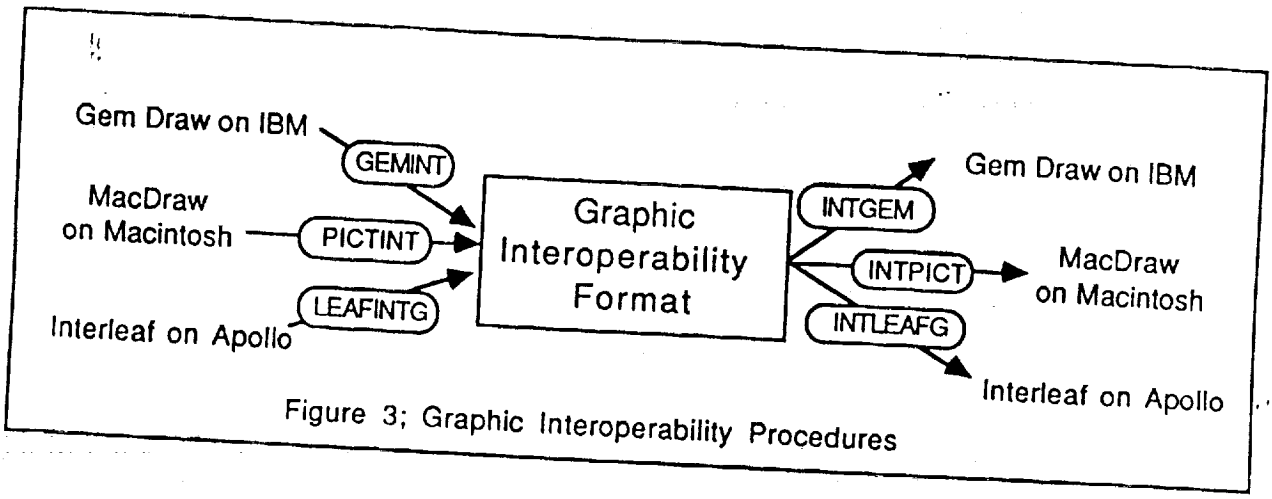


Figure 3; Graphic Interoperability Procedures

Implementation Overview

Design Drivers

It is possible to separate those qualities of software design that are featured in the development of SSE tool interoperability into two sets; those design drivers of SSE which are supported by the concept and implementation of the transformation procedures, and those design drivers which characterize the implementation of the transformation procedures.

Support to SSE Design Drivers

The following list is an example of those design drivers (quoted from the SSE System Concept Document) mandated for the SSE System which the transformation procedures support directly.

Commonality; "from the users' perspective, each host/workstation combination will have exactly one set of tools available for SSP software support.. These tools will support interoperability between host/workstation combinations"

Expandability; "the architecture of the SSE System will be amenable to addition of new capabilities or external system interfaces"

Technological Transparency; "every aspect of the SSE design will enable change brought about by advancing hardware and software technologies to be of minimal impact"

Data Type Transparency; "the SSE will shield the user from explicit, required knowledge of the data types addressed for work in levels where such is not appropriate"

Integration; "the SSE will be a tightly integrated whole, centered around a policy of software life-cycle management. Tight integration is comprised of maximizing information transfer between SSE functional capabilities with a simultaneous minimization of user intervention."

Use of Ada; "the SSE supports the use of Ada for the development of all SSP operational software, including itself"

Vendor Independence; "the SSE design will be highly portable so as to avoid dependence on any particular vendor, computer hardware system, workstation, data base management system, operating system, network, or application program. The interoperability features provided by the SSE System remove much of the dependency on vendor-specific hardware and software, minimizing the risk in that area"

Transformation Procedures Design Drivers

Maintainability; Once a vendor puts out a word-processing or graphic package, there is no guarantee that when an update is released, the vendor will have maintained the old format. The lack of control over vendor output has forced the design of the transformation procedures into a highly data-driven design, to reduce impact on the software when the input file format changes.

Reusability; Each transformation procedure is structured in a similar manner, using the same skeleton Ada package specifications, and calling the same service procedures. When a new transformation procedure is required, it is not developed from scratch, but rather assembled from existing components and tailored for its new use.

Portability; The SSE requirement to support equivalent functionality on each host architecture has driven the transformation procedures to identify and isolate machine dependencies.

Reliability and Usability; The transformation procedures are used several times daily by nearly all current SSE developers and testers; the PRC developers have made reliability and usability the highest of design priorities, the PRC testers have exercised creativity in the design of test cases which attempt every unforeseen way of making the software fail.

Implementation Details

Each transformation procedure is either a LALR(1) parser or a context sensitive, recursive descent parser, which scans the input format and, based on parse tables containing the syntax for the input format and action rules transforming input constructs into output constructs, creates a file in the desired output format. (An LALR(1) parser Looks Ahead from Left to Right 1 symbol) Rather than writing each parser from scratch, a public domain parser generator (developed for NASA Langley) was used to create the skeleton for each transformation procedure and the parse tables to be used by the transformation. The parser generator used was PARGEN, part of the MYSTRO suite of compiler-compiler tools. PARGEN takes as its input a production grammar which describes the transformations from one format to another in unambiguous terms. From this grammar, the LALR(1) parse tables are built, along with procedures to do the lexical analysis, or scanning, and the syntax-semantics synthesis.

As SSE is required to be completely in Ada, PARGEN was modified to generate Ada, and all host-resident transformation procedures are written in Ada. Due to vendor restrictions on access to their proprietary formats and routines accessing these formats, two transformation procedures, GEMINT was developed on the IBM P/S 2, and PICTINT was developed on the Macintosh II; both in C.

Interoperability Formats

Key to the design and implementation of the transformation procedures is the definition of the interoperability formats. Each format (text and graphics) attempts to define in a tool-independent format that functionality which is both common to all baselined workstation tools and required by the majority of the users. Both text and graphic interoperability are defined in Backus-Naur Form

(BNF), however the graphics interoperability format provides a more extensive object-oriented language, by virtue of the nature of graphics, and standard manipulations on graphic objects. For example, drawing a rectangle filled with your choice of patterns is a fairly standard capability for a graphics package.

Significant Challenges

Any integrated system (office automation, software development, etc) which provides a set of tools with equivalent functionality should provide some means for the interoperability of those tools and their outputs. It is worthwhile to review some of the challenges that must be taken into consideration.

Parsing techniques; the development of the SSE transformation procedures required an in-depth understanding of parsing techniques, at least equivalent to the information provided in a typical compiler design course. Properly managed, the entire team need not have this degree of expertise in compiler construction, but should have mastery of the fundamentals of the theories involved.

Determination of common functionality; Probably one of the most controversial aspects to achieving interoperability is the definition of the interoperability format. During this exercise, the common subset of the functionality provided by the baselined tools is defined formally as a grammar. The main problem is dealing with users who would prefer the interoperability format to provide a **superset** of all the functionality provided by the tools.

Control of vendors; The SSE Program has no control over the schedule of update releases from vendors, or the content of update releases. This can create serious schedule problems, as the transformation developers attempt to keep up with several package updates at once, especially as an older package may cease to be available before the corresponding transformation procedure or format has been updated for the updated package. A second severe problem is the access to the vendor formats. Not all vendors subscribe to 'open systems'; as a result some transformation procedures must reside on the system on which the vendors provide access routines to their proprietary formats. The ideal

solution to the problem of controlling vendors is to publish the interoperability formats, for use by each vendor to provide their own transformation procedures.

Plans for OI 3.0

Transformation Procedure Maintenance

No extensive enhancements in the word processing and graphic transformation procedures are planned for SSE OI 3.0. Typical maintenance will consist of making any changes necessitated by vendor updates, modify any undesirable features, and analyze the widening of the interoperability formats.

Design Tool Interoperability

A significant enhancement to SSE tool interoperability is planned for OI 3.0 through the introduction of a new class of transformation procedures. PRC plans to define an interoperability format and implement the transformation procedures to bring about interoperability between the SSE baselined CASE tools; Cadre's TEAMWORK on the Apollo, Excelerator on the IBM P/S 2, and Iconix PowerTools on the Macintosh. Again, the most significant challenge will be to determine the common required functionality and a means for representing that functionality in a production grammar. The transformation procedures themselves will be built upon the legacy of the word processing and graphic transformation procedures.

omit

Developing Software Engineering for Competitive Advantage- Industry and Federal Government

Session Co-Chair: **John R. Garman**

Session Co-Chair: **Richard Kessinger**

Speakers

Dana L. Hall

Jack Munson

Howard L. Yudkin

Om17

The Role of Software Engineering in the Space Station Program

Dana L. Hall

(NOTES)

UNISYS' EXPERIENCE IN SOFTWARE QUALITY AND PRODUCTIVITY MANAGEMENT OF AN EXISTING SYSTEM

By John B. Munson

Vice President and General Manager

Unisys Houston Operations

A summary of Quality Improvement techniques, implementation and results in the maintenance, management and modification of large software systems for the Space Shuttle Program's ground-based systems.

UNISYS' EXPERIENCE IN SOFTWARE QUALITY AND PRODUCTIVITY MANAGEMENT

For more than a quarter-century, the Johnson Space Center (JSC) in Houston, Texas developed and operated large computer systems to support manned spaceflight. Until three years ago, JSC used 11 contractors to develop, operate and maintain these systems. Integration and management were performed by government personnel.

In 1985, JSC decided to separate system development from maintenance and operations. Unisys Houston Operations, as part of the Rockwell STSOC team, was selected to manage, modify and maintain the Space Shuttle Program's ground-based, flight-support systems.

We are responsible for the Shuttle program's more than 14 million lines of executable code, which was written in 15 different programming languages. The code operates on equipment made by eight manufacturers, and runs on 173 computers located in 13 separate JSC facilities.

The software spans the entire life cycle of every mission, including flight planning, astronaut and flight controller training, flight simulation, flight software verification and flight support. Our Shuttle program software responsibilities encompass JSC's:

- o Flight and Mission Planning,
- o Flight Software Verification
- o Flight Simulation and Training, and
- o Mission Control Center Operations and Communications.

In addition, we support the integration and testing of all associated software and its flight-to-flight reconfiguration. Our software includes the code that:

- o monitors the Shuttle's launch, orbit and landing;
- o maintains communications between JSC and other NASA centers and flight support facilities, as well as with the Shuttle itself;
- o tracks the Orbiter's progress, performance and physical state; and
- o calculates the amounts of oxygen, water, fuel, electricity and other critical onboard resources for every flight.

We must complete our work accurately the first time and every time in order to attain the quality essential to

ensure safe Shuttle operations, and to meet cost and schedule requirements. Any error of a magnitude sufficient to affect a single system delivery will be propagated into downstream systems and cause substantial cost and schedule impacts. More importantly, any error in systems we support could endanger astronauts, the Orbiter and Shuttle missions.

It is critical, of course, that we achieve the highest degree of quality in every one of our tasks. We constantly strive to make outstanding performance the priority goal of every employee at every level.

At Unisys Houston Operations, this means we must achieve excellence in all requirements of very large software systems, including their design, code generation, testing and verification, and system release management. We must meet the same standards while retrofitting software generated by third parties, and in our support activities for flight simulation and training as well as for actual missions themselves.

Attaining quality and productivity in all these functions is a very large task in itself. We have learned that an organization must strive to prevent defects in every process and every job to accomplish this. We believe that procedures similar to ones we've implemented could be used by any company to reduce the size of the quality task. By concentrating on a few simple quality concepts and capitalizing on commonalities in their application, the goal can be achieved.

It was clear to us at the outset that we could benefit from the knowledge of quality improvement experts. Several of our managers attended the Phillip Crosby Quality College in Orlando, Fla. We understood that the courses gave generic training in quality concepts and that it would be up to us to apply these to our business. We began our Quality Engineering Program by tailoring these guidelines to our specific needs.

When we assumed responsibility for the Shuttle's ground-based software systems, we immediately instituted a Quality Improvement Process within our organization. We established a Quality Policy (see attachment) setting forth our management commitment to Quality Improvement. The policy includes Quality Training for every employee, the documentation and measurement of all software processes, and an infrastructure consisting of quality improvement teams and a formal, corrective action review process.

Our first step to implement the process was to remove ambiguity from the concept of quality by giving it a specific, easily understood definition. We define Quality as conformance to requirements. We understand and practice that preventing nonconformance while building a product is more effective than finding and eliminating defects through appraisals after the product is built. We measure the effectiveness of our software processes in real terms, rather than with indices, and we eliminate permanently recurring nonconformances by correcting these processes. We

strive for 100% employee participation in all aspects of the Quality Improvement Process.

How we achieve quality

The emphasis on quality must flow from and to management. Employees have deep motivations to fulfill management expectations. When management emphasizes quality improvement and provides the direction, resources and goals to attain it, management commitment is visibly demonstrated. It is essential that employees understand the sincerity of our efforts. I personally participate in the initial sessions and graduations of our Quality Education classes.

Through this Quality Education for all employees, we provide a hands-on training environment which assures that a consistent set of quality principles and concepts will develop an organizational mindset for quality improvement. An infrastructure of Quality Improvement Teams, Steering Committees, and Corrective Action Teams raises the level of quality awareness and the effectiveness of the quality improvement process. We also document all software processes, standards and procedures we employ; measure process-defect yields and their rates of occurrence; and conduct formal, corrective action reviews.

We have instituted the Oregon Objectives Matrix as an aid to record and report our measurements of quality and productivity factors. The Oregon Objectives Matrix, developed by the University of Oregon, is used extensively in the Unisys Defense Systems Group, of which we are a division. The matrix provides a graphic method to track and report several measurements on just one page. This enables managements to quickly see trends in the measurements, which are tracked monthly.

Quality improvement is an integral part of everything we do. There are two very important aspects of our Quality Improvement Process - the Quality Engineering Process and the Quality Assurance Process. These are two distinctly different processes performed independently of each other.

It is the responsibility of the engineering departments to define, document and implement processes assuring that we produce quality products. It is the responsibility of the Quality Assurance Department to read, understand and supervise implementation of the defined processes. We have the capability to file nonconformance reports against the engineering process when product defects are discovered, and within the engineering departments when the correct processes are not followed.

In our software work, the engineering process begins with the detection of nonconformances. We initiate a thorough investigation to determine why the defect occurred, then we correct it as soon as possible. In addition, we

evaluate each new defect in the context of prior ones to uncover commonalities or clues to root causes. We are determined that no defect will recur. Our corrective action process review then locates the defect's cause and recommends action to correct the faulty process. This correction of the process permanently eliminates that class of defects.

How we achieve high reliability

We use the quality principles we've learned to maintain the high reliability of our software systems. Through our experience, we've found what we believe are fundamental principles to ensure the dependability of large software systems. First, such systems must be allowed to mature so that indispensable knowledge of their operations can be acquired.

Also, no unnecessary changes should be made. In fact, large areas of code may never need to be changed. We identify these and set them aside to protect them from being changed accidentally. We then concentrate on the small area that remains, where most defects occur.

In addition, we:

- have identical software functions independently developed to be able to compare results,
- control and manage the number and size of changes,
- use table-driven software to minimize software changes,
- incorporate redundancy whenever possible,
- rigorously enforce software engineering standards,
- Submit our software releases to independent tests and quality assurance audits,
- conduct extensive regression testing of unchanged software plus structured hierarchical testing of new software changes,
- exercise the software extensively prior to flight, and
- establish failure/recovery procedures.

Results to date

Our efforts in quality and reliability improvement have yielded outstanding results, particularly in terms of productivity. Some of our major accomplishments are listed below.

- Reduced our independent verification and testing schedule by six days, saving 440 labor-hours for each major software release
- Reduced the number of database trajectory discrepancies attributable to database reconfiguration from 80% to 5%
- Reduced the mean-time to close discrepancies from 40 days to 2 days
- Achieved a 10% reduction in the backlog of change requests
- Achieved a 40% reduction in the backlog of discrepancy reports
- Reduced our mean-time to evalute new changes from 16 days to 9 days

Future plans

We will continue our Quality Improvement Process for the length of our Space program software responsibilities. We will continue to evaluate every software process that is instituted. In addition, we will:

- improve the quality characteristics of all supported software bases,
- establish a requirements engineering program and a test engineering program,
- consolidate and standardize wherever possible,
- refine and evolve our metrics for measurement processes, and
- automate software support and operations whenever possible.

The Quality Improvement Process we employ was adopted virtually in toto from the renowned Crosby method. The process is now used throughout our entire corporation, but we at Unisys Houston Operations are proud we were one of the first divisions to implement the process and are a leader in its application.

Quality Policy

We shall strive for excellence in all endeavors

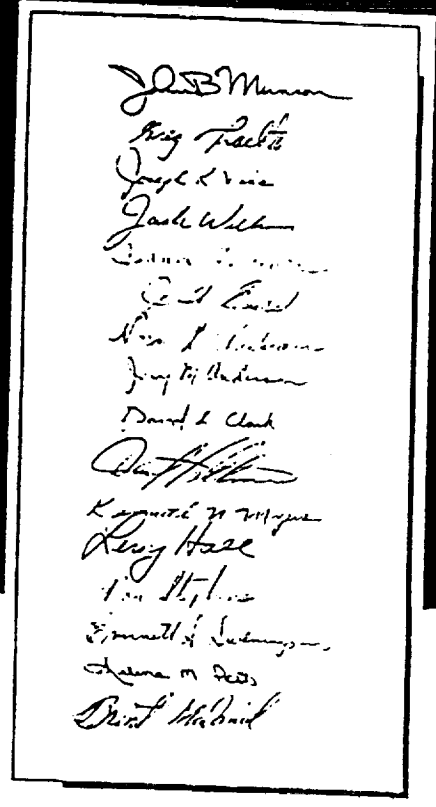
We shall set our goals to deliver error free products and services, on time.

We shall understand and conform to the requirements.

We shall understand the software processes and standards associated with our jobs.

We shall measure our performance in terms of satisfying the requirements.

We shall analyze failures and take corrective action to prevent their reoccurrence.



UNISYS
Houston Operations

Next Generation

omit

Howard Yudkin

The synthesis process for incorporating reuse and prototyping ideas into large software system development suggest how the acquisition process might be changed to support the new development process.

(NOTES)

Software Engineering as an Engineering Discipline

Panel Chair and Moderator: **Glenn B. Freedman**

Panel

John Brackett

Ed V. Beard

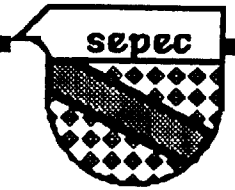
Robert B. MacDonald

Norman Gibbs

N94-71147

Software Engineering as an Engineering Discipline

Glenn B. Freedman



SOFTWARE ENGINEERING AS AN ENGINEERING DISCIPLINE

A Panel Presentation for

RICIS Symposium '88

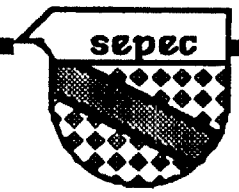
Chaired by

Dr. Glenn B. Freedman, Director

Software Engineering Professional Education Center

University of Houston - Clear Lake

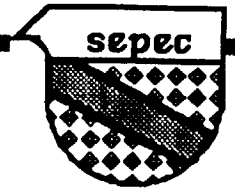
November 10, 1988



Purpose of the Panel

To explore the emerging field of software engineering from a variety of perspectives:

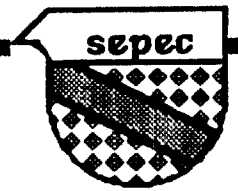
- √ University Programs
- √ IndustryTraining and Definition
- √ Government Development
- √ Technology Transfer



INTRODUCTION

The panel will address the issues of what a current definition of software engineering might be -- and what that definition could become.

The panel will address the issues the distinctions among software engineering, computer science, and computer hardware engineering as they relate to the challenges of large, complex systems.



INTRODUCTION

KEY POINTS

Software life cycle issues are increasingly important for all organizations.

Software systems that are large, complex, distributed, non-stop and have indefinite life spans must be engineered for change -- education and training systems must also be responsive to the environment.

There are two kinds of software engineering costs: for acquisition and for ownership. We can pay now or we can pay later.

Higher productivity can result from use of modern software engineering practices, software reuse, use of commercial products, education and training, automated support, and good management.

Meeting the Challenge of Industrial Software Development: the Boston University Graduate Program in Software Systems Engineering

Panel Presentation
"Software Engineering as an Engineering Discipline"
NASA/Johnson Space Center and RICIS Symposium

Joh W. Brackett, Professor
Department of Electrical, Computer and Systems Engineering
Boston University

Key Aspects of the Boston University Software Engineering Program

- Integral part of the College of Engineering
- Embedded system orientation
  - required courses on hardware
  - Ada as principal language
- Developed to meet industrial needs in cooperation with Corporate Advisory Board
  - ATT Bell Laboratories
  - Digital Equipment Corporation
  - GTE Laboratories
  - Hewlett Packard
  - MITRE
  - Raytheon

Software Engineering Skills Lacking in Most CS Graduates

- Hardware/software integration
- Requirements analysis
- Test planning
- Configuration management
- Design experience
- Project planning and scheduling

Goals of the Boston University MS in Software Systems Engineering

To provide graduates with

- theoretical foundations needed to assess and new hardware and software advances
- understanding of, and experience with, tools and methods for embedded system software development
- managerial skills to plan, organize and lead a software development project
- experience as a team member

4 Strategic Decisions

- To build a wide base of industrial support in order to ensure long-term financial viability
  - Corporate Associates Program
  - Corporate Advisory Board
- To use interactive television to reach part-time students
- To emphasize software which is integral to a hardware product
- To develop the program in the College of Engineering and to seek faculty support for it as an engineering discipline

Why Software Engineering in the College of Engineering?

- WHY NOT??
  - relationship to systems engineering and computer engineering due to embedded system emphasis
  - little interest in an industrially-oriented program in BU Computer Science department
  - existing faculty have science and engineering degrees
- Eliminates problems with an "engineering" degree that is not in the College of Enginering
- Engineering is the natural home for Software Engineering in the long term
  - computer science is to software engineering as:
    chemistry is to chemical engineering
    physics is to electrical engineering

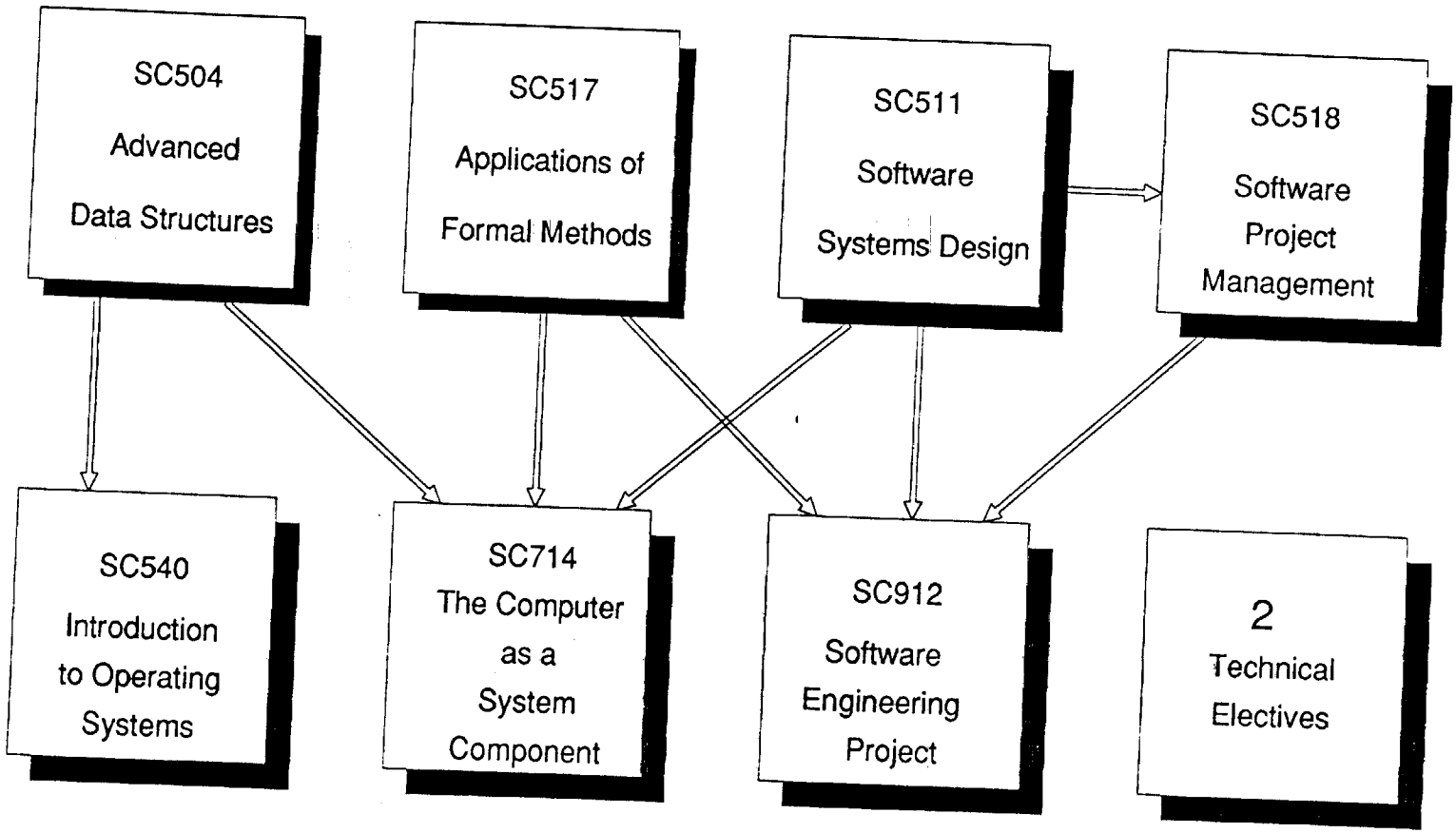
Current Program Status

- Enrollment as of 10/88:
  - 5 full-time students
  - 20 part-time degree candidates
  - 50 special students
- 4 faculty teaching required courses
- 4 courses a semester on interactive TV
- Over $1.3 million in support raised in 18 months from
  - Digital Equipment Corporation
  - Hewlett-Packard
  - MITRE
  - Raytheon

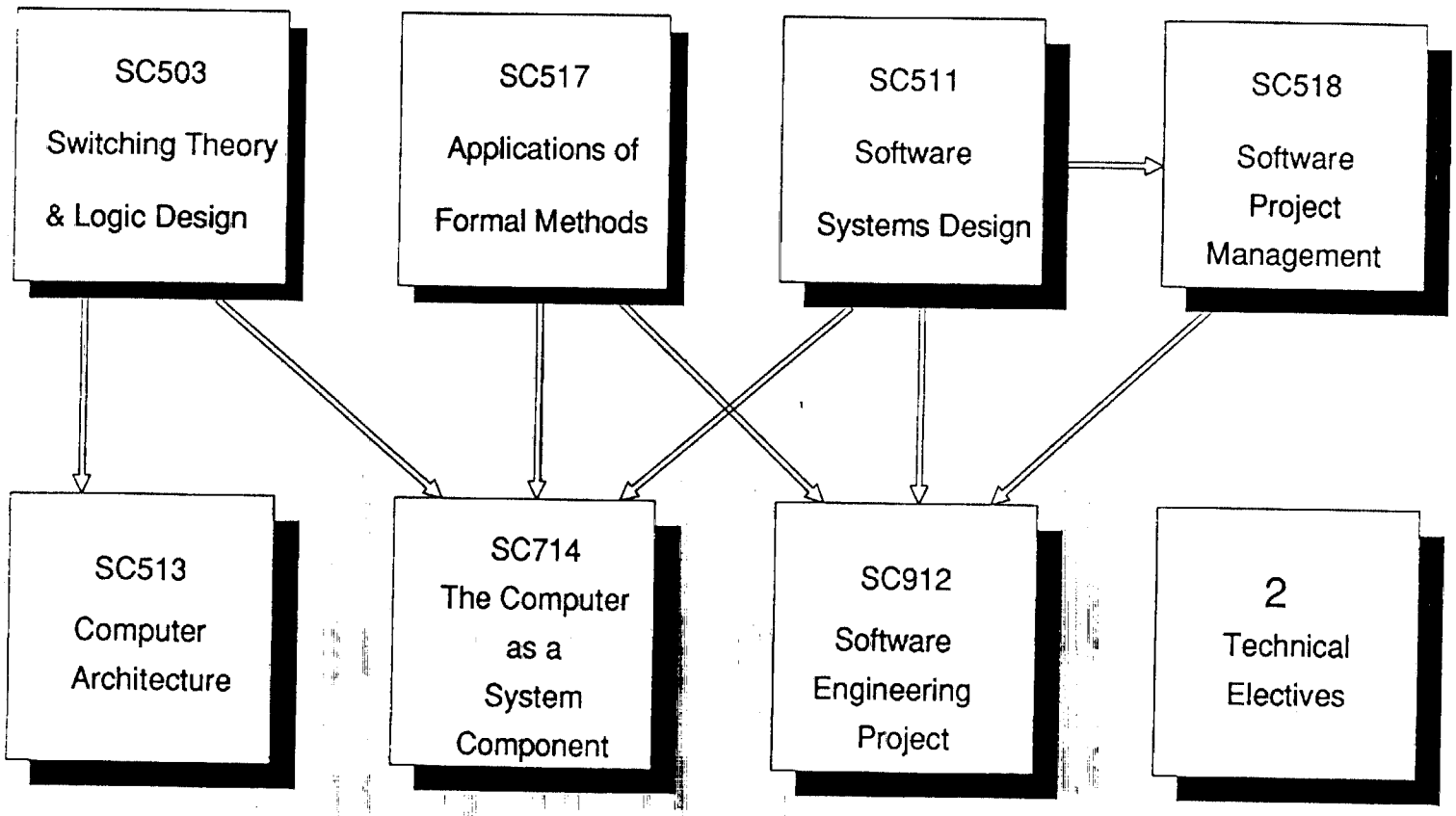
Embedded Systems Laboratory

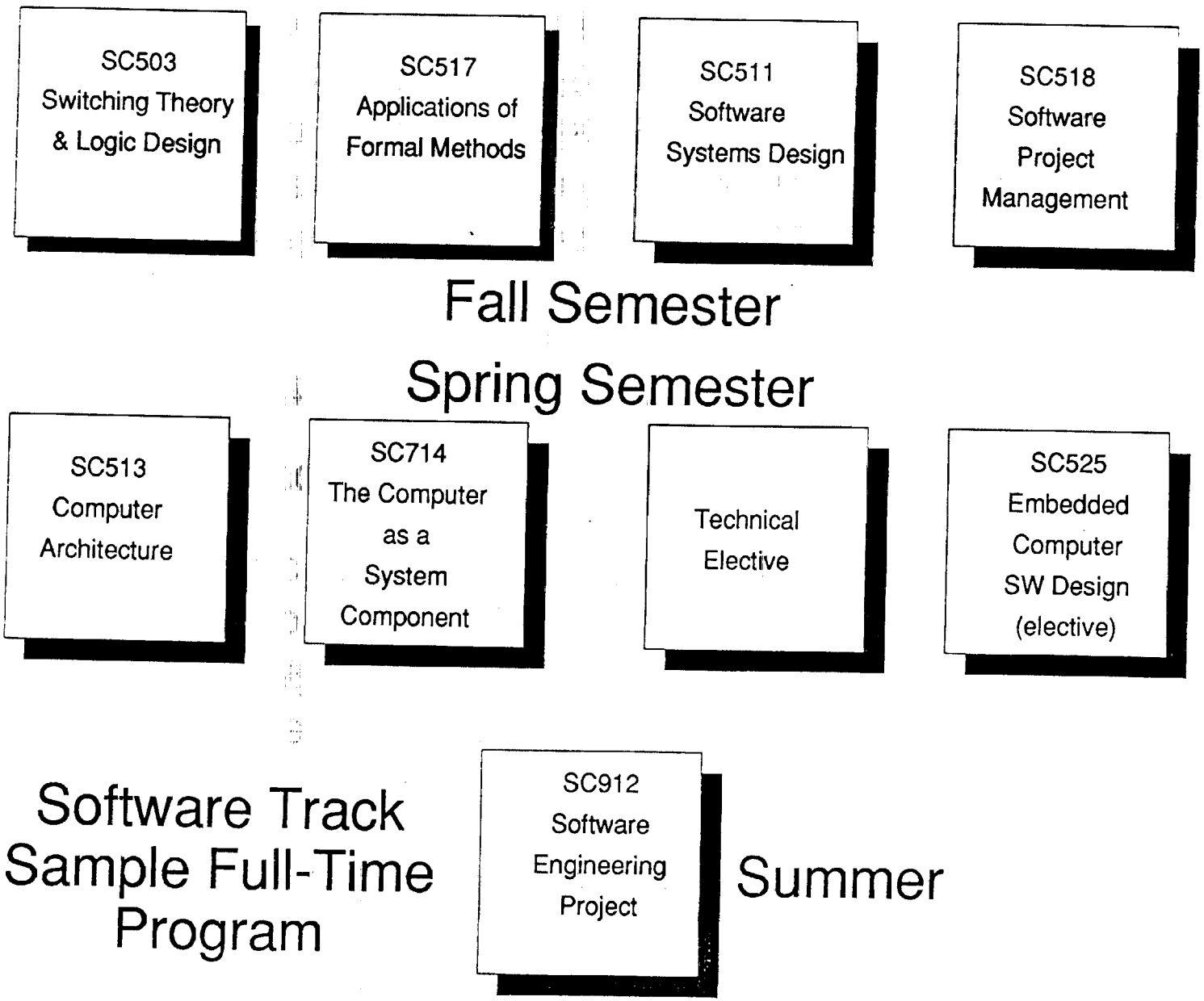
- State-of-the-art showcase supporting modern software engineering techniques for building embedded systems
- Network with 20 VAXStations and target machine equipment
- VAX Ada environment and "industrial grade" support tools
- Central compute and file servers with remote access
- Over 50 mips of computing power
- Graphically-oriented analysis and design tools including Teamwork and Statemate
- Made possible by grants from Digital Equipment Corporation, Cadre Technologies and i-Logix

Prerequisite Structure
Hardware Background



Prerequisite Structure
Software Background




SC503
Switching Theory
& Logic Design
SC517
Applications of
Formal Methods
SC511
Software
Systems Design
SC518
Software
Project
Management
Fall Semester
Spring Semester
SC513
Computer
Architecture
SC714
The Computer
as a
System
Component
Technical
Elective
SC525
Embedded
Computer
SW Design
(elective)
Software Track
Sample Full-Time
Program
SC912
Software
Engineering
Project
Summer

Implications of Remote Delivery by Satellite Television

- Will only be available to corporate sponsors of the educational program and selected government agencies
  - 8-10 major corporations
  - 2-3 agencies supporting Ph.D. research
- 7 out of 9 courses can be taken on interactive television
- Courses with team projects (511 and 518) require corporate support for 2 faculty visits and easy access to software
- One semester required on campus for the degree
  - "Computer as a System Component" requires the Embedded Systems Laboratory and close faculty supervision
  - "Software Engineering Project" is done in 4-6 person teams with frequent faculty interaction

Objectives for the Next 3 Years

- Grow enrollment to steady-state of 60 FTE students
  - 25 full-time on campus
  - 70 part-time Boston area (on campus and local TV)
  - 70 part-time outside Boston area (satellite TV)
- Develop additional laboratories
  - human-interface laboratory
  - networking/distributed processing laboratory
- Build software engineering faculty to 6
- Develop Ph.D. program and software engineering research
- Be recognized as the leading graduate program concentrating on software engineering for embedded/real-time systems

Software Engineering as an Engineering Discipline

Edward V. Berard
EVB Software Engineering, Inc.

5320 Spectrum Drive
Frederick, Maryland 21701
(301) 695-6960

Presented at
Research Institute for
Computing and Information
Systems

University of Houston,
Clear Lake
November 10, 1988

Software Engineering

Software Engineering as an Engineering Discipline

Presented at
Research Institute for
Computing and Information
Systems

University of Houston,
Clear Lake
November 10, 1988

Edward V. Berard
EVB Software Engineering, Inc.

5320 Spectrum Drive
Frederick, Maryland 21701
(301) 695-6960

What Is Software Engineering?

- Early Use of the Term
- 1968 NATO Conference
- Barry Boehm's Definition
- Four Requirements for Software Engineering
- Additional Criteria for Software Engineering

Early Use of the Term

- "Coder" - Early 1950s
- "Programmer" - Mid 1950s
- "Programmer/Analyst" - 1960s
- "Software Engineer" - 1980s (1963)

1968 NATO Conference on Software Engineering

- Is software engineering different from computer science ?
- If so, what does software engineering entail ?

Barry Boehm's 1976 Definition of Software Engineering

Software engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation.

Four Major Requirements for Software Engineering

1. Computer Science
2. Mathematics
3. Engineering Disciplines
4. Excellent Communication Skills

Computer Science

- **Programming Topics**: Algorithms, Programming Languages, Programming Style, Debugging and Verification, Applications
- **Software Organization**: Computer Structure and Organization, Data Representation, Symbolic Coding and Assembly Systems, Addressing Techniques, Macros, Program Segmentation and Linkage, Linkers and Loaders, Systems and Utility Programs
- **Hardware Organization**: Computer Systems Organization, Logic Design , Data Representation and Transfer, Digital Arithmetic, Digital Storage and Accessing, Control and I/O, Reliability
- **Data Structures and File Processing**: Data Structures, Sorting and Searching, Trees, File Terminology, Sequential Access, Random Access, File I/O

Mathematics

- Integral Calculus
- Special Functions
- Differential Equations
- Linear Algebra
- Discrete Mathematics
- Set Theory
- Graph Theory
- Numerical Analysis
- Complex Analysis
- Probability
- Statistics
- and more

Engineering Disciplines

- Error Analysis
- Metrics
- Methodologies
  - ✔ Configuration Management
  - ✔ Quality Assurance
  - ✔ Testing
  - ✔ Debugging
  - ✔ Maintenance
  - ✔ Development
- Project Management

Communication Skills

- Oral
- Written

Additional Skills Needed by Software Engineers

- Creativity
- Ingenuity
- Interpersonal Communications
- Analytical Thinking
- Logical Thinking
- Organization
- ... and more

Software Engineering Training

Given that software engineering is at least as involved, as technical, and as rapidly evolving as more formally recognized forms of engineering (e.g., electronics engineering), it is far more appropriate to speak of the need for software engineering **education** , than for the need for software engineering **training** .

A Healthy Dose of Reality

It is given that:

- ✔ Software engineering education at the university level is virtually non-existent
- ✔ The need for properly educated software engineers is enormous
- ✔ There are literally hundreds of thousands of "programmers" already in the field

Observations

- "Scientific" programmers are far less likely to accept software engineering as a discipline than are "business" programmers.
- "Scientific" programmers are better equipped to handle the mathematics and technical rigor associated with software engineering than are their "business" counterparts.
- Europeans more readily accept the rigorously technical nature of software engineering than do Americans.
- The Japanese will do whatever is required, but seem to prefer rote methods as opposed to thinking.
- The Japanese are far more willing to invest in the long term than are their American or European counterparts.

Observations (Continued)

- Software engineering in the United States often involves the installation and automation of technologies which are at least ten (10) years out of date.
- The "half life" of software engineering technology seems to be about three to four (3-4) years.
- Computer Aided Software Engineering (CASE) is a very frequently used term, but its actual implementation, in most cases, involves little more than an automated graphic design tool.
- People other than programmers, e.g., managers, are often ignored by CASE.
- Many life-cycle activities are ignored by most CASE environments, e.g., activities outside of development are often ignored.

Observations (Continued)

- Many so-called CASE environments not only automate outdated technology, but often resemble a hodge podge of unrelated tools.
- Too many CASE environments are one-dimensional (e.g., UNIX™), and too few are two-dimensional (e.g., Macintosh™-like).
- Too few environments are seamless (e.g., Rational™).
- Little thought is given to integrating CASE tools with in-house standards, government standards, training, methodologies, and development platforms.
- Many colleges and universities ignore CASE altogether. Those that do pay attention to CASE often fail to realize that CASE tools introduced to freshmen are often out-of-date before those freshmen graduate four years later.

Observations (Continued)

- Many students, like many programmers (There are very few practicing "software engineers."), have a reverence for both the past and the difficult. They shun efforts to change the status quo.
- New technology is seldom what we expect it to be.
- Most of the training given to software practitioners places too much emphasis on coding and too little emphasis on software engineering.
- Those already in, and those just entering the software community, are given little motivation to make use of such things as CASE tools — *primarily because software people (managers and technical people alike) do not view themselves as being responsible for their own actions.*

omit

Robert B. MacDonald

(NOTES)

Software Engineering as an Engineering Discipline

Norman Gibbs

Education Program

Software Engineering Institute

Carnegie Mellon University
Pittsburgh, PA 15213

Sponsored by the U.S. Department of Defense

Challenge

"The SEI shall develop and conduct courses and seminars with respect to the evolving state of the art and practice in software engineering for mission-critical computer systems as well as the results of its activities in technology transition. It shall influence software engineering curricula development throughout the education community."

-SEI Charter

Education Program Goals

- Increase the number of highly qualified software engineers
  - new software engineers
  - existing practitioners
- Be the leading center of expertise for software engineering education and training

Education Program Objectives

- Accelerate the development of software engineering programs in academia to increase the quality and quantity of the next generation of software engineers
- Enhance continuing education opportunities to improve the quality of the current generation of software engineers

Problem Definition

- Substantial advances in the practice of software engineering require a solid educational foundation
- Rapid developments in software engineering vs. enormous inertia of the educational system
- Need more new software engineers
- Need to improve productivity of current software engineers
- Need more qualified educators
- Need more and improved educational materials
- Need to identify the fundamental principles of an emerging discipline
- Must balance principles and current best practice

Strategy

- Identify, organize, and *document* the body of *knowledge* of software engineering
- Create and disseminate high-quality educational *support materials*
- Design, *develop*, and keep current a model *curriculum* for a graduate degree
- Promote the implementation and *delivery of* software engineering *curricula* in academia, industry, and government
- Increase the number of quality educators through *faculty development* activities
- Explore the use of *advanced technology* for delivery of software engineering education and training

Education Program Projects

- Graduate Curriculum
- Undergraduate Software Engineering Education
- Video Dissemination
- Advanced Learning Technologies

Graduate Curriculum Project

Purpose:

- Promote graduate-level software engineering education

Major goals:

- Identify, organize, and document the body of knowledge that might be taught in graduate-level software engineering programs
- Design, develop, and support a curriculum for a Master of Software Engineering (MSE) degree

Carnegie Mellon University
Software Engineering Institute

Graduate Curriculum Project

- Module development
- Support material development
- MSE curriculum
- Ada artifact
- Addison-Wesley book series

Nov 10EDneg8

Carnegie Mellon University
Software Engineering Institute

Undergraduate Software Engineering Education Project

Purpose:

- Influence existing computer science undergraduate programs to increase the quality and quantity of software engineering content

Nov 10EDneg9

Undergraduate Software Engineering Education Project

- Providing support materials for teaching the senior level software engineering project course
- Identifying the needs of educators and students for more software engineering content
- Examining the use of Ada in the undergraduate curriculum and Ada as a first programming language

Advanced Learning Technologies Project

Purpose:

- Demonstrate the applicability to software engineering of technologically advanced learning media, such as
  - interactive video discs
  - intelligent tutors
  - advice-giving expert systems
  - compact disc read-only memory
  - digital video interactive equipment

Carnegie Mellon University
Software Engineering Institute

Advanced Learning Technologies Project

- First prototype developed to demonstrate feasibility
- Producing a course for demonstrating the applicability of digital video interactive and compact disc read-only memory technologies

Nov 10EDneg12

Carnegie Mellon University
Software Engineering Institute

Video Dissemination Project

Purpose:

- Produce and deliver courses on modern software engineering methods to practitioners in cooperation with the academic, government and industrial communities.

Nov 10EDneg13

Video Dissemination Project

- Studio completed
- Pilot Course offered at CMU in January 1988
- Academic Series
  - Formal Methods in Software Engineering
  - Software Project Management
- Continuing Education Series
- Current Technology Series

Nov 10EDneg14

Faculty Development Workshops

Purpose: To transition SEI educational materials to educators

- Fall 1986 -- Pittsburgh -- 110 attendees
- Spring 1987 -- Pittsburgh -- 140 attendees
- Fall 1987 -- Pittsburgh -- 100 attendees
- Spring 1988 -- Fairfax, Virginia -- 156 attendees
- Winter 1988 -- Scottsdale, Arizona, January 6,7
- Summer 1989 -- Pittsburgh, week of July 17

Nov 10EDneg15

Annual Software Engineering Education Conference

Purpose: Promote an exchange of ideas and methods among educators from academic, government, and industrial education and training communities

- Spring 1987 - first SEI conference, 200 attendees
- Spring 1988 - second conference, first remote, 152 attendees
- Summer 1989 - Pittsburgh, week of July 17

Springer-Verlag contract for proceedings

Nov 10EDneg16

Academic Affiliates Function

- Provides a mechanism for interactions between the SEI and the academic community
- Administered by the Education Program for the entire SEI

Nov 10EDneg17

Academic Affiliates

Accomplishments:

- 41 academic institutions are academic affiliates
- 25 scientists have worked at the SEI under the visiting scientist program
- First MSE primary test site has been designated

Affiliate Activity - 1

- Module Development
  - Arizona State (3), Boston University, California at Irvine, George Mason, Lehigh, Maryland, Pittsburgh (3), Seattle (2), Stirling, Wichita State (2), William and Mary, USC
- Support Material Development
  - Arizona State, Pittsburgh, Stirling, Wayne State, Wichita State (2)
- Video Pilot
  - California State at Sacramento, East Tennessee State, George Mason, North Carolina, Wichita State

Affiliate Activity - 2

- Other SEI Programs
  - California at Sacramento, Columbia, Michigan
- Curriculum Design Workshop
  - Arizona State, George Mason, SUNY at Binghamton, Rochester Institute of Technology, Wichita State, William and Mary
- Primary Test Site for MSE
  - Wichita State
- Ada in Freshman Courses
  - Arizona State, Maryland, Washington, West Virginia

Nov 10EDneg20

Current Academic Affiliates

Air Force Institute of Technology
Arizona State University
Boston University
California State University, Sacramento
Clemson University
Columbia University
East Tennessee State University
George Mason University
Lehigh University
Naval Postgraduate School
Old Dominion University
Purdue University
Queen's University at Kingston
Queen's University, Belfast
Rochester Institute of Technology
School of Informatics, Polytechnic University of Madrid
Seattle University
State University of New York at Binghamton
Temple University
Texas A&M University
United States Air Force Academy
University of California, Irvine
University of Houston, Clear Lake
University of Illinois at Urbana-Champaign
University of Maryland
University of Michigan
University of North Carolina, Chapel Hill
University of Pittsburgh
University of Texas, Austin
University of Southern California
The University of Stirling
The University of Strathclyde
University of Tennessee, Knoxville
University of Washington
Virginia Polytechnic Institute and State University
Wayne State University
The University of West Florida
West Virginia University
The Wichita State University
The College of William and Mary
Wright State University

Nov 10EDneg21

Uniqueness

- International focus for software engineering education
- Permanent staff in support of curricula effort
- Catalyst for new ideas
- Notion of an evolving curriculum
- Visiting scientists
- CMU connection
- A research infrastructure exists; we provide educational infrastructure
- A center for expertise unlike that in any discipline

Nov 10EDneg22

CMU MSE

- Two year program
- First year remote
  - six core courses
    - -- Software Systems Engineering
    - -- Specificiation of Software Systems
    - -- Principles and Applications of Software Design
    - -- Software Generation and Maintenance
    - -- Software Verification and Validation
    - -- Software Project Management
  - two electives

Nov 10EDneg23

CMU MSE

- Second year in residence
  - admissions procedure
  - project in three phases -- planning, execution, evaluation
  - visits by leading software engineers plus student tasks to study their work
  - advanced courses
  - advanced electives in software engineering related topics

Nov 10EDneg24

omit

Computer-Aided Software Engineering Environments for Real-Time Systems

Panel Chair and Moderator: **Charles W. McKay**

Panel

Migual A. Carrio, Jr.

E. Douglas Jensen

A Conceptual Model for Evolving Run Time Support of Mission and Safety Critical Components in Large, Complex, Distributed Systems

Charles W. McKay

A Conceptual Model for Evolving Run Time Support of Mission and Safety Critical Components in Large, Complex, Distributed Systems

Charles W. McKay
SERC/HTL@UHCL

INTRODUCTION

Large, complex, distributed systems should be evolved to maximize life cycle support for non-stop operation of mission and safety critical components. This paper outlines the key issues and a recommended approach for tailoring a conceptual model of Ada run time support environments to meet the specific needs of such an application. Prerequisite concepts for this model have been described previously by this author (e.g., McKay, 1987) and are summarized in Figures 1 through 9.

This model proposes upward-compatible extensions to a previously published model of Ada run time environments from the ARTEWG (Ada Run Time Environment Working Group). Whereas the first model was used to identify Ada run time: requirements, dependencies, issues, features, and options for single processor applications; the particular needs for distributed processing were not explicitly described. The purpose of this extended model is to address the needed systems software support for Ada application programs in distributed computing environments.

OVERVIEW OF RELATED CONCEPTS AND TERMS

"Overview"

Consistent with earlier documents by ARTEWG, this model supports multiprogramming. Specifically, this model is intended to support not only the distribution of entities of a single Ada program across a distributed processing environment but also to support the distribution of entities of multiple Ada programs across such an environment. The upward-compatible extensions to the original ARTEWG model address a spectrum of needs found in: multiprocessors, local area networks, wide area networks, wide area networks of integrated local area networks, and other forms of distributed computing systems. Furthermore the extensibility and tailorability of the model are intended to facilitate the support of such operational requirements as: non-stop operation, fault tolerance, multilevel security, and others.

From the perspective of an Ada application program which is to have selected entities mapped to components of a distributed computing system, the run time environment interface issues may be

macroscopically divided into four sets. From the top-down view of an Ada application program, the ordered sets are identified as: Distributed Information Services (DIS), Distributed Communication Services (DCS), Distributed Configuration Control Services (DCCS), and Distributed Operating Systems (DOS). Not all applications will need all four sets of services. Even those that do need all four are likely to benefit from tailoring to meet application specific needs. Consistent with the philosophy underlying previous ARTEWG work, such tailorability will be facilitated by appropriate subsections of the Catalog of Interface Features and Options for each set of services.

Figure 10 depicts a logical view of these services at a site in a distributed computing system. Applications software components and users share the perspective labeled Distributed Applications Services (DAS) of the DIS, DCS, and the DCCS. Not explicitly shown is the DOS which provides integrated support for all of the other sets.

Before proceeding with an introduction to the four interface sets, it will be helpful to clarify and distinguish four terms: services, resources, architecture of computing systems, and bare machine philosophy. Services refer to operations performed on behalf of a user. Resources refer to items available to or from an object or user where the resources are distinct from the services that provide, consume, or affect them. For example, in the Ada statement "PUSH(x);", PUSH refers to the service and X refers to the resource.

For purposes of this model, the architecture of computing systems refers to:

> The structural organization and the interrelationships of the software, hardware, and operational interface elements that comprise the system.

Also for purposes of this model, a bare machine philosophy recommends that all source code for: applications software, subsystems software, and systems software be written in Ada and transformed into executable object code by the same compiler and associated tools. This may exclude some small percentage of code required to interface machine dependent idiosyncracies to the kernel of the system software. The reader should note that the bare machine philosophy is in sharp contrast to the approach of retrofitting Ada application software to systems and subsystems software written in other languages and often representing older models and paradigms which are inconsistent with the more modern software engineering principles embodied in Ada.

"Distributed Operating System"

The DOS encapsulates the system hardware. Three major criteria for a good operating system, including DOS's, are:

1. An explicit set of policies for managing the integrated operation of all categories of system services and resources which are to be sharable among independent application program components and users.

2. An explicit set of management modules (software, hardware, and operational interfaces) to implement and enforce the policies.

3. A precise model of the operating system which provides rules and guidelines for modifications including extensions, regressions, and reconfigurations.

DOS's typically involve some combination of ten major categories of system services and resources:

1. Workload, jobs, processes, tasks, and processors

2. Memory: primary..secondary

3. Devices and buses

4. Data and information

5. Stable Interface Sets: Users and applications software

6. Stable Interface Sets: Major subsystems and systems software, hardware, and operational interfaces

7. Communications: systems..applications

8. Configuration control:
   - System services and resources..applications software and users
   - Normal processing..exception processing

9. Time and events

10. Access control including security and integrity

"Distributed Information Services"

Whenever an information or data resource or service is to be shared by multiple application programs or users in a distributed, on-line environment, the application access should be provided at a virtual interface set known as the DIS. For example, access and manipulation of elements of a distributed data base should depend upon compilation visibility of the DIS to the application software.

"Distributed Communications Services"

Communications resources and services which are to be shared

among multiple application programs or users in a distributed, on-line environment should be accessible to the application at a virtual interface set (ie, the DCS). Note that the DCS may also view the DIS as a user and vice versa. For example, a user request from the DAS to the local DIS for a resource of data might result in a transparent (to the user) request to the local DCS to obtain the data resource from a remote site.

"Distributed Configuration Control Services"

As shown in Figure 1, the DCCS virtual interface set has visibility of the: DAS, DIS, and DCS. This provides a unique opportunity to exploit known semantics about the various components that provide the services and resources of the three in order to monitor, manipulate, and control distributed processing. For example, programs can be distributed dynamically, processes can be advanced or blocked, parameterized performance monitoring can be enabled or disabled, and interactive debugging and reconfiguration can be supported among remote sites. The reader should note that this is a much higher semantic level of configuration control services than is typically found in underlying operating systems.

CONCEPTUAL MODEL OF THE EXECUTION ENVIRONMENT OF A DISTRIBUTED COMPUTING SYSTEM ARCHITECTURE

"Abstractions: Four Functional Layers and Major Interface Sets"

As shown in Figure 11, the major interface sets extend from the DIS through the DCS, DCCS, and DOS functional layers down to the hardware. These virtual interface sets built from a common Catalog of Interface Features and Options provide a perspective of a Portable Common Execution Environment (PCEE) to the application program components and users.

"Issues Common to Each Layer"

Eight major issues common to all layers and major categories of system services and resources are:

1. Five Management Responsibilities
   a. Track the Status and Utilization of Each Service and Resource
   b. Enforce Policies
   c. Schedule
   d. Dispatch
   e. Reclaim (e.g. Completion, Unrecoverable Fault, Abortion, Preemption)

2. Measurement, Testing, Debugging

3. Abstractions: Objects, Messages, Semantic Models

4. Synchronization

5. Protection

6. Errors and Faults

7. Naming and Identification

8. Baseline Modification

"Issues with a Large
Potential Return-on-Investment
for Optimization Across Layers"

Six issues with a high potential return-on-investment for optimization across layers are:

1. Reusability

2. Interoperability and Transportability

3. System Measurements, Testing, and Debugging

4. Optimum Location and Reconfiguration of Services and Resources

5. Universal Scheduling

6. Universal State Consistency and Congruity

"Important Issues for
Supporting Mission and
Safety Critical Components"

Unfortunately, the difficult challenges of supporting Mission and Safety Critical (MASC) components in large, complex, distributed systems are less understood than the issues identified on the other axes. This is particularly true in applications with requirements for non-stop operation, fault tolerance, and meeting real time deadlines of both periodic and aperiodic processes. As an example, twelve issues and components of one proposed model are given below.

Issues and Components of the
Clear Lake Model for Run Time
Support of Mission and Safety
Critical Components:

1. A tailorable RTSE developed & sustained in Ada upon bare machines

2. Software structuring which facilitates: firewalling,

layered recovery/capability, dynamic reconfiguration and extensibility

3. Pools of processes and processors capable of non-stop operation in a fault-tolerant environment

4. A command language interface between the SIS of the integration environment's PCEE and the SIS of the target environment's PCEE

5. System-wide, lifecycle-unique identification of all objects and transactions/subtransactions

6. Dynamic, multilevel security in the integration & target environments

7. A message interface which supports three forms of communication among clusters: asynchronous send/receive with 'no waits', remote procedure call, Ada rendezvous

8. Hierarchical runtime structure of the threads-of-control

9. A redundancy management subsystem for services and resources which life and property depend upon

10. A stable storage subsystem for each cluster

11. A management subsystem for distributed, nested transactions

12. A multiversion, fault-tolerant programming capability with a granularity within any program which extends at least to the subtransaction level and explicitly identifies the recovery capabilities at that level

FIRST LEVEL MAPPING TO AN IMPLEMENTATION MODEL

Figure 12 depicts an extension of the original ARTEWG model to include support for distributing entities of Ada application programs across components of a distributed computing system. Scenarios are useful for explaining the model. Suppose an Ada application programmer logs into an APSE (Ada Programming Support Environment) to prepare a source code version of a program to be deployed and operated in a distributed target environment. The capabilities assigned to the programmer on this project determine whether the DIS, DCS, DCCS, or DOS virtual interface sets are to be available to this programmer. (Note that these features and options are part of the extended runtime library --ie, XRTL-- which are documented in the CIFO and legally go beyond the minimum set of components in the runtime library which are required for validation --ie, RTL.) Along with imports from the explicitly "with'ed" applications library, every shareable service and resource available at the four interface sets of the XRTL may be explicitly referenced within the application source code of the

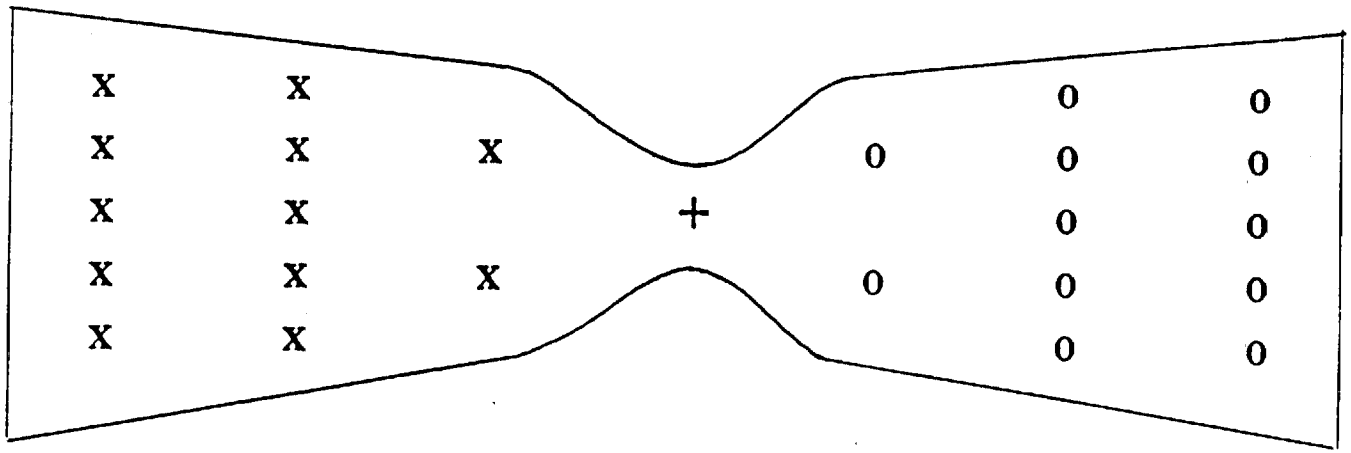
authorized application programmer. (Note that the XRTL is an instance of the CIFO.) Now the source code can be compiled and an inventory of the references to the XRTL and RTL will partially determine the Ada components which will be selected from these two system level repositories to be transformed by the same compiler as the application code and exported to the bare machine. However, two additional inputs are needed to determine the remainder of the object code to be exported to the target environment. First, the project object base should be checked to determine if non-functional requirements which should be transparent to the application programmer are to apply to this program. For example, the program might be required to execute in a B3 class, multilevel secure environment. These non-functional transparent-to-the-source-code requirements may cause still additional components from the XRTL and RTL to be transformed for deployment. Finally, the idiosyncracies of the hardware itself may cause a small percent of non-Ada code to be required for export to the target environment.

ADDITIONAL WORK NEEDED TO FURTHER DEVELOP THE MODELS

Although this is a potentially very large and complex undertaking which can benefit from work stimulated principally from issues on any one of the functional layers of this model, the most crucial issues are believed to be the support of Mission and Safety Critical components in distributed, embedded computing systems. An integrated approach to these interface sets with MASC component support as the first priority should be developed and prototyped.

BIBLIOGRAPHY

McKay, C. "A Proposed Framework for the Tools and Rules to Support the Life Cycle of the Space Station Program", IEEE Compass 1987, TH0196-6/87/0000-0033.



Host Environment (shown as Distributed)

- Develop
- Substain

Integration Environment

- Monitor & Sustain Current Baseline
- Control Integration & Evolution
- Support Emergency Interactions

Target Environment

- Deploy
- Operate

Figure 1 Three Types of Environments Addressed by Software Engineering

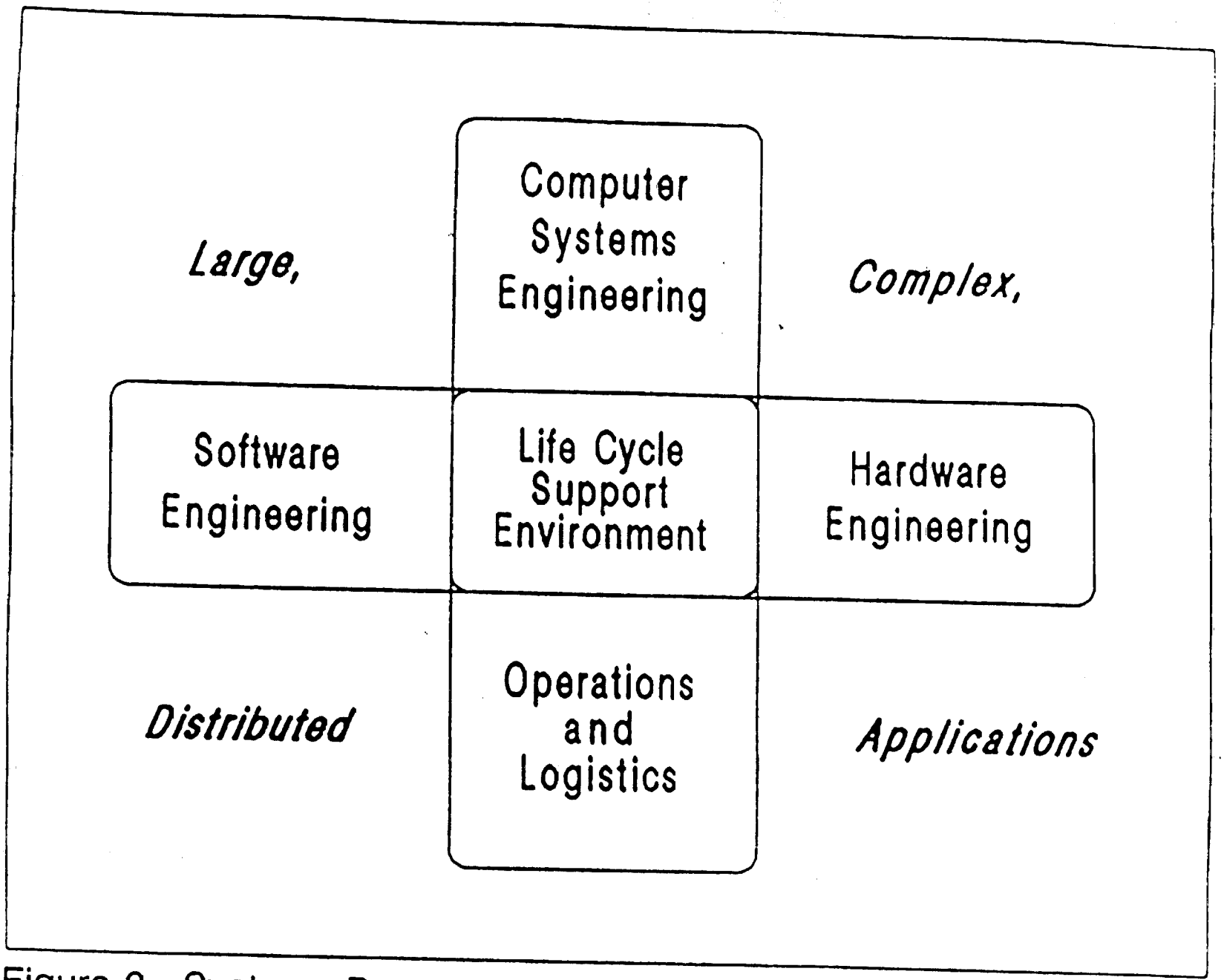
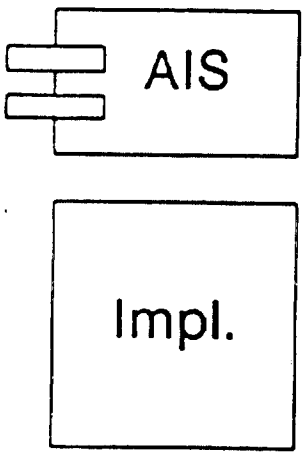


Figure 2. Systems Perspective of a Life Cycle Support Environment

Figure 3. Objects

Passive

Borrows thread of control

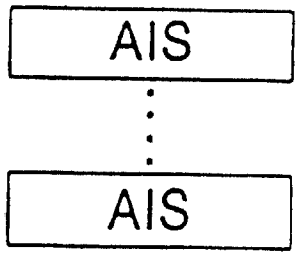


Often called Triggers in AI/KB/ES

Neutral

Contain types & values only

Eg, "Relations" in Dr. Codd's Relational Data Model



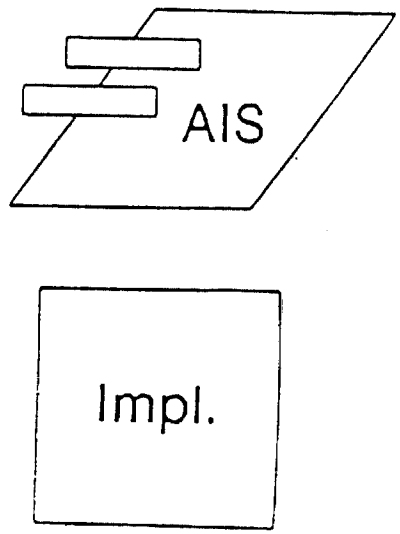
Note: Values are accessable to a collection of Relational Operators

- Untyped (normally)
- Ada Typed (needed)

Active

Possess their own thread of control (eg, Ada task)

Often called Daemons in AI/KB/ES



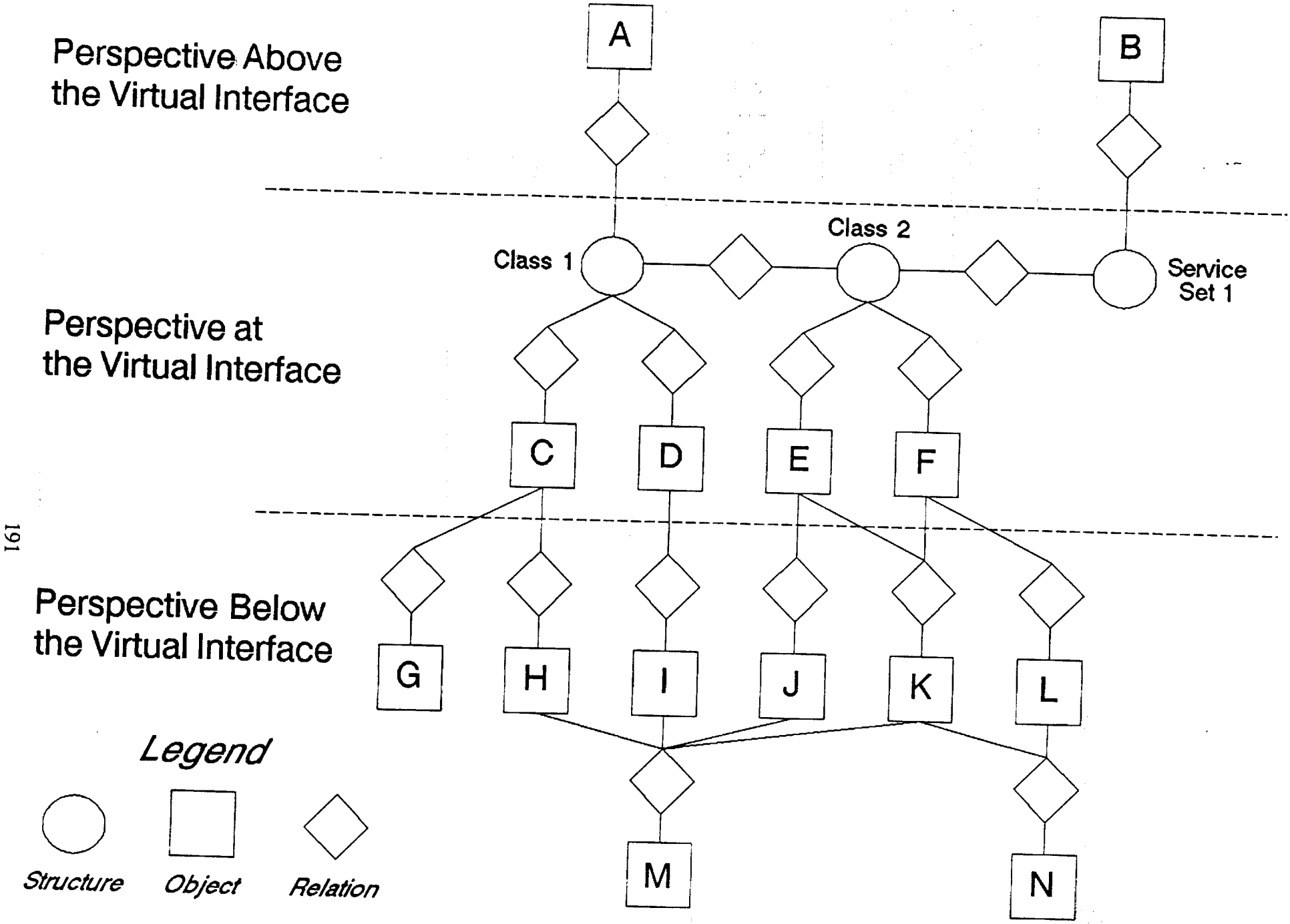


Figure 4. Virtual Interface Perspectives

Snapshots of Evolving Stable Frameworks

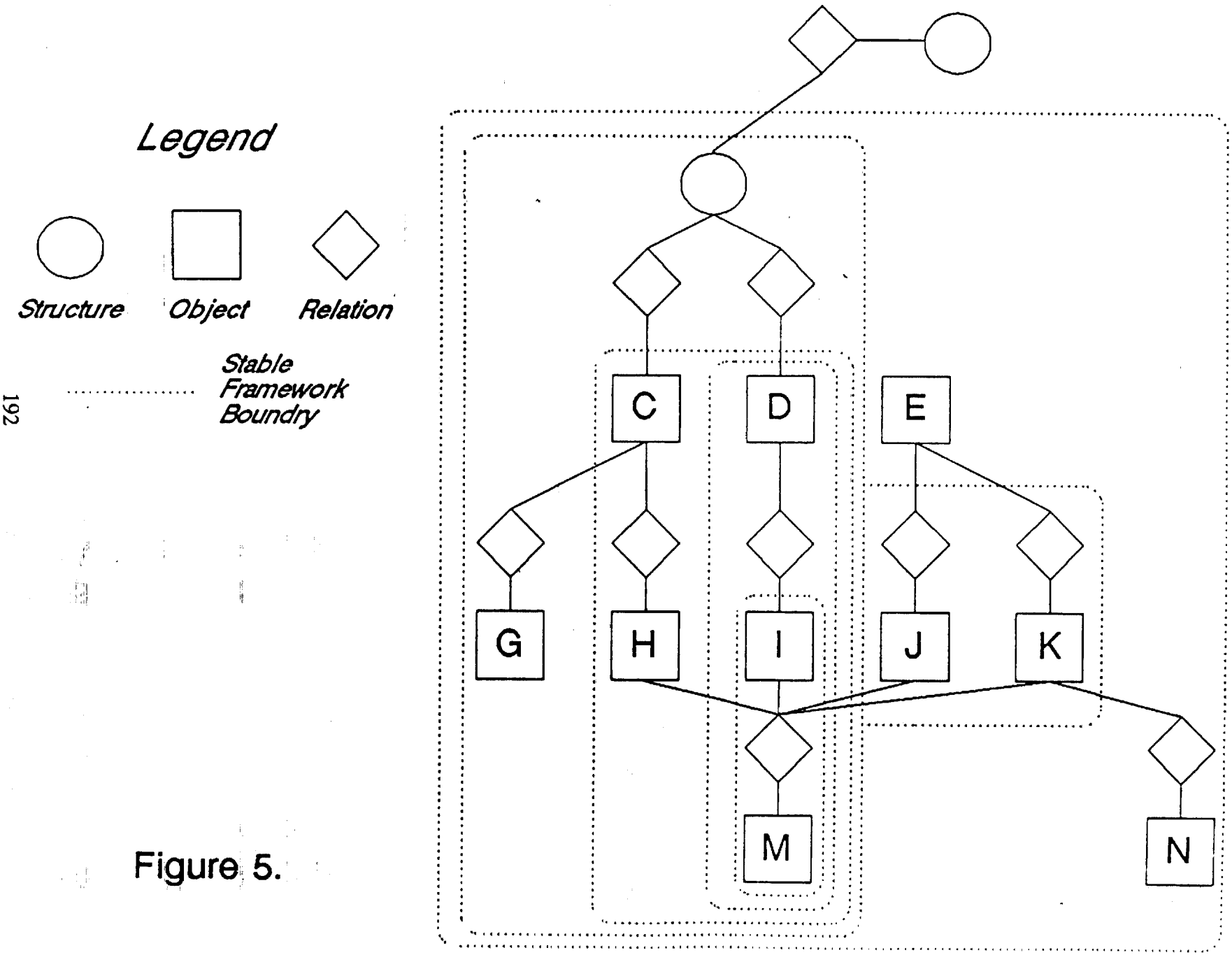


Figure 5.

FIRST LEVEL ABSTRACTIONS

Conceptual Model of a Life Cycle Support Environment

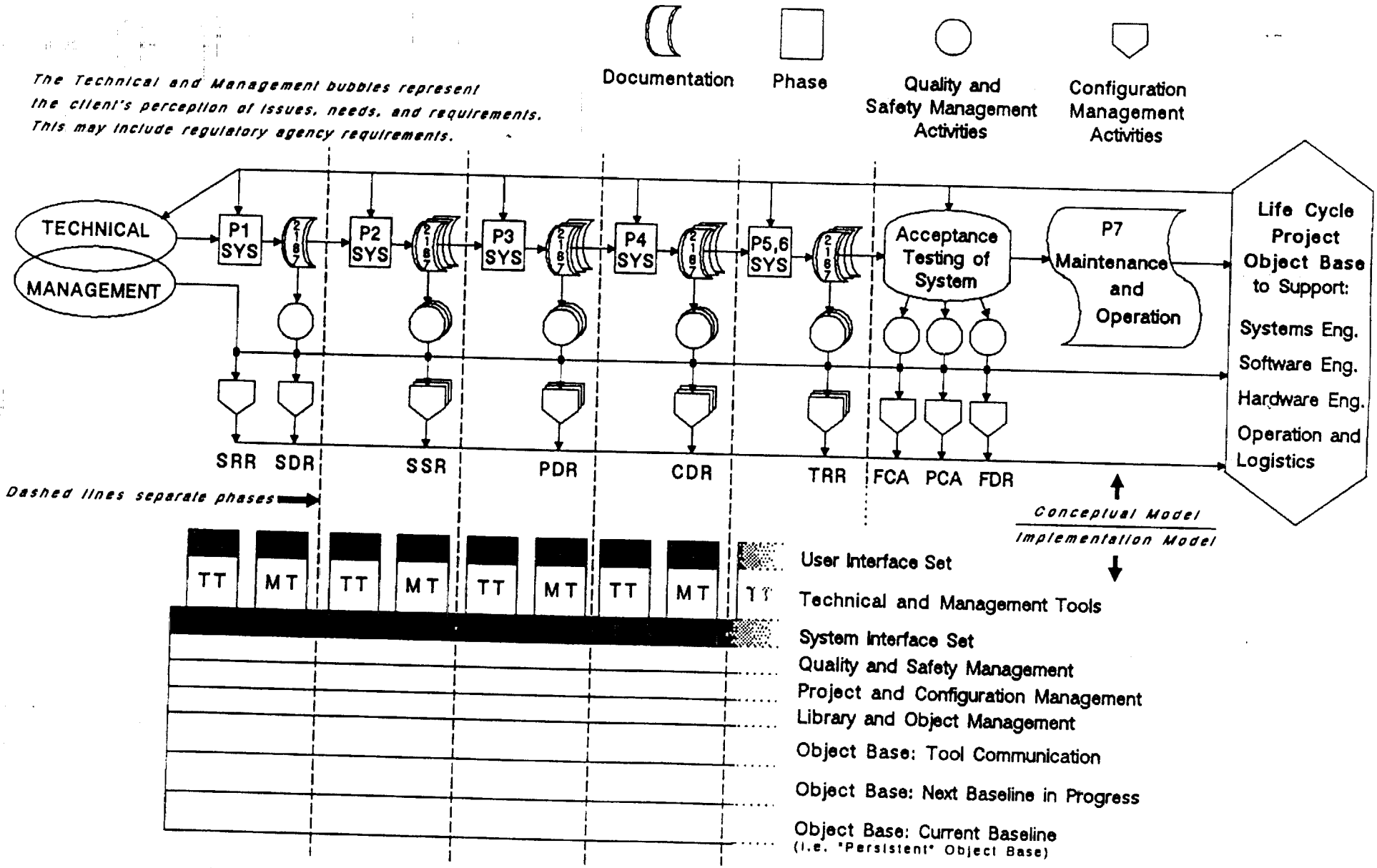


Figure 6. Implementation Model of this Life Cycle Support Environment

SECOND LEVEL ABSTRACTIONS

A Taxonomy of Taxonomies

The Life Cycle Support Environment has attributed relationships to all interfaces, phases, quality and safety activities, and project configuration management activities.

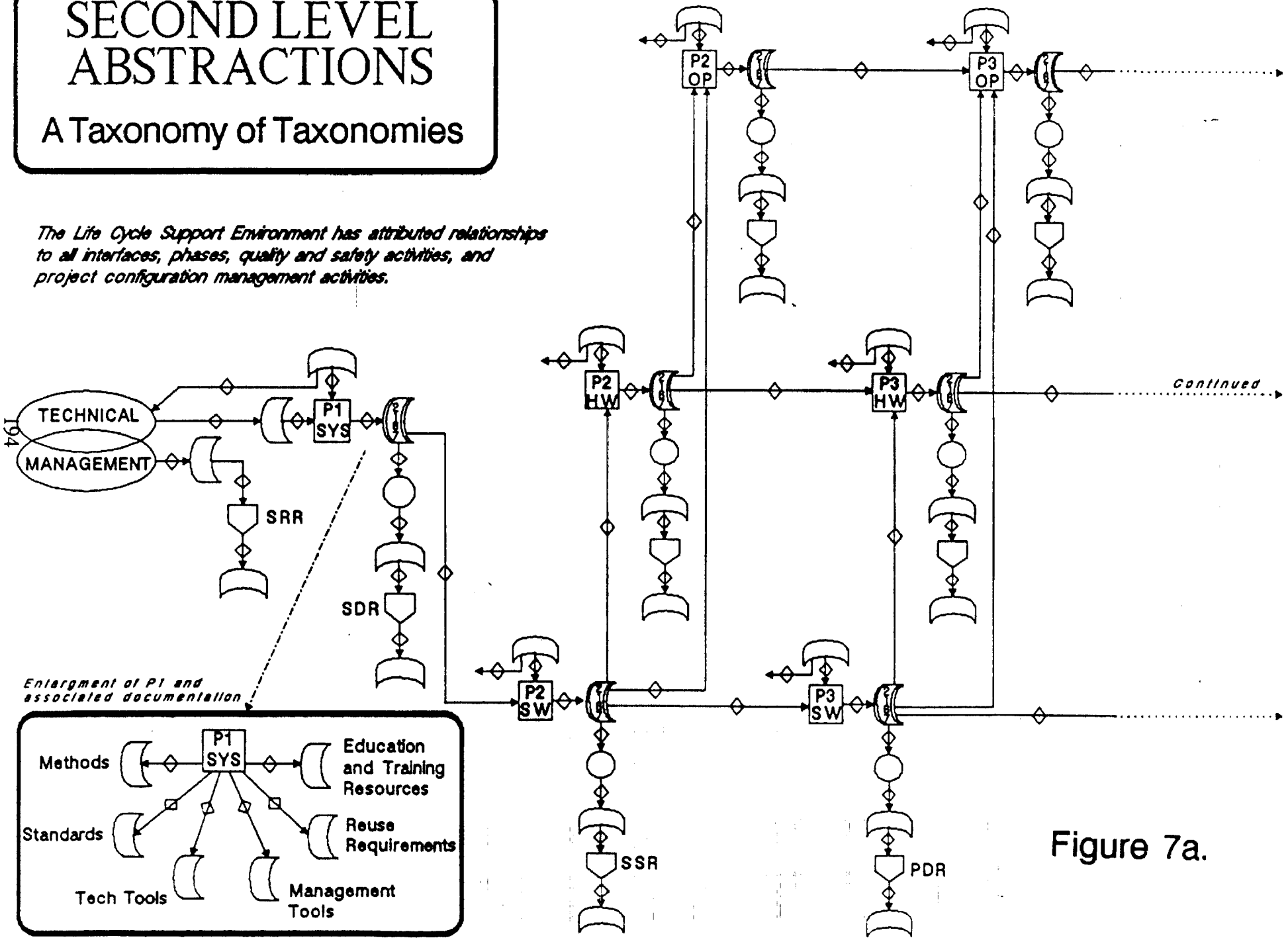


Figure 7a.

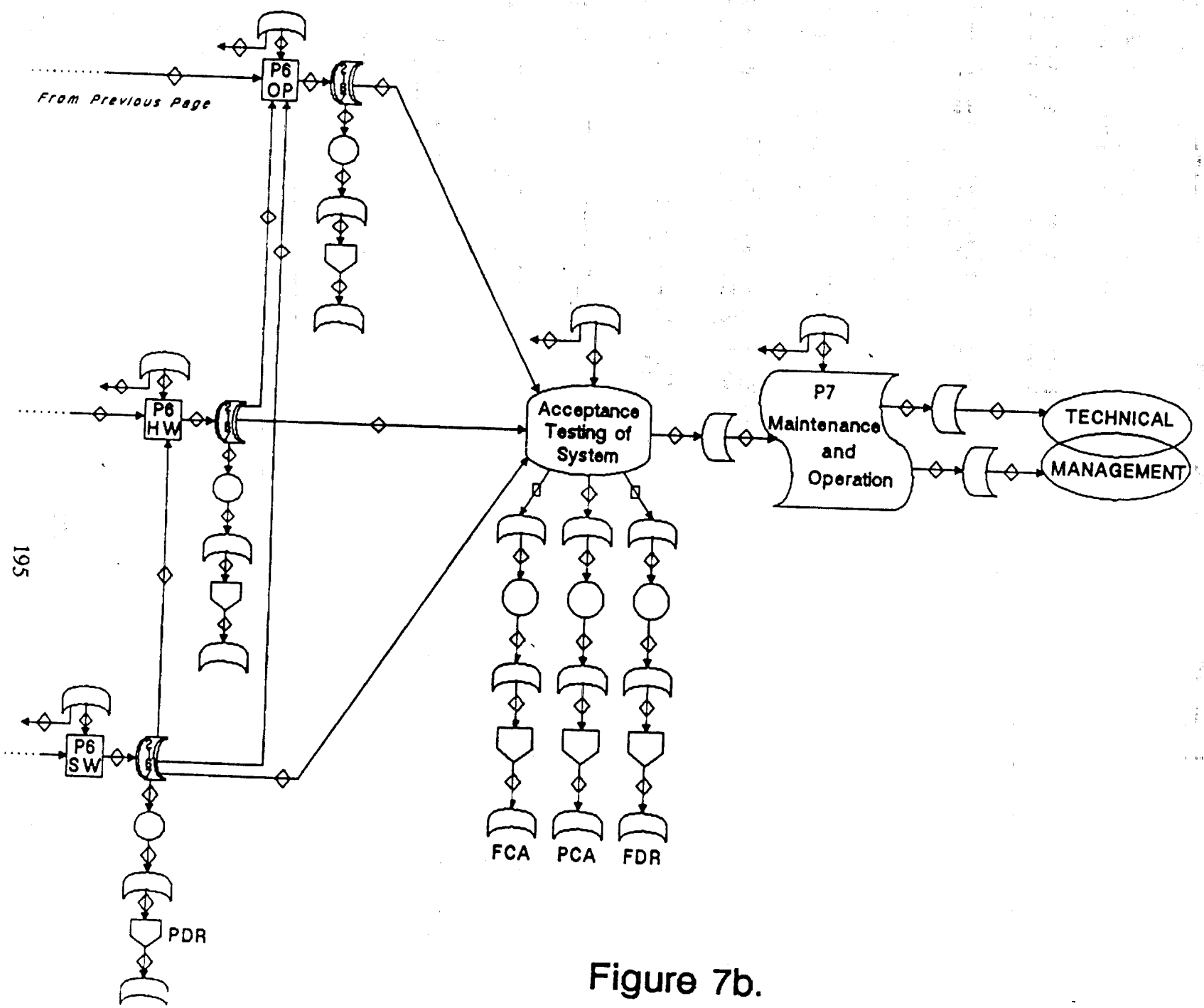

From Previous Page
P6 OP
P6 HW
P6 SW
PDR
Acceptance Testing of System
FCA
PCA
FDR
P7 Maintenance and Operation
TECHNICAL
MANAGEMENT

Figure 7b.

THIRD LEVEL ABSTRACTIONS

Library and Component Management

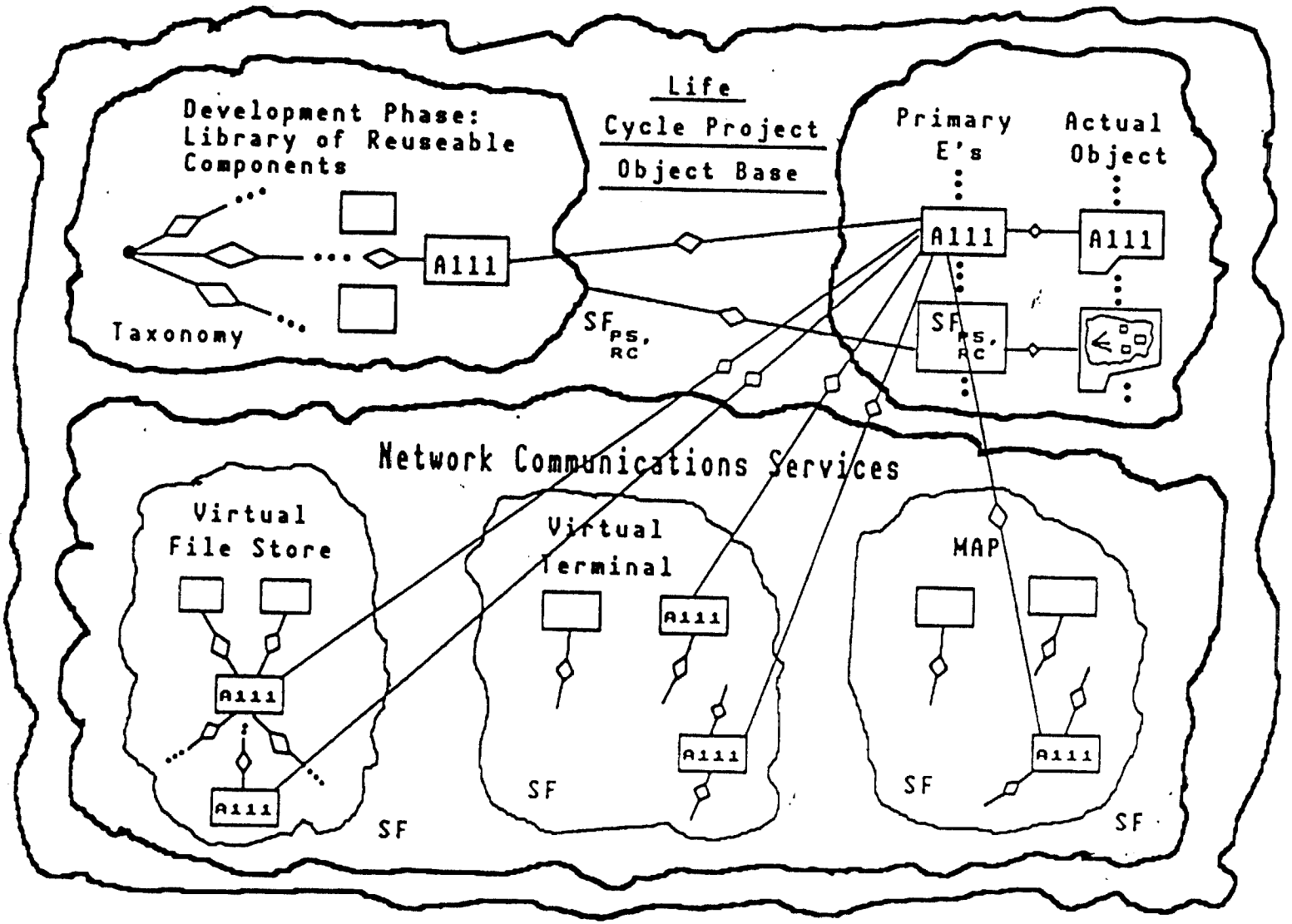


Figure 8.

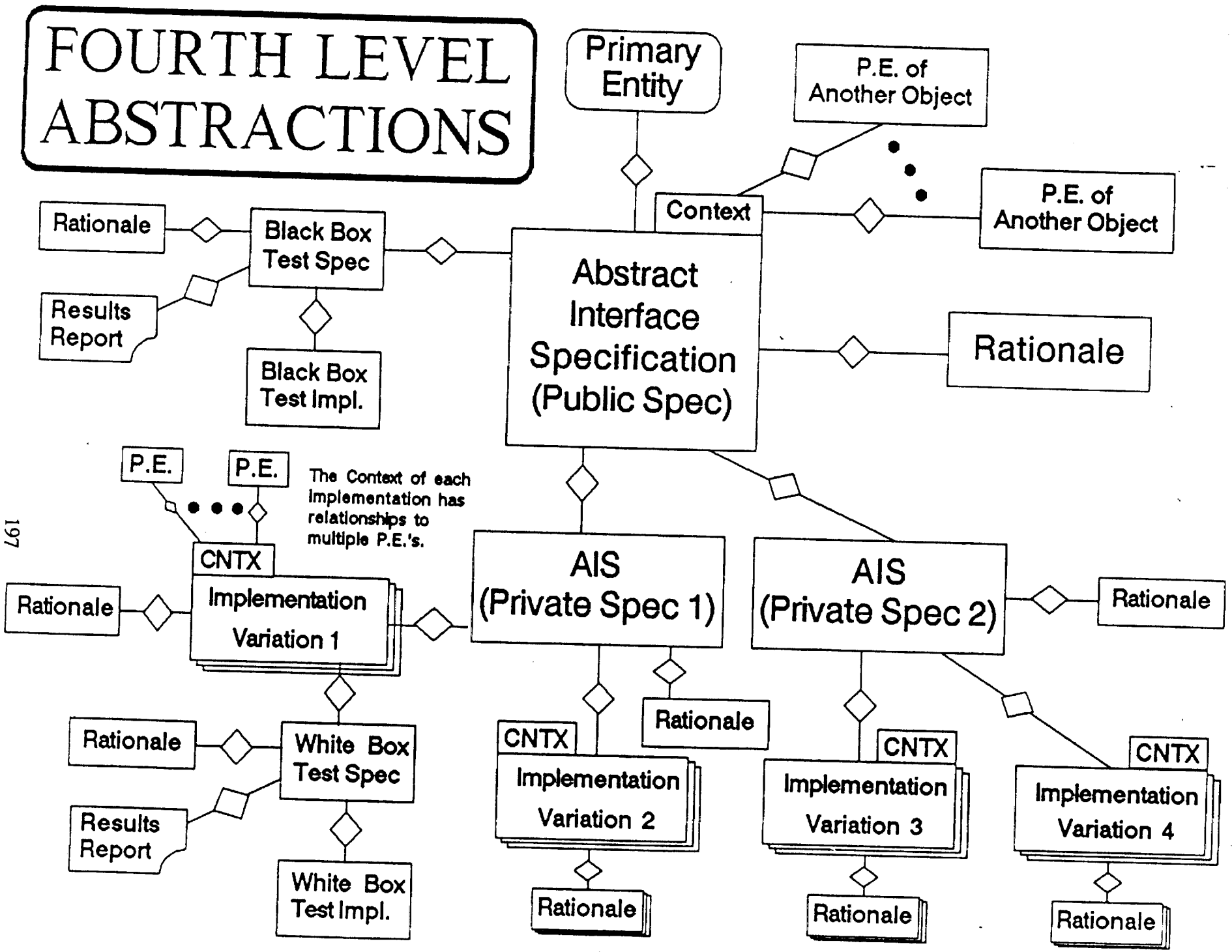
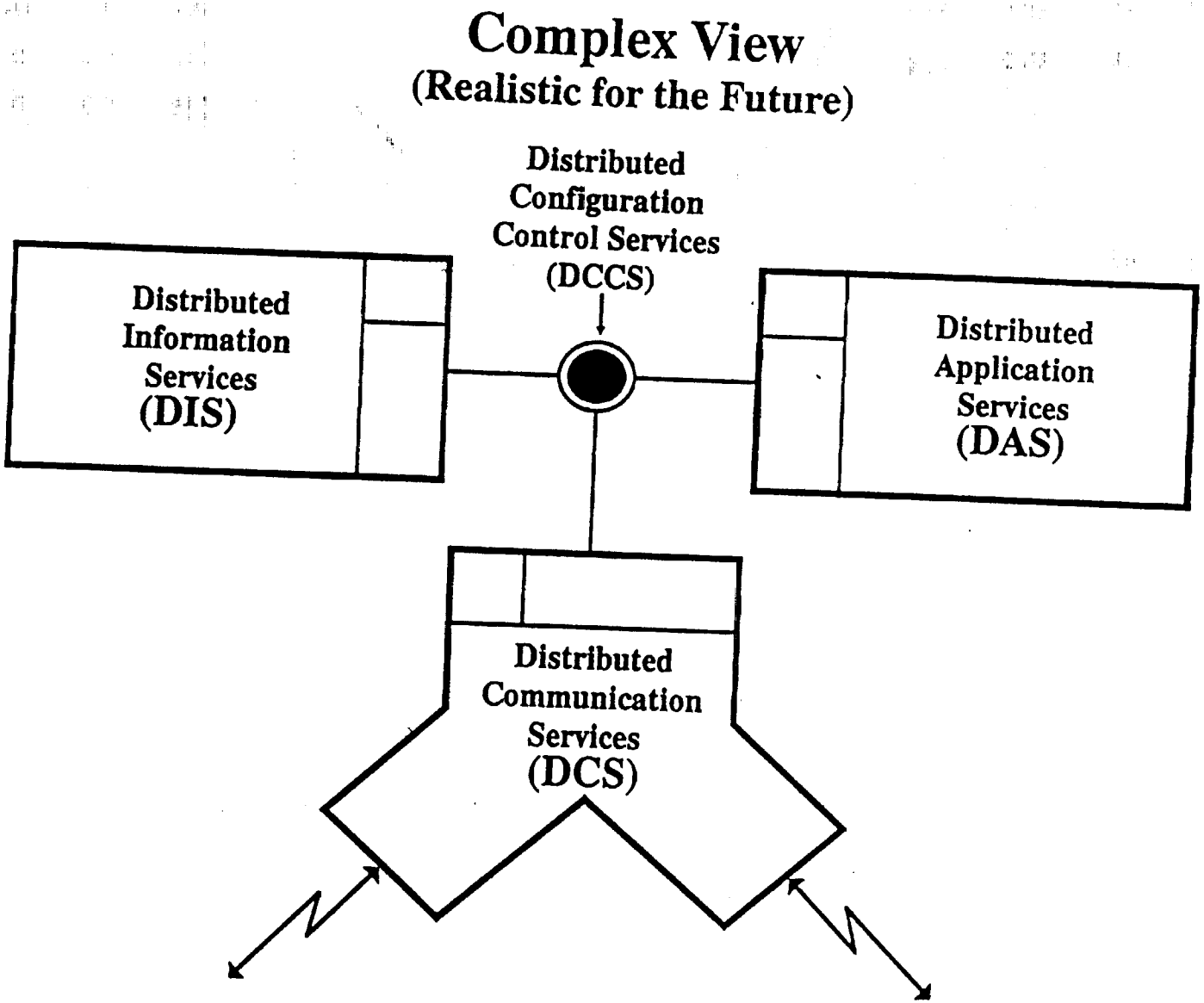

FOURTH LEVEL ABSTRACTIONS
Primary Entity
P.E. of Another Object
P.E. of Another Object
Context
Rationale
Black Box Test Spec
Results Report
Black Box Test Impl.
Abstract Interface Specification (Public Spec)
Rationale
P.E.
P.E.
The Context of each implementation has relationships to multiple P.E.'s.
CNTX
Rationale
Implementation Variation 1
AIS (Private Spec 1)
AIS (Private Spec 2)
Rationale
Rationale
White Box Test Spec
Results Report
White Box Test Impl.
CNTX
Implementation Variation 2
Rationale
Rationale
CNTX
Implementation Variation 3
Rationale
CNTX
Implementation Variation 4
Rationale

Figure 9.



Distributed Systems Cluster

Figure 10.

Distributed System Architecture Model

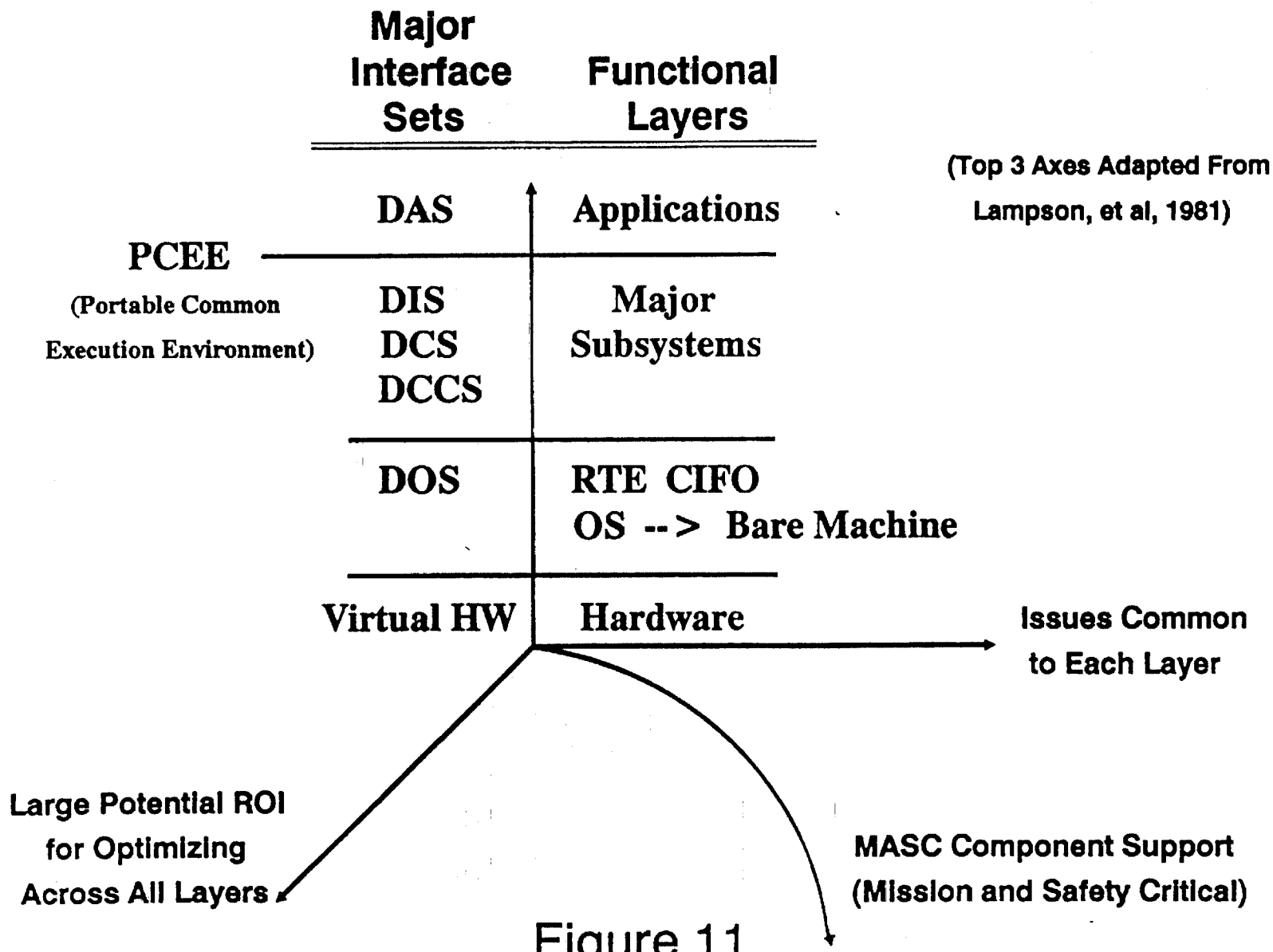


Figure 11.

A Model Supporting a ' Bare Machine' Philosophy for Ada Runtime Support Environments (Ada RTSE's) used in Distributed Computing Systems

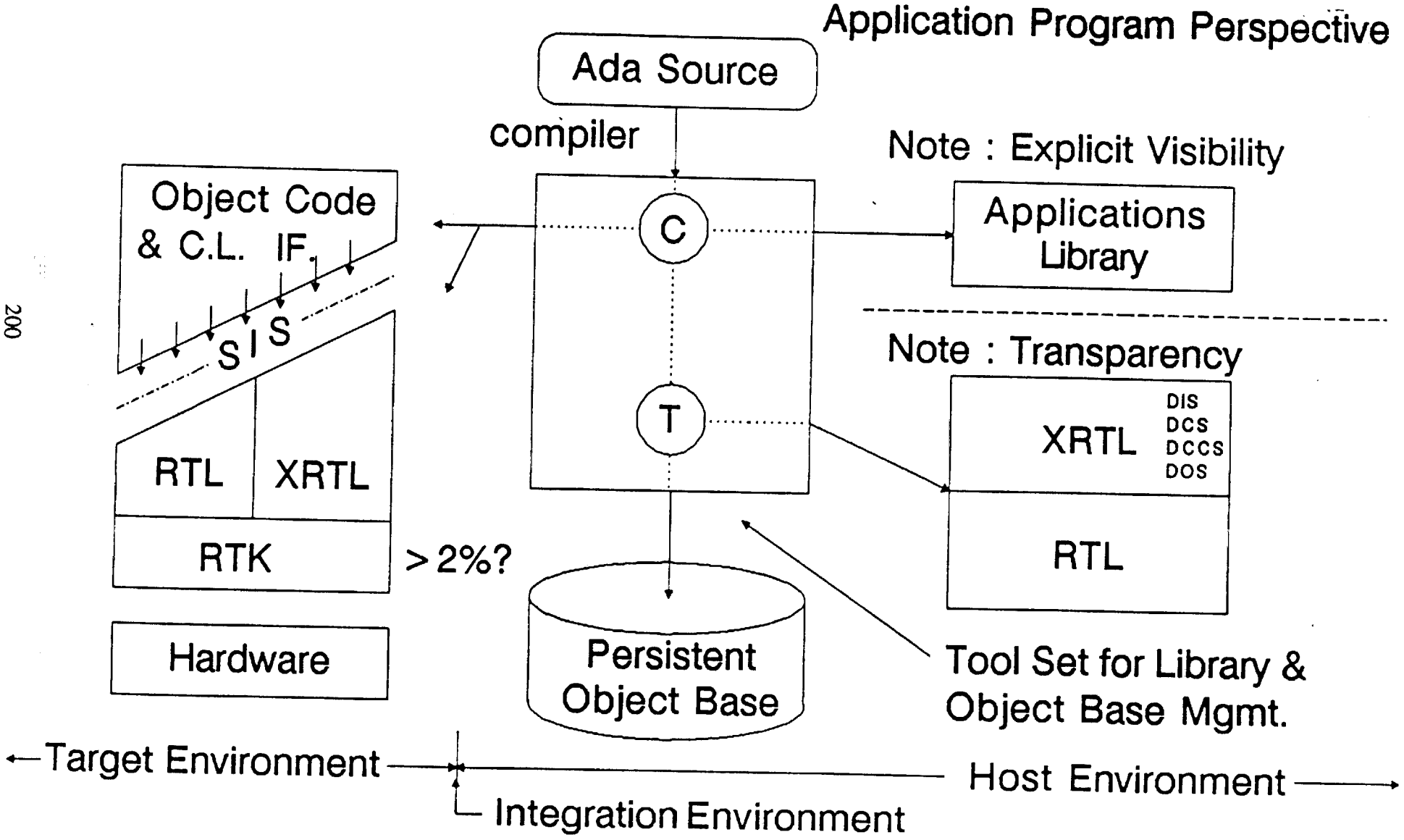


Figure 12. Implementation Model

A NEW TECHNOLOGY PERSPECTIVE & ENGINEERING TOOLS APPROACH FOR LARGE, COMPLEX & DISTRIBUTED MISSION AND SAFETY CRITICAL SYSTEMS COMPONENTS

Miguel A. Carrio
Teledyne Brown Engineering

A NEW TECHNOLOGY PERSPECTIVE & ENGINEERING TOOLS APPROACH FOR LARGE, COMPLEX & DISTRIBUTED MISSION AND SAFETY CRITICAL SYSTEMS COMPONENTS

ABSTRACT

Rapidly emerging technology and methodologies have out-paced the systems development processes' ability to use them effectively, if at all. At the same time the tools used to build systems are becoming obsolescent themselves as a consequence of the same technology lag that plagues systems development. The net result is that systems development activities have not been able to take advantage of available technology and have become equally dependent on aging and ineffective computer-aided engineering tools. New methods and tools approaches are essential if the demands of non-stop and Mission and Safety Critical (MASC) components are to be met.

INTRODUCTION

Expectedly, the systems development management and technical communities continue to remain slow and reluctant to accept change and new approaches in spite of the overwhelming evidence in support of a need for it, and the disappointing track record in systems development of the last 30 years. The resistance to change in the midst of new approaches and innovative concepts, is accompanied by a perceived threat and expectation that there is now added risk to the ever escalating cost of development by introducing new methods and tools. The risk and added cost of development, unfortunately, result in not accepting and implementing many of the recent approaches and methods available in the marketplace and continuing to ignore them.

ISSUES

Because of the following, it is strongly felt that new approaches to modeling; tools design and conceptualization in the initial life cycle phases (i.e., requirements analysis, allocation, and design synthesis) are necessary.

I. Systems have become so large that the traditional concept of prototyping is not looked favorably upon due to the large costs, expenditure of time, and complex issues raised in developing prototypes.

II. It is a given and well known fact that the earlier in the life cycle design weaknesses and errors are identified, the more cost effective the design fix and less labor intensive.

III. An inordinate amount of time and effort is expended on the software coding phase (one of the smallest life cycle cost drivers - less than 10%). Ironically, the latest industry tool craze intended to focus on life cycle productivity - CASE, continues to focus on software instead of systems. (Computer-Aided Software Engineering instead of Computer-Automated Systems Engineering)

IV. Requirements continue to change in any given systems development and necessitate a "control or harness" mechanism to provide disciplined management of them. The requirements instability,[8/] together with scarce development resources is expected to result in increased evolutionary and iterative system activities.

V. Design efforts require formal and global configuration management (i.e., across life cycle phases) that does not presently exist. If the cost of maintenance and support is to be significantly reduced, design environments and tools must support configuration management.

VI. The very large amounts of software generated in systems, require that serious consideration be given to the need for automatic code generation from a formal specification, in order to achieve any significant control and productivity gains.

VII. Hardware and software engineering, as well as their associated integration efforts continue to be treated separately for the most part. The extensive amount of "requirements implementation dumping" (RID) over the fence to software that occurs when a particular hardware requirement cannot be implemented; for example, continues to polarize these two communities.

VIII. Most design methodologies[5/] that exist today and are incorporated into the many development tools are homogeneous (e.g., either purely data-flow oriented or control-flow oriented), and system directed or application specific in nature. Homogeneity is not a negative aspect if sight is kept of the application and problem space intended for a particular methodology; and the discontinuities that result when a domain or boundary is crossed. The complex real-time issues of today amplify these discontinuities when data and control flow theory are integrated together. At best, when integration attempts are made, these homogeneous and domain specific methodologies result in a loosely coupled effort achieving an "inefficient bastardized methodology" that sells products, never intended for the solution needs of the systems they are supposed to assist.

IX. Static and dynamic analyses are attempted via uncoupled and informal design representations (i.e., using a natural language with one or more homogeneous methodologies), as opposed to using a formal design representation (i.e., using a formal syntax or design language).

X. New life cycle models and paradigms[1,2/] must be employed if quality, high confidence, non-stop MASC components are to be designed accompanied by high productivity rates.

APPROACH

The ten points identified have served as requirements drivers in molding Teledyne Brown Engineering's TAGS[R] technology aimed at solving problems early in a cost effective manner. The approach has been to:

A. Focus on the initial phases of the life cycle to ferret out design issues and capture requirements.

B. Adopt a systems engineering perspective that looks at the entire picture (i.e., views hardware and software engineering together and as driven by systems engineering).

C. View alternative life cycle models relative to the specific development activity, using automation based paradigms.

D. Employ a heterogeneous systems methodology with integrated configuration management elements.

E. Use a systems design language supported by a formal syntax.

F. Use/develop tools that support and embody items A through E.

These issues will be addressed, but as a consequence of their interrelationships, they cannot be viewed independently. Thus, for example, focusing on the early life cycle phases requires an understanding of the relationships of the various life cycle phases and activities inclusive of the maintenance and operational phases (later phases). Furthermore, the iterative life cycle forms and types must also be understood. Despite an awareness and extensive documentation of the lower costs of detecting and correcting errors in the early life cycle phases,[3/] program managers continue to ignore the early phases and resources required to support them. The question should be repeatedly asked of developers and tool builders - "Why haven't past and on-going efforts focused on the early phases to discipline and stabilize requirements and design issues?" "Why haven't the tool builders addressed the front or early life cycle phases, and provided the marketplace with extensive tools in this area?" The number of commercially available requirements generation and analysis tools that exist or can be integrated with systems engineering design synthesis tools are virtually non-existent. A basic tenet of the automation based paradigms is that early life cycle emphasis supported by automated tools is a must, if significant achievements in productivity and maintainability are to be made.

The development processes and thus modeling and prototyping efforts must be initially viewed and driven from a systems engineering perspective. The systems engineering view will insure that premature allocations of requirements to either hardware or software are deferred until the proper time and more important into the proper allocated design specification. A systems engineering view ensures that from the

very beginning, that the system to be synthesized and hierarchically decomposed is related properly to its operational environment and system interfaces. System interface identification also insures that perturbations generated across them can be properly accounted for and contained. Encouragement to invoke reusability[6] at the architecture, design, specification, algorithmic, logical and code level as early as possible in the life cycle insures maximum reuse yields. This approach is nothing more than a return to classical systems engineering supported by functional flow block analysis, hierarchical decomposition, input-output flow integrity and structured design concepts, all considered together.

Additionally, this initial top-down design, through the iterative nature established by the systems engineering approach also enables a bottoms-up view refinement that "kicks-in" as an overdrive when requirements must be revisited and reallocated for whatever the reason from hardware to software or vice-versa.

Embracing the concepts of automation based paradigms requires that a formal executable design specification and prototype be established. The design specification must also be formally configuration manageable to enable the maintenance of requirements and design. Formal maintenance of requirements and design envelopes supported by executable prototypes based on the design provides visibility into it, and assures the ability to maintain the resulting system long after it has been built. Design enhancements, technology insertion and preplanned product insertion activities are also vastly facilitated by formal maintenance of the requirements and design baselines.

The iterative nature of the different types of life cycles (e.g., spiral life cycle, technology life cycle[4]), necessitates that traditional forms (e.g., waterfall) be viewed in perspective and the new ones viewed relative to the environment factors (i.e., evolutionary, reusability, rapid prototype).

This brings us to a very important issue - a heterogeneous systems methodology with integrated configuration management. Most methodologies that have been developed over the last 30 years have been tempered by functionality, applications or domain specific solutions they were focused at to provide solutions. The MIS community has benefitted tremendously from data flow methodologies for example.

However, different methodologies and views are required to address the new solution spaces presented by non-stop and mission and safety critical components in complex systems. The ability to view simultaneously architecture, functionality, data flows, control flows and timing requires an integrated heterogeneous methodological approach that both represents and integrates these elements in a "unified field theory". Furthermore, the ability to examine, compare, and analyze these elements requires greater formalism of representation than ever before. A cohesive binding (i.e. a formal syntax) is required of the methodology representation if subsequent dynamic analyses of design and simulation, via executable design prototypes are to be performed.

Additionally, if an executable prototype and formal design representation is to be generated and analyzed, then the concept of the need for an integrated heterogeneous methodology must be further extended to capture and represent other process, logic and algorithmic elements down to formal mathematical representations. Thus, what emerges is a need to capture both the methodology and design elements via a unified and extensive systems engineering vehicle capable of representing the engineering processes in a formal representation. A sample representation of this is a systems design language, of Backus Naur Form, called the Input-Output Requirements Language (IORL).[7] It is a high order language comprised of a character set and a graphical set of pictures that support both the systems engineering processes as well as the methodology requirements to bring together the elements needed for viable design synthesis.

AVAILABLE TECHNOLOGY/NDI

TAGS technology consists of three components - a methodology and paradigm; a formal systems design language; and an environmental suite, consisting at this time of nineteen integrated tools (figure 1). The workstation hosted environment is intended to support design teams in a networked mode consisting of host, integration and target components.

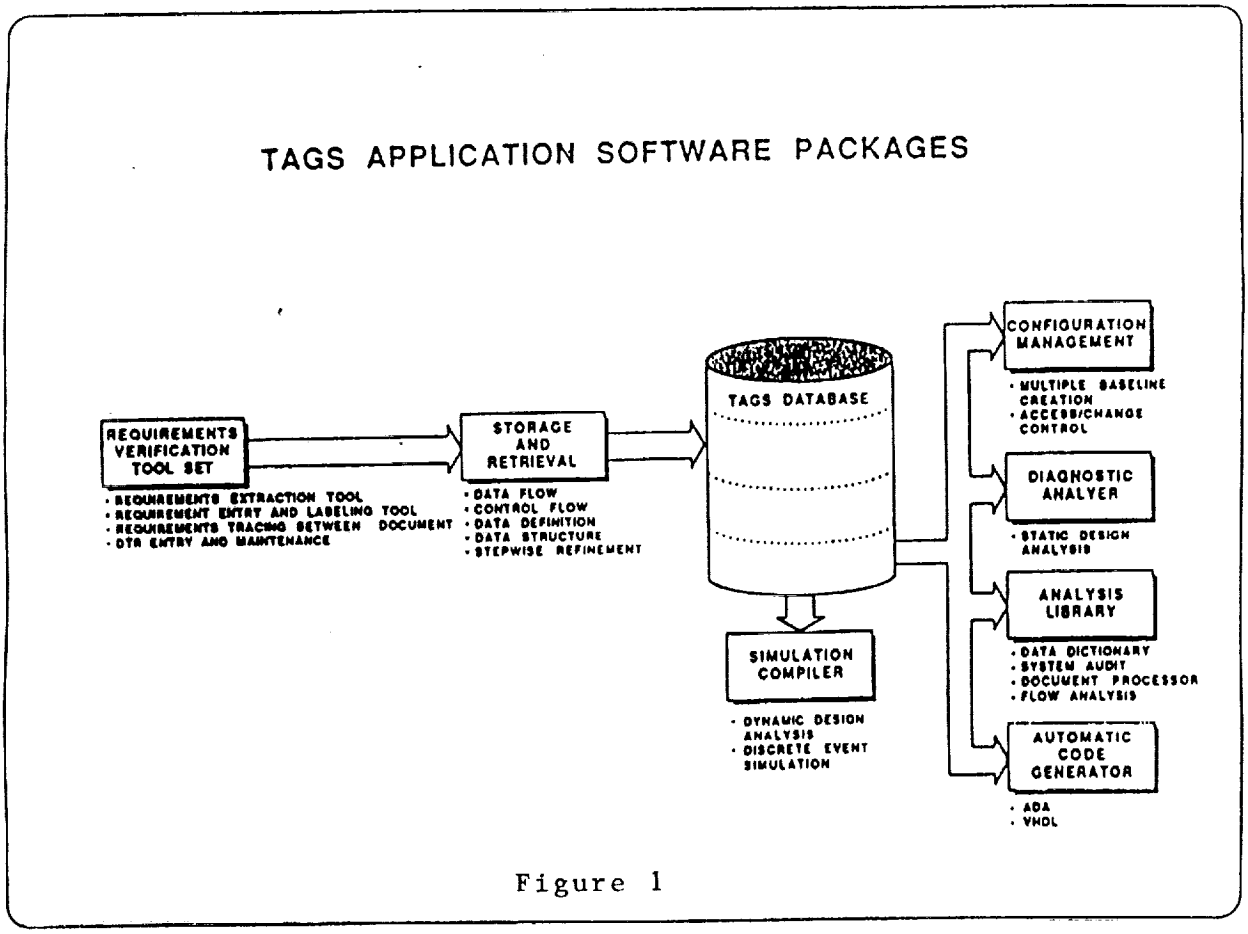


Figure 1

The technology is intended to also enable distributed design and simulation activities to occur concurrently and independently, emulating target environment characteristics and operational modes. The simulation compiler supported by merge library utilities contained in the Analysis Library enables parallel execution modes that provide insight into the target environment performance. It's two-pass simulation architecture enables the compiler to support the development host activities on lesser capacity workstations, while enabling simulation exportation to larger mainframes for execution of complex problems requiring such (e.g., strategic defense type problems). In this manner large complex and parallel simulations can be performed efficiently, requiring less execution and simulation run times, while converging on target environment solutions.

ENVIRONMENT COMPONENTS

TAGS consists of the following tools:

- A requirements verification tool set (RVTS) - to assist in the management of requirements. The four tools comprising this module enable requirements traceability within and across specifications, establish parent-child relationship between system/performance and derived requirements, allow the establishment of function, subfunction and keyword associations, provide a document trouble report capability, enable the parsing of complex english sentences containing multiple requirements references into unique requirements statements, and provide the ability to automatically generate different reports and trace matrices consistent with the evolving specification tree hierarchies.

- A storage & retrieval module (SR) - to allow the IORL page creation, storage, retrieval, modification, viewing, and updating; supported by menu-driven highly interactive features. The SR module provides access to the IORL design language and three levels of security to protect the design pages. SR is the basic design module.

- A configuration management module (CM) - provides system integrity and formal Mil-Std configuration management to be established on a single or multiple set of baselines. CM is fully integrated into the TAGS-IORL data base to insure full management and control of design regardless of system complexity and network/team size.

- A diagnostic analyzer (DA) - to provide static analyses of the IORL design pages generated via the SR module. The design is analyzed in a background mode, monitored via a separate process so as to allow a continuation of the other designer activities on a non-interference basis. Designed as

an incremental compiler, DA allows the completion of individual pages or modules that can be integrated and analyzed in a background mode with other completed design. Error messages with unique reference numbers provide for a rapid and efficient troubleshooting mode that can be supported by lesser experienced members of the design team. DA performs static analysis (i.e., completeness, consistency and closure checks on the design).

- A simulation compiler (SC) - produces an executable discrete event simulation of a system designed in IORL. The simulation performs dynamic error analyses which can locate problems such as timing and control faults; and errors that cannot be found through static testing. Also, the resulting execution trace listing assists the user in determining the optimum system and processing algorithm designs as well as performance analysis.

- An analysis library (AL) - consists of ten tools that fall into three categories:

  Auditing - to allow the user instant visibility into the design database, and also provides for system partitioning and ready identification of all system components.

  Documentation Management - to allow the automatic generation of data dictionaries, and provide a document processor interface (e.g., POSTSCRIPT). The document processor interface provides the user the ability to merge system design text and graphics with other commercial document processors (e.g., Context^{R} and Interleaf^{R}) to further extend the ability to generate automatic documentation and support such standards as Mil-Std-2167.

  Reusability - to allow the reuse of design, architecture, functionality, specifications and code. These tools greatly support the reusability development paradigms and enable the invocation of reusability concepts very early in the life cycle.

- An automatic code generator (ACG) - allowing the designer to generate executable source code from the IORL formal design representation for transition and insertion into the target environment. The code generator thus provides the capability to support "software first" development, rapid prototype paradigms and the early identification of source code that requires further optimization via a lower level software engineering environment. Currently, the only language the code generator supports is Mil-Std-1815A (Ada), with a VHSIC Hardware Description Language (VHDL), IEEE Std 1076-1987 code generator under development.

QUANTITATIVE RESULTS USING THE TECHNOLOGY

Use of TAGS technology on two DoD programs has provided both qualitative and quantitative results that are most encouraging and support predictions of achievable and significantly higher productivity.

The development approach and environment have clearly established a direction that represents: a significant higher productivity yield, life cycles with significantly reduced implementation time scales, lower development costs (figure 2), and higher design confidence and quality levels. The two BM/C^3 (Battle Management/Command Control Communications) efforts, N-SITE and SIE, went from the requirements phase through to integration, test and delivery of software end product to the customer. The costs of investment and learning the new technology are also factored in. While the total source lines of code for each of the systems numbered in the 200-300K range, similar yields equal to or greater are expected for larger systems of the type SDI or NASA would encounter. Part of the rationale for this is derived from the fact that program module size for manageability and optimization require breakdown into smaller blocks of several thousand lines of code (less than 10K LOC) independent of overall program size or use of automatic code generators.

It should be noted that these two DoD efforts did not utilize the Ada Code Generator, since FORTRAN was identified by the user as the implementation language. The simulation compiler and RVTS were not in product form when these efforts were initiated some fifteen months ago. Thus, the expectation of using the extended tool set in future efforts, provides an even firmer basis for supporting higher productivity and quality. Given the existence of a code generator and simulation compiler enables the developer to begin testing and integration earlier, deemed essential to supporting higher confidence levels. The delivered software, when used, performed satisfactorily as intended and as specified in the user/operational environment.

PROJECT A (N-SITE)
SOFTWARE DEVELOPMENT

METRIC	SOURCE		
	INDUSTRY STANDARD[1]	PROJECT EXPERIENCE	COMPARISON (%)
Lines of Code/Hour	0.77 LOC/hr[1]	2.26 LOC/hr[4]	294
Calendar Schedule	8.4 months[1,2]	5.5 months	65.5
Effort (man-months)	132 mm[1]	45 mm	34
Cost per LOC	$45 to 50[3]	$14.57	30.7
Total Cost	1M	$0.237M	23.7

1. Based on Boehm: Software Engineering Economics
2. Compressed Schedule; Normal Schedule = 11.6 months
3. Based on $90K - 100K/man-year at 2,000 hr/yr
4. Converted 3.1K LOC of Project B Monitor and Control; Reused Portions of Project B IORL® Design

PROJECT B (SIE)
SOFTWARE DEVELOPMENT COMPARISONS AT BUILD 3

METRIC	SOURCE		
	INDUSTRY STANDARD[1]	PROJECT EXPERIENCE	COMPARISON (%)
Lines of Code/Hour	0.69 LOC/hr[1]	1.17 LOC/hr	170
Calendar Schedule	12 months[1,2]	9 months	75
Effort (man-months)	316 mm[1]	188 mm	59
Cost per LOC	$45 to 50[3]	$34	72
Total Cost	$2.1M	$1.02M	49

1. Based on Boehm: Software Engineering Economics
2. Compressed Schedule; Normal Schedule = 15 months
3. Based on $90K - 100K/man-year at 2,000 hr/yr

Actual Results obtained are shown in the "Project Experience" column. The significant drop in cost per lines of code (LOC) in Project A resulted from the reuse of design and algorithms from Project B.

Figure 2

CONCLUSIONS

In the near future, resulting data from other on-going development projects is expected to provide further credibility for use of the approach, technology and environment. Invoking the approach and paradigms, with the type of environment identified will almost certainly result in higher levels of design confidence and integrity accomplishable in significantly lesser times. Early results in using the technology on other on-going efforts continue to support existing data. It is recognized that the technology and approaches will themselves continue to evolve to more mature forms. It is also recognized that with continued use, new products and enhancements will also occur in the environments either as a direct consequence of it or as a result of integrating and interfacing other commercial products with it. Furthermore, risk is less and certainly no more than that which exists in on-going developments today. And, if expectations should fall short for whatever the reason, the developer maintains his status quo, with everything to gain by using what is presently available and nothing to lose.

REFERENCES

1/ Balzer, Cheathem, Green - Computer, Vol. 16, No. 11, Nov. 83, pp. 39-45. Subject: Software Technology in the 1990's: Using a New Paradigm.

2/ Balzer, Robert - Proceedings of COMPSAC 84 Conference on Computer Software and Applications; Nov. 84, Chicago, Ill.; IEEE #0730-3157, pp. 471-475. Subject: Evolution as a New Basis for Reusability.

3/ Boehm, B. - Software Engineering Economics; 1981, Prentice-Hall, Inc.

4/ Carrio, Miguel - Proceedings of the 4th National Conference on Ada Technology; March 1986, Arlington, VA, pp. 75-81. Subject: The Technology Life Cycle and Ada.

5/ Davis, Alan M. - Communications of the ACM, Vol. 31, No. 9, Sep. 88, pp. 1098-1115. Subject: A Comparison of Techniques for the Specification of External System Behavior.

6/ Jones, G. - Proceedings of COMPSAC 84 Conference on Computer Software and Applications; Nov. 84, Chicago, Ill., Library of Congress No. 83-640060, pp. 476-478. Subject: Software Reusability: Approaches & Issues.

7/ Sievert, G. & Mizell, T. - IEEE-Computer; Vol. 18, No. 4, Apr. 85, pp. 56-65. Subject: Specification-Based Software Engineering with TAGS.

8/ Yadav, Bravoco, Chatfield, Rajkumar - Communications of the ACM, Vol. 31, No. 9, Sep. 88, pp. 1090-1097. Subject: Comparison of Analysis Techniques for Information Requirement Determination.

Alpha: A Real-Time Decentralized Operating System for Mission-Oriented System Integration and Operation

E. Douglas Jensen

Concurrent Computer Corporation

Westford, MA
508-692-6200
edj@cs.cmu.edu, uunet.uu.net!masscomp!jensen

Alpha:
A Real-Time Decentralized Operating System for Mission-Oriented System Integration and Operation

E. Douglas Jensen

Concurrent Computer Corporation
Westford, MA
508-692-6200
edj@cs.cmu.edu, uunet.uu.net!masscomp!jensen

Abstract

Alpha is a new kind of operating system, which is unique in two highly significant ways. First, it is *decentralized*, providing reliable resource management transparently across physically dispersed nodes, so that distributed applications programming can be done largely as though it were centralized. And second, it provides comprehensive, high technology support for real-time system integration and operation, an application area which consists predominately of aperiodic activities having critical time constraints such as deadlines. Alpha is extremely adaptable so that it can be easily optimized for a wide range of problem-specific functionality, performance, and cost. Alpha is the first systems effort of the Archons Project, and the prototype was created at Carnegie-Mellon University directly on modified Sun multiprocessor workstation hardware. It has been demonstrated with a real-time C^2 application written by General Dynamics Corp. Continuing research by Concurrent Computer Corp. is leading to a series of enhanced follow-ons to Alpha; these are portable but initially hosted on Concurrent's MASSCOMP line of multiprocessor products. Both the initial and the subsequent versions of Alpha are sponsored by the USAF Rome Air Development Center and are in the public domain for government use.

A *Decentralized* OS is New

Alpha is oriented towards systems having on the order of 10 to 100 nodes which are physically dispersed on the order of 1 to 100 meters (longer distances are possible). Alpha is for the most demanding kind of situation: mission-oriented systems where all nodes are contributing to the same application, not simply for the network case of individual users at each node doing unrelated computations. Our focus is on having nodes be logically integrated together rather than autonomous. Alpha provides this logical integration by executing on the bare hardware and managing resources in the same sense as a uniprocessor OS does, not by being just a "UNIX-style" user process and providing standard application interfaces and protocols for simple inter-node resource sharing like conventional computer network style distributed OSs do. Resources must often be managed by Alpha across node boundaries in the best interests of the whole application, not just on the usual per-node basis. This necessitates that Alpha also accept responsibility for handling certain fundamental asynchronous concurrency and reliability issues which arise in distributed systems, instead of passing them all up to the users for recurring, lower performance solutions. Alpha provides mechanisms which are necessary and sufficient to maintain consistency of data and correctness of operation at both the OS and application levels despite concurrent execution, and node or communication path failures, using techniques similar to those normally found far above the OS in distributed database systems—e.g., nested atomic transactions, replication. With Alpha, the nodes collectively form a single computer, not a computer network; thus, distributed application software can be written as though it were for a conventional uniprocessor—without even knowing about, much less having to manage, distributed resources.

Alpha is decentralized in another valuable and difficult sense. It does not depend on the existence of any phys-

ically or even logically centralized resource management entity or service, such as a "location broker."

"Real-Time" is Different in the System Integration and Operation Context

The term "real-time" is usually intended to mean "deterministic behavior" and "faster is better", particularly in the area of interrupt handling and context swaps. Real-time control in this sense applies only to computer systems which simply do low-level sensor/actuator sampled-data loop applications, and are traditionally designed to have rigidly periodic behavior. But real-time system integration and operation is far more difficult because it encompasses not just such static periodicity but also predominantly dynamic and aperiodic activities which nonetheless have critical time constraints, such as deadlines. These constraints are part of the correctness criteria of the computation, and failure to meet them is a threat to the systems's mission and to survival of property and human life. **Alpha** personnel invented a novel approach whereby the application's time constraints are expressed in terms of the value to the system of completing each activity as a function of its completion time (deadlines are a simple special case—a step function). In addition, activities have relative importances which are also time-dependent. These time value functions and importances are dynamic and must be continuously re-evaluated. Every evaluation is performed for all executing and pending activities collectively so as to maximize the total value to the system across the whole time period represented by the expected durations of all these activities. This sophisticated and explicit treatment of real time has been conclusively shown in both theory and practice to be exceedingly cost-effective. The conventional and seemingly simpler notions of "priority" in real-time systems are zero'th order approximations which extensive experience has consistently demonstrated introduces massive and uncontrollable complexity into all but the most trivial real-time systems. **Alpha** employs this new real-time management technique to resolve *all* contention for resources such as processor cycles, communication access, secondary storage, and synchronizers (e.g., semaphores, locks). Time constraints and importance are among the attributes propagated with computations which cross node boundaries so that resource management can be global. The ubiquitous client/server model is unsuitable in this respect since it does not maintain such essential correspondences between the service and client on whose behalf that service is being provided.

Alpha exhibits a fundamental philosophy which is contrary to that of OSs for other application environments. Instead of optimizing performance of the normal cases at the expense of infrequent ones, it does the opposite. It is in the exception cases such as emergencies (e.g., being in danger due to attack or failure) when a real-time OS must be depended upon to perform best, even if the system's routine behavior must be compromised to ensure that. This is one of the principal reasons why real-time UNIXs are inherently limited.

Of course, **Alpha** also has all the features usually sought in real-time operating systems, including a fully preemptable kernel, synchronization, asynchronous notification, i/o directly to/from user space, contiguous files on disk, memory-locked objects, pre-allocatable resource pools, low interrupt latency and services times, etc.

Extraordinary "Adaptability" is Essential to Real-Time System Integration and Operation

Real-time system integration and operation applications are very complex, and are not (perhaps cannot be) well understood; in addition, the environment and technology are always in a state of flux. Thus, the functional and performance requirements for their computers evolve continuously throughout the entire life cycle of the system, which can be decades. **Alpha** accommodates this situation through a variety of techniques, many of which are quite innovative. Its design is kernelized and strictly adheres to the principle of policy/mechanism separation. Specific OS policies are carefully excluded from its kernel level mechanisms so that a wide range of different service facilities, and indeed entire DOSs, can be effectively constructed using **Alpha**'s kernel, in accordance with application needs. For example, **Alpha**'s kernel provides atomicity, serializability, and permanence as orthogonal mechanisms. Conventional atomic transaction facilities bundle all three properties together, with correspondingly high overhead, as the only choice of policy regardless of need and affordability. But the client layers of **Alpha**'s kernel can base their policies on other combinations of these mechanisms. For example, there are many instances in real-time systems when problem-specific consistency constraints yield correct results more efficiently than serializability would, or when permanence is not worth its cost. This same philosophy is followed in scheduling, communications, and all other types of

resource management.

Computers embedded in real-time systems usually must produce the highest possible performance from the allowable hardware size, weight, and power, including memory space for the OS. A general-purpose computer system can easily be an order of magnitude lower performance than a special-purpose one for a particular application. Thus, to achieve the balance of performance and flexibility needed for cost-effectiveness in a multiplicity of changing system integration and operation applications, **Alpha** is general-purpose but unusually malleable so as to exploit all the problem-specific static and dynamic information available from the application. In addition, application functionality can readily be migrated downward into the OS, and even into its kernel, for increased performance when necessary.

Alpha's internals are organized so that its subsystems such as scheduling, communications, secondary storage, etc. can all execute truly concurrently at each node. We intend that these separate hardware points of control within **Alpha** are a mixture of dynamically assigned general-purpose processors (i.e., each node in the decentralized computer can be a multiprocessor) and algorithmically specialized hardware accelerators (co-processors and other forms of augmentation). **Alpha** extends to its client applications the same opportunities for taking advantage of multiple special-purpose as well as general-purpose processors at each node.

Alpha presents a programming model which is object oriented, in the sense of abstract data types. This imposes a structure and discipline conducive to modular software at both the DOS and application levels, as well as improving fault isolation. The active entity, or unit of logical computation, is a *thread* stringing through objects via operation invocation, without regard for address spaces or node boundaries; fundamental distribution and reliability issues are the responsibility of **Alpha** instead of the user. This network uniformity and transparency greatly aids the creation and modification of distributed applications.

Status

Alpha Release 1 (done at CMU) has been demonstrated to DoD agencies since late 1987 with a real-time C^2 application written by General Dynamics Corporation. Concurrent Computer Corporation is creating Releases 2 and 3 of **Alpha** which are significantly enhanced and commercial quality; these will be available for experimental use on their multiprocessor products by the Fall of 1989 and 1990, respectively. Alpha is an open operating system in the sense of being both intended and designed for portability to multiple vendors' hardware, and has begun to emerge as the *de facto* standard for next-generation mission-oriented real-time operating systems.

Acknowledgment

Alpha is funded jointly by the USAF Rome Air Development Center and Concurrent Computer Corporation, with additional support from General Dynamics Corporation and others.

References

Northcutt, J. D., Clark, R. K., Shipman, S. E., Maynard, D. P., Lindsay, D. C., Jensen, E. D., Smith, J. M., Kegley, R. B., Keleher and Zimmerman, B. A.
Alpha Preview: A Briefing and Technology Demonstration for DoD.
Archons Project Technical Report #88031, Department of Computer Science, Carnegie-Mellon University, March, 1988.

Jensen, E. D., Northcutt, J. D., Clark, R. K., Shipman, S. E., Maynard, D. P. and Lindsay, D.C.
The Alpha Operating System: An Overview.
Archons Project Technical Report #88121, Department of Computer Science, Carnegie-Mellon University, December 1988.

Northcutt, J. D.
The Alpha Operating System: Requirements and Rationale

Archons Project Technical Report #88011, Department of Computer Science, Carnegie-Mellon University, January, 1988.

Northcutt, J. D. and Clark, R. K.
The Alpha Operating System: Programming Model.
Archons Project Technical Report #88021, Department of Computer Science, Carnegie-Mellon University, February, 1988.

Northcutt, J. D., Clark, R. K., Shipman, S. E. and Lindsay, D. C.
The Alpha Operating System: System/Subsystem Specification.
Archons Project Technical Report #88122, Department of Computer Science, Carnegie-Mellon University, December 1988.

Northcutt, J. D.
The Alpha Operating System: Kernel Programmer's Interface Manual.
Archons Project Technical Report #88111, Department of Computer Science, Carnegie-Mellon University, November 1988.

Trull, J. E., Northcutt, J. D., Clark, R. K., Shipman, S. E. and Lindsay, D. C.
An Evaluation of Alpha Real-Time Scheduling Policies.
Archons Project Technical Report #88123, Department of Computer Science, Carnegie-Mellon University, December 1988.

Clark, R. K., Kegley, R. B., Keleher, P. J., Maynard, D. P., Northcutt, J. D., Shipman, S. E. and Zimmerman, B. A.
An Example Real-Time Command and Control Application on Alpha.
Archons Project Technical Report #88032, Department of Computer Science, Carnegie-Mellon University, March, 1988.

Northcutt, J. D. and Shipman, S. E.
The Alpha Operating System: Program Maintenance Manual.
Archons Project Technical Report #88123, Department of Computer Science, Carnegie-Mellon University, December 1988.

Northcutt, J. D. and Shipman, S. E.
The Alpha Operating System: Programming Utilities.
Archons Project Technical Report #88041, Department of Computer Science, Carnegie-Mellon University, April, 1988.

Northcutt, J. D.
The Alpha Distributed Computer System Testbed.
Archons Project Technical Report #88033, Department of Computer Science, Carnegie-Mellon University, March, 1988.

Northcutt, J.D.
Mechanisms for Reliable, Distributed Real-Time Operating Systems: The Alpha Kernel.
Academic Press, 1987.

*U.S. GOVERNMENT PRINTING OFFICE:1988-661-008/80687