AD-A273 676

NSWCDD/MP-93/172

PROCEEDINGS OF THE 1993 COMPLEX SYSTEMS ENGINEERING SYNTHESIS AND ASSESSMENT TECHNOLOGY WORKSHOP (CSESAW '93) (20-22 JULY 1993)

STEVEN HOWELL, COORDINATOR

SYSTEMS RESEARCH AND TECHNOLOGY DEPARTMENT

DTIC ELECTE DEC 13 1993 S B D

17 OCTOBER 1993

Approved for public release; distribution is unlimited.

93-30135

NAVAL SURFACE WARFARE CENTER
DAHLGREN DIVISION • WHITE OAK DETACHMENT
Silver Spring, Maryland 20903-5640

93 12 10039

Best
Available
Copy

NSWCDD/MP-93/172

PROCEEDINGS OF THE 1993 COMPLEX SYSTEMS ENGINEERING SYNTHESIS AND ASSESSMENT TECHNOLOGY WORKSHOP (CSESAW '93) (20-22 JULY 1993)

STEVEN HOWELL, COORDINATOR
SYSTEMS RESEARCH AND TECHNOLOGY DEPARTMENT

17 OCTOBER 1993

Approved for public release; distribution is unlimited.

NAVAL SURFACE WARFARE CENTER
DAHLGREN DIVISION • WHITE OAK DETACHMENT
Silver Spring, Maryland 20903-5640

1993 COMPLEX SYSTEMS ENGINEERING SYNTHESIS AND ASSESSMENT TECHNOLOGY WORKSHOP (CSESAW '93)

As technology has developed, computer-intensive systems have increasingly become extremely large and complex, controlling a wide variety of resources and operating in many unforeseeable situations. Many of today's systems have hard real-time, stringent dependability, intensive security, and demanding cost of ownership requirements. They are typically implemented on a combination of parallel and distributed architectures and are embedded within a human organizational structure and/or have human operators in the loop.

This is the third year of the CSESAW (pronounced see-saw) workshop. This year the workshop is co-sponsored by Office of Naval Research, Naval Surface Warfare Center Dahlgren Division, and Advanced Technology and Research (ATR). The workshop was created to explore system level design synthesis and assessment capabilities for large, complex systems. These capabilities will facilitate the development of such systems from informal system requirements, through the design phase prototyping, and into implementation and post deployment. Component products produced by these capabilities are specifications that subenvironments, e.g., Hardware Engineering Environment (HWEE), Software Engineering Environment (SEE) and Human Computer Interaction Engineering Environment (HCIEE), will receive. The focus of this workshop is the development and integration of these multiple technologies and the exploration of the creation of a system level engineering discipline with support technologies to provide potential high payoff solutions to the difficult problems encountered by designers, developers, and maintainers of real-time systems. The emphasis is on resolving system level technology issues that cut across component boundaries, such as those associated with system behavior requirements of real-time, fault tolerance, cost, and security.

The emphasis on this year's workshop is integration. The technologies and capabilities need to be integrated with the rest of the engineering process. Therefore, the capability to provide tight linkages to detailed design evaluation, systems forward engineering and systems reengineering must be developed, ultimately providing a seamless overall engineering process. A significant amount of effort has been put into component technologies, such as hardware, microelectronics, memory, databases, software, man-machine interface, etc. Major strides have been made in these areas in the last few years. However, the formal, systematic integration and engineering of these components into an overall system has lagged far behind. For large and complex systems with real-time, cost, dependability and

security requirements, the problem is especially acute. This is a direct result of a lack of a system level engineering methodology.

We welcome you to this year's workshop. We hope to continue to provide in the workshop an atmosphere in which the participants, including technology developers, researchers, users and customers can meet, interact and exchange ideas on relevant issues. In the near future we hope to be able to say that this workshop was the beginning of a new focus on systems design and evaluation technologies.

This workshop would not have been possible without the hard work of many people, including the workshop, program, and advisory committees, authors, presenters of the submitted papers, panel members, workshop attendants, panel chairs, and breakout session chairs. A very warm "Thank You" is extended to all. In particular, we wish to acknowledge Michael Edwards, Ngocdung Hoang, Cuong Nguyen, Michael Jenkins, Chuck Sadek, Kathy Lederer, Adrien Meskin, and Dong Choi. A particular thanks goes to Elizabeth E. Wald and CDR. Gracie Thompson, of the Office of Naval Research for tirelessly working for and supporting the technology developments in this important area. Finally, we would like to give a special thanks to Phillip Q. Hwang, who has chaired the workshop for the past two years, and whose insight and foresight has made the workshop possible.

We hope you have a productive and enjoyable workshop!

Steven L. Howell
Workshop General Chairman

William Farr
Workshop Assistant Chairman

1993 COMPLEX SYSTEMS ENGINEERING SYNTHESIS AND ASSESSMENT TECHNOLOGY WORKSHOP (CSESAW '93)

CSESAW '93 is exploring system level design synthesis and assessment capabilities for large, complex systems. These capabilities will facilitate the development of such systems from informal system requirements, through the design phase prototyping, and into implementation and post deployment. Component products produced by these capabilities are specifications that subenvironments will receive. The focus of the workshop is the development and integration of these multiple technologies and the exploration of the creation of a system level engineering discipline with support technologies to provide potential high payoff solutions to the difficult problems encountered by designers, developers, and maintainers of real-time systems. The emphasis on this year's workshop is integration. To be effective, technologies and capabilities developed need to be integrated with the rest of the engineering process. Therefore, the ability to provide tight linkages to detailed design evaluation, systems forward engineering and systems reengineering must be developed, ultimately providing a seamless overall engineering process. The workshop explores technology issues in providing this seamless process.

DTIC QUALITY INSPECTED 3

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

1993 COMPLEX SYSTEMS ENGINEERING SYNTHESIS AND ASSESSMENT TECHNOLOGY WORKSHOP (CSESAW '93)

July 20-22, 1993

Holiday Inn
4095 Powder Mill Road
Beltsville, Maryland 20705

Tuesday, 20 July 1993

0730 Registration

0900 *Workshop Overview*
Steve Howell, Naval Surface Warfare Center Dahlgren Detachment

0930 *Automating the System Engineering Process*
John Rumbut, Naval Undersea Warfare Center

0945 *Requirements Metrics: The Basis of Informed Requirements Engineering Management*
Robert J. Halligan, Technology Australasia Pty Ltd

1000 **COFFEE**

1015 *Engineering and Analysis of Real-Time Systems*
Jay K. Strosnider, Carnegie Mellon University

1045 *On the Structure and Dynamics of a Deeply-Integrated Information System*
Bruce I. Blum, Johns Hopkins University/Applied Physics Laboratory

1115 *Integration Components, Spaces, and Cells*
Jeffrey O. Grady, General Dynamics

1130 *An Efficient Approach to Systems Evolution (EASE)*
Thomas C. Choinski, Naval Undersea Warfare Center

1200 **LUNCH**

1300 *An Overview of the Processing Graph Support Environment*
Roger Hillson, Naval Research Laboratory

1330 *A Real-Time Object Model: A Step Toward an Integrated Methodology for Engineering of Complex Dependable Systems*
Kane Kim, University of California, Irvine

1400 *A Methodology for Complex Computer Systems Engineering*
Robert L. Harrison, Naval Surface Warfare Center Dahlgren Division

1430 *An Assessment Control Board (ACB) and a System Integration (SI) Program as Complements to the Configuration Control Board (CCB)*
Richard Evans, George Mason University

1445 *A Generic Object Oriented Conceptual Pivot Model*
Naoufel Kraiem, University of Paris I

1515 *The System Engineering Technology Interface Specification (SETIS): An Update*
Evan Lock, Computer Command and Control Company

1545 **COFFEE**

1600 *RESEARCH AND TECHNOLOGY VISION PANEL*
Chair: Phillip Q. Hwang

Wednesday, 21 July 1993

0800 *The Representation of Resources for Large-Sized and Complex Systems*
Nicholas Karangelen, Trident Systems Inc.

0830 *A Software Metrics Integration Framework*
William M. Evanco, MITRE Corporation

0900 *Measurement and Evaluation of Complex Navy System Designs*
Osman Balci, Virginia Polytechnic Institute and State University

0930 *Optimal Selection of Failure Data for Predicting Failure Counts*
Norman F. Schneidewind, Naval Postgraduate School

1000 *Design Structuring for System Engineering*
Jee-In Kim, Computer Command and Control Company

1015 **COFFEE**

1030 *COMPUTER SECURITY TRADE-OFF PANEL*
CHAIR: Kathy Meadows

1200 **LUNCH**

1300 *A Platform for Complex Real-Time Applications*
Alexander D. Stoyenko, New Jersey Institute of Technology

1330 *A Testbed for Prototyping Distributed and Fault-Tolerant Protocols*
Farnam Jahanian, IBM T. J. Watson Research Center

1400 *Architectural Synthesis of Mission-Critical Computing Systems*
Parameswaran Ramanathan, University of Wisconsin-Madison

1430 *An Intelligent Real-Time System Assessment Tool*
Ed Andert Jr., Conceptual Software Systems Inc.

1500 *An Environment for Analysis of Parallel Systems (EAPS)*
Mohsen Pazirandeh, Innovative Research Inc.

1530 *A Dependable System Perspective*
Michelle Hugue, Allied-Signal Aerospace Company

1545 **COFFEE**

1600 *COMPUTER BASED SYSTEM ENGINEERING (CBSE) ISSUES AND DIRECTIONS PANEL*
Chair: Dave Oliver

Thursday, 22 July 1993

0800 *Real-Time Databases for Complex Embedded Systems: Predictability and Serializability*
Kwei-Jay Lin, University of Illinois

0830 *Divide and Conquer Strategies and Underlying Lossless Principles*
Harold Szu, Naval Surface Warfare Center Dahlgren Division

0845 *A Fault Injection Simulation Testbed for Analyzing Fault Tolerance Protocols*
William F. Dudzik, Advanced System Technologies Inc.

0900 *Effectively Using the UNIX Make Utility for Permanent and Temporary Changes*
David H. Jennings, Naval Surface Warfare Center Dahlgren Division

0915 *Utilization Bounds for Tasksets with Known Periods*
Swaminathan Natarajan, Texas A&M University

0930 *A Stochastic Control Approach to Combined Task-Message Scheduling in Distributed Real-Time Systems*
Dar T. Peng, Allied-Signal Aerospace Co.

1000 **COFFEE**

1030 *SYSTEM INTEGRATION PANEL*
Chair: Evan Lock

1200 **LUNCH**

1300 *Comparing Formal Approaches for Specifying and Verifying Real-Time Systems*
Ralph Jeffords, Naval Research Laboratory

1335 *Advanced Integrated Requirements Engineering System (AIRES): Processing of Natural Language Requirements Statements*
Richard Evans, George Mason University

1345 *Requirements Management/Requirements Engineering (RM/RE)*
Luke Campbell, Naval Air Warfare Center

1415 *Computer Security, Safety and Resilience Requirements as Part of Requirement Engineering*
Daniel Mostert, Rand Afrikaans University

1445 **COFFEE**

1500 *REQUIREMENTS AND TRACEABILITY PANEL*
Chair: Stephanie White

Page

DESIGN STRUCTURE

Automating the System Engineering Process 2
John Rumbut--Naval Undersea Warfare Center

Requirements Metrics: The Basis of Informed Requirements Engineering Management 9
Robert J. Halligan--Technology Australasia Pty Limited

Engineering and Analysis of Real-Time Systems 15
Jay K. Strosnider--Carnegie Mellon University

On the Structure and Dynamics of a Deeply-Integrated Information System 29
Bruce I. Blum--Johns Hopkins University/Applied Physics Laboratory

Integration Components, Spaces, and Cells 38
Jeffrey O. Grady--General Dynamics Space Systems Division

An Efficient Approach to Systems Evolution (EASE) 43
Thomas C. Choinski, John G. DePrimo--Naval Undersea Warfare Center

An Overview of the Processing Graph Support Environment . . . 49
Roger Hillson--Naval Research Laboratory

A Real-Time Object Model: A Step Toward an Integrated Methodology for Engineering of Complex Dependable Systems . . 56
Kane Kim, L. F. Bacellar--University of California, Irvine

A Methodology for Complex Computer Systems Engineering . . . 65
Alexander D. Stoyenko, Lonnie R. Welch--New Jersey Institute of Technology; Robert L. Harrison, Harry Crisp--Naval Surface Warfare Center Dahlgren Division

An Assessment Control Board (ACB) and a System Integration (SI) Program as Complements to the Configuration Control Board (CCB) 74
Richard Evans--George Mason University

A Generic Object Oriented Conceptual Pivot Model 81
Naoufel Kraiem--University of Paris I

The System Engineering Technology Interface Specification (SETIS): An Update 94
Baba Prasad, Moon Lee, Rajesh Puroshothaman, Evan Lock--Computer Command and Control Company

Page

REPRESENTATION AND MEASUREMENT

The Representation of Resources for Large-Sized and Complex Systems 107
Nicholas Karangelen, John Intintolo—Trident Systems Inc.; Ngocdung Hoang, Steve Howell--Naval Surface Warfare Center Dahlgren Division

A Software Metrics Integration Framework 112
William M. Evanco--MITRE Corporation

Measurement and Evaluation of Complex Navy System Designs 126
Osman Balci, David DeVaux, Richard E. Nance--Virginia Polytechnic Institute and State University

Optimal Selection of Failure Data for Predicting Failure Counts 141
Norman F. Schneidewind--Naval Postgraduate School

Design Structuring for System Engineering 158
Jee-In Kim, Evan Lock--Computer Command and Control Company

ASSESSMENT

A Platform for Complex Real-Time Applications 172
Alexander D. Stoyenko, Lonnie R. Welch, Carlos Amaro, Bo-Chao Cheng, Matthew Harelick, Xue Jin, A. K. Ganesh, Gray Yu--New Jersey Institute of Technology; Phillip Laplante--Fairleigh Dickinson University; Thomas J. Marlowe--Seton Hall University

A Testbed for Prototyping Distributed and Fault-Tolerant Protocols 179
Farnam Jahanian--IBM T. J. Watson Research Center; Ragunathan Rajkumar--Carnegie Mellon University; John J. Turek--IBM T. J. Watson Research Center

Architectural Synthesis of Mission-Critical Computing Systems 185
Raed Alqadi, Parameswaran Ramanathan--University of Wisconsin-Madison

An Intelligent Real-Time System Assessment Tool 193
Ed Andert, Jr.--Conceptual Software Systems, Inc.; Larry Peters--Software Consultants International Ltd.

An Environment for Analysis of Parallel Systems (EAPS) . . . 198
Mohsen Pazirandeh--Innovative Research Inc.; Oliver McBryan--University of Colorado

A Dependable System Perspective 207
M. M. Hugue, N. Suri, C.J. Walter--Allied-Signal Aerospace Company

Page

IMPLEMENTATION TECHNOLOGY

Real-Time Databases for Complex Embedded Systems: Predictability and Serializability 215
Kwei-Jay Lin--University of Illinois: Sang H. Son--University of Virginia

Divide and Conquer Strategies and Underlying Lossless Principles 235
Harold Szu, Edgar Cohen, John Wingate--Naval Surface Warfare Center Dahlgren Division

A Fault Injection Simulation Testbed for Analyzing Fault Tolerance Protocols 249
William F. Dudzik--Advanced System Technologies, Inc.

Effectively Using the UNIX Make Utility for Permanent and Temporary Changes 257
David H. Jennings, John J. Reilly--Naval Surface Warfare Center Dahlgren Division

Utilization Bounds for Tasksets with Known Periods 265
Dong-Won Park, Swaminathan Natarajan, Arkady Kanevsky--Texas A&M University

A Stochastic Control Approach to Combined Task-Message Scheduling in Distributed Real-Time Systems 273
Dar T. Peng, Kang G. Shin--The University of Michigan

REQUIREMENTS

Comparing Formal Approaches for Specifying and Verifying Real-Time Systems 300
C.L. Heitmeyer, R.D. Jeffords, B.G. Labaw--Naval Research Laboratory

Advanced Integrated Requirements Engineering System (AIRES): Processing of Natural Language Requirements Statements 309
James D. Palmer, Richard Evans--George Mason University

Requirements Management/Requirements Engineering (RM/RE) 316
Luke Campbell--Naval Air Warfare Center-Aircraft Division, PAX

Computer Security, Safety and Resilience Requirements as Part of Requirement Engineering 324
DNJ Mostert, SH von Solms--Rand Afrikaans University

Page

PANEL PAPERS

Distributed Design of Computer-Based Systems: Traceability 364
Stephanie White--Grumman Corporate Research Center

Distributed Design of Computer-Based Systems: Needed Academic Programs 366
Julian Holtzman--CECASE/University of Kansas

Distributed Design of Computer-Based Systems: Methodology . . 368
David W. Oliver--GE Corporate Research and Development

Distributed Design of Computer-Based System 370
David G. Owens--Paramax Systems

Panel Description: Computer Security Tradeoffs 373
Catherine Meadow--Naval Research Laboratory

Appendix A--List of Panels A-1
Appendix B--List of Attendees B-1

DISTRIBUTION . . . . . (1)

DESIGN STRUCTURE

Automating the System Engineering Process

John Rumbut
Naval Undersea Warfare Center
Architecture and Computer System Division
Architecture and System Development Branch
Newport, RI 02841
(401)841-3616
rumbut@ada.npt.nuwc.navy.mil

Abstract

Navy system development is a participative process carried out by two primary groups: customers/users and producers/builders. These two groups communicate amongst themselves in a terminology and within a framework that is specific to their respective domains. Currently, the main bridge of communication between the two domains exists primarily in the form of natural language (specifically text and notional drawings). With "small" problems this type of communication is adequate. However, with "larger" complex systems the problem of communication between the groups is exacerbated. This paper begins to outline a generic framework for system engineering but focuses on the needs for Navy Combat Systems development.

Introduction

Analysis is the systematic process of reasoning about a problem and its constituent parts to understand what is needed or what must be done. Analysis thus involves communicating with many people. Initially those who are most familiar with the existing need and its surroundings, that is, the problem domain must be contacted. Developers will also need to communicate with the users, managers, and maintainers because they are all potential sources of new requirements. A method is needed to achieve a common understanding of the problem domain. This will allow for both the user of the proposed systems and the developer to have means to ensure that they are understanding each other during the development process.

There are few accepted standards for process tools and there are no guidelines pointing out an efficient usage pattern. A metric with which the effects or influences of any tool or method can be predetermined does not exist. There is currently no individual tool or method that is appropriate or suitable to every organization or problem domain, nor is there a method in which to predict the future methodologies, tools, or standards which will eventually emerge. The mythical silver bullet still remains elusive.

Automation will have an important role in developing solutions for large system development. However we must determine a desirable method to interface with information. Ideally the problem can be simplified to an extremely large database consisting of problem domain information with links (multiple frontends) into the database. The links are determined by the types of information being entered, queried, etc. With large system development many different specialists will be involved and each will have different requirements as to what type of information is required for them to perform their job correctly. This will require different views ('linkages') of the problem domain data.

The maintenance of these linkages, both in a horizontal (across the problem) and vertical (detailed) direction, is an ideal application of automation. However, this large amount of data causes what has been referred to as the information glut. So much data from many different areas needs to be effectively managed. Concepts such as information filtering, information retrieval and collaborative filtering will play a key role in our managing of this information. [5,6,7] These are similar problems that fall under the category of library science.

A framework is needed that will allow for easier bi-directional transition of information between the customer and the system designer. Currently, development life-cycles generate a confrontational environment with each side attempting to interpret documents in a way that will best represent their organization. The success or failure of the product depends upon high quality information exchange.

Some Simple Business Models Observation

System development projects consist of three basic types. The first is for a system built by a user for a user. Most of us have done this type of application development everywhere from building up a spread sheet macro for our checkbook to building a prototype system to go onboard a military platform for evaluation. Communication between the application domain and development is obviously very high. This type of relationship is what we hope for in all our systems, how realistic this is, is questionable.

The second type consists of a developer building a product for a perceived user. This is the type of system developed by companies like Microsoft, Symantec, Borland and other commercial venders. They perform market research and determine what new product will sell.

The third type consists of a user describing a need to a developer who then goes off and builds the new system. This type of system can be seen in the types described above if the systems are large enough to be built by others. These groups though can still belong to the same organization/agency but because of the diversity of the organization may not have a complete understanding of the problem domain.

As we examine each of these models we can see we go from a very high fidelity communication model (user to user) to a very difficult relationship in the user to developer model. Navy systems mostly fall within the third category. Communication between the developer and the customer is filled with noise which hampers information exchange. There exist several different points of view within this type of development. Managing this form of communication is very difficult.

The information that will be provided from the problem domain will be in terms of that domain. Whether this information comes from documents, questionnaires, demonstrations of existing systems or tutorials provided by the customer to the developer, a great amount of time, energy and training is needed for the designer to become informed enough to generate a proposal. Usually the developer needs to learn a new vocabulary, discipline or method. Even if the developer has worked with the problem domain before, chances are the new system reflects changes of technology within the domain, so new (potentially radically different) functional behavior will have to be learned.

Information exchange occurs again when the designer submits his proposed design back to the customer; this time the customer needs to be trained. Most software engineering firms are utilizing various CASE (Computer Aided System Engineering) tools as part of their specification/analysis effort. These tools will support the designer by assisting in developing graphical and textual information for the specification. An assumption is made that the customer will easily understand the resulting documentation. This however, is not always true.

Commercial informal graphical models (CASE tools) have yet to be proven to be overly effective in system development. Systems like the NASA space shuttle system which did not use any CASE tools was very successful in their SEI (Software Engineering Institute) process development review and they only used 'pencil and paper' to manage their development process. This leaves us with questioning the usefulness of the informal notation. Are the informal graphical models useful? Or is it more important to provide a disciplined structure to development?

Automated tools are commonly used to help improve communication and system understanding. Software engineering organizations today see selection and use of software development and support tools as crucial in improving personal productivity and product quality. Each organization is different; specific problem domains may be their primary business, which can require a certain hardware device and/or programming language. Therefore, each organization will have specific expectations and requirements to be addressed by automation.

Often the customer has sent out requests for proposals for a new system to different competing firms. This is done with the hope of keeping the costs down and improving the quality of the product by stimulating competition. However, since each of these competing firms may make use of different tools and methodologies it will become very difficult for the customer to evaluate the submitted proposals adequately. This same type of problem can easily occur in large projects where numerous subcontractors (or divisions in large companies) are responsible for different subsystems. Each may have already made a significant investment in a methodology/representation tool. How does one convert, without losing information, between these different tools? Is there some form of mapping from one to the other? Some standard is required. CASE

tool research has focused on standardizing on the use of a graphical representation.

Before the designer became proficient with the model used by the CASE tool, a significant amount of training was required. The designer went to training sessions, used the technique to develop small scale applications (threw the initial attempts away and then tried again) and worked with other team members who had experience with using the tool/model. The customer will need similar skills and experience in order to interpret the design documentation correctly. The customer may not have to be an expert in using the tool but the customer must be able to interpret and understand the model representation of the decomposition.

Unfortunately, this expertise rarely exists on the side of the customer. This means that the participants in this process are often divided between those who understand abstraction and formal terminology of system/software development and those who do not. For the latter, formal terminology is an unacceptable method of determining system feasibility. However, without formalism, the specification of a system cannot be a basis for development or analysis. If a specification is vague then the entire development process will be serendipitous. This will cause cost overruns, increased length of project development time, and a possible lack in desired system capabilities.

As mentioned earlier how will competing firms be evaluated by customers if each is using different notations? The problem domain specialists are not necessarily system engineering notation specialists and do not have the inclination nor the time to learn numerous notations (and become expert) in order to evaluate proposals. This appears to provide a strong disconnect between developer and customer. A simple solution to this is for the customer to standardize on a set of notations and force all bidders to use this set of notations. Is there a universal set of notations that will satisfy all types of Navy applications? Do we need a method of categorizing problems and using this information to best select a development model?

Methodologies and supporting tools must be carefully matched to the adopting body in order to facilitate the development process. The choice of the wrong tools can not only fail to improve the process, but can actually work against it. A universal set of rules has not been devised to aid users in selecting the 'optimum' set of tools, nor does a set of 'generic' tools currently exist that will satisfy the needs of all organizations and problem domains. This is not unusual; universal models of any sort are hard to come by and are often never completely accepted.

Process and the System Engineering Database

When a need has been determined for a new Navy application, feasibility studies will be performed to determine whether the system needs to be built. If the need is strong enough a specification of what is required will be developed. Lets call this Capture Point 1 (CP1). Some of the questions we need to answer are:

- What determines the success/failure of our feasibility study?
- How do we effectively estimate costs?
- Can we estimate technology shortfalls?
- Is this similar to something we have done before?
- Does or can reuse play a role?

Assuming that a need for a new system was determined, the Navy will generate a specification on what is needed to be built. The Navy will have responsibility for generating this document that will be delivered to competing contractors who may wish to bid on the work for this new system. We'll call this Capture Point 2 (CP2). Some of the questions at this point are:

- What are the information needs of those who must start this process?
- How do we capture information from the feasibility studies?
- What form should this data be in for the bidder to adequately bid upon this new system?
- What is the metric for determining completeness of the specification?

The developers involved will generate a proposal back to the Navy for evaluation. This is Capture Point 3 (CP3). Again we have some questions we want to answer at this point:

- In what format (graphical, text and/or prototype) should it arrive back for evaluation?
- How do we ensure completeness in the proposal?
- Can we automatically rate portions of the proposal?
- Are the cost estimates accurate?

Is the technological approach correct?

The last capture point is the development process itself. This embodies all of the work of the developer and how we manage this process activity. This is capture point four (CP4). We will not cover this part in this paper.

The system engineering framework needs to unify all of these capture points into a single framework, no matter in what format they are currently stored. The framework must have the ability to collect data from a user community with diverse functional interests. This information has to first be evaluated, validated and *tied* together in a cohesive package. At some point this package will be judged ready to be available for others to review and submit bids for the work of product development. Packages will be returned to the Navy in a format that is 'tied' to the original specification generated by the Navy. The proposal package from each developer is accepted into the system and then these proposals packages are evaluated against each other and the original specification to determine a final candidate. The framework supports bringing the problem to a solution space and then managing the system throughout its lifetime. The framework should also support supplying information (for reuse) to other projects to be built in the future.

A key element of large scale system development is understanding other participant's perspective. By increasing the amount of communication to all participants a better appreciation for other's needs should emerge. We can achieve this by collecting information from these sources provide retrieval, filtering and analysis of this information. We have already started much of this. What we have not done is effectively 'tied' all of these sources together.

As the data is entered into the database it will need to be filtered. The data will need to be entered in a manner that allows for retrieval in a variety of ways. First we need to handle the data base horizontally. What this means is we want to handle the individual sources of information (documents, charts, graphs, code, multimedia or other information representations). We propose to manage this in the form of a Dynabase: a dynamic database containing the notes, sketches, papers and other documents that are created over time. Much of the information that organizations collect today is already in a machine readable form. In a sense then, the creation of a dynabase is a byproduct of everyday work. Once information is in the dynabase, however,

how will it be viewed and retrieved?

To what extent will users have to add retrieval-enhancing information such as key words?

To what extent will the system be able to generate such clues by analyzing a document or the context in which it was created?

These are difficult questions, but if they sound insurmountable, do not despair-the system does not have to be perfect just better than it is today. [1] Rather than having to track down information in paper form (which often do not even include indexes) or go looking for a copy of a floppy with a document/chart on it or do a file search through directories we can use more automated searching/tracing techniques. An example of this is the Wide Area Information Servers (WAIS). This is a free-text search which is highly amenable to parallel computing (WAIS is implemented on a Thinking Machine's Connection Machine) . In most text-retrieval systems, queries are limited to Boolean combinations of a few terms, but since text on a Connection Machine is fast, searches for documents which are similar to an entire document are practical. This same technique is used in Dow Jones's DowQuest, a commercial system that uses a Connection Machine to scan more than 150,000 articles from 195 publications for relevance to on-line queries. [1]

Lotus notes is a commercial venture in providing tools for organizing people and diverse information sources. There are several other firms which have entered the field commonly known as groupware. This technology has stressed the importance of keeping all parties informed. What makes this technology very useful for our purposes is that it links (or ties) information not just vertically like traditional CASE tools but horizontally across numerous project functions; thereby increasing the amount of positive exchange of information between all developers and customers.

The information should be used to assist the customer and the developer in identifying critical areas, errors and make suggestions for possible remedies. For example with development data on line it may be possible to determine system schedules by accessing each of the development area's databases. It should also be possible to query the system to see how many functions are assumed to

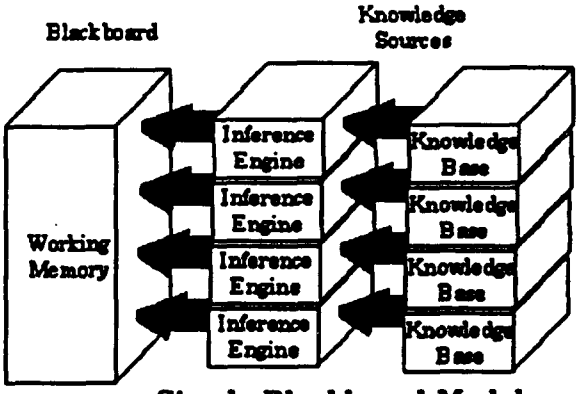
have access to a device or provide timing budget information so system performance can be determined. This information is available online, not reported every month. With the resultant information from these types of data analysis we hope to provide insight on our system shortfalls. This type of checking is simplistic. If we are to manage numerous forms of information we have to add more intelligence into our automation.

In [2] we worked on the problem of symbolic/numeric representation. In that work we managed the state space by observing trends in both the measured and computed data. On the basis of the observations, the knowledge base would generate a solution to the problem or generate a new set of parameters for reevaluating the state space. We accomplish the generation of symbolic information detecting thresholds and patterns [2]. Thresholds are numeric values for such things as in the example above. As thresholds are reached symbolic representations are given to the data set. These data sets are evaluated against our rule set to look for higher order patterns. When we examine the set we can determine what errors there are (if any) and where they are located by using our linked data. A useful technology for working this problem is a blackboard architecture (see Figure 1).

Large diverse clusters of data need to be used in a synergistic pattern in order for system engineers to better manage the process of development. A potential analysis tool is blackboard architecture. [3,7] A very basic blackboard concept is shown in Figure 1. The blackboard model is a complex problem-solving model prescribing the organization of both knowledge/data and the problem-solving behavior within a single architecture. This is similar to the familiar structured model of computation which has a program acting on data. A blackboard model consists of a global database called the blackboard. There are logically independent sources of knowledge which are called the knowledge sources. The knowledge sources will respond to changes in the blackboard to generate new hypothesis about the data. (there are several different models for blackboard architectures see [6] for more information on blackboards)

Typically the blackboard architecture is using these sources to find 'patterns' to help solve a problem jointly. For managing this data the solution may include finding issues that are common to a cluster of users. Simple examples may include keyword searches. Each author of a document submits the paper to the system. The system searches through the document and compares it with some keywords it may already have in its data base. The author of the document may have made changes since the last time the document was entered into the system. The system can recognize that there were changes and attempt to associate those changes with possible side-effects to other sections of the system under development. These changes can impact others in technical, cost and schedule areas. Low level information sources are clustered and the effects of this information is brought to the attention of the appropriate personnel.

These searches of patterns are not limited to simple keywords. It should be possible to access information from all of the dynabase that have been collecting information. It is possible for example to perform evaluations of data dictionaries of data flow diagrams and compare it with legacy systems to see if there are candidates for reuse and/or mapping of performance requirements to potential COTS (Commercial Off-the-Shelf) hardware/software components. (Currently there is an effort underway in the COSIP program to develop such a database to support this kind of effort.) Obviously not all factors can be evaluated, but a lot of the mundane consistency checks can be automated or at least provide some level of assistance (e.g. checking the government's specification against the contractors proposal).



Simple Blackboard Model
Figure 1

This type of activity would need to run in a batch type environment. Once the knowledge base has been created we envision the user navigating through both raw and processed information. By processed we mean that the knowledge sources have made some recommendations to the user which they may want to

follow up. We feel this is more how the system engineer is working,. Working in two directions both in a vertical direction and a horizontal, by having a tool kit that saves the engineer from spending their time looking for information, evaluating trivial information or reporting information they should be able to focus more on the problem solving issues. This type of information merger also provides a better picture to management who needs to monitor budget and schedule.

System development is a process of discovery, where as you add more information and evaluate it against previous knowledge you will find that more information is needed or the wrong information was given or new problems are exposed, etc. Each time you increase the granularity of knowledge you face the potential of finding errors. For complex systems help is needed in evaluating all of this information. In the area of reuse, we feel that incorporating previous work at earlier stages of development will help ensure maximum reuse.

Robust standards will play key roles in making large system development more manageable. Providing a developer a standard to 'design to' rather than having to develop new technologies to meet the needs will obviously reduce some degrees of freedom but it will also help bound the problem. The effectiveness of this will be premised on the quality of the standards. These standards need to be fully evaluated towards issues such as performance before they are specified.

Forcing the use of hardware/software/humware /security/communication standards will be difficult. Quite often a developers' cultural attitude is not to accept work from some other place, the classic 'not from my shop' syndrome. Fortunately, newer technologies such as object-oriented technologies are changing this perspective. The culture here is 'why should I rebuild what has already been done?'. This attitude is already common in hardware development where it was far from cost effective to generate a new hardware device when using existing ones is much more cost effective. The hardware culture here is about reuse. They proactively search for new sources of reusable components and develop skills and tools for solving problems with this *limited* set of components.

The hardware developer's task is somewhat limited. The software aspect has to encapsulate the functionality of what the user wants and logically control the hardware resources. This mapping to the hardware is dynamic and difficult to predict. But standards are a good method of reducing the number of unique mappings. These standards will need to be part of our database/knowledge sources, they will provide convenient 'bounding' to our problem solving activity.

Future: Object Technology + Visual Technology + Assemblers

In the past when we were faced with problems of high degrees of complexity we raised the level of abstraction to manage the problem. Where we had once used assembly language as the language to build our systems we now use high level languages such as Pascal, Ada and C++. We do not attempt to manage the assembly code generated by these languages. We argue that it is time again to raise the level of abstraction. In the beginning part of this paper we discussed ways of automating the system engineering process to help improve communication between the developer and the customer. These are solutions for today. For tomorrow we envision a different relationship all together.

Instead of having developers work with the customer as in model three, developers will provide classes of objects. These classes will be accessed with visual environments that will help the user understand their functionality. Since there are differences between the rather simple behavior of a hardware device and the wide range of functionality of a software component additional information should be made available to the user. This can include multimedia presentations, graphical representations (data flow, object, Petri nets, etc.) besides textual descriptions. The user will 'assemble' these classes together at a high level of abstraction using graphical techniques.

Visual programming languages (such as Prograph) and even simulation tools (like SES/Workbench and CACI) have added iconic representations of elements that can be connected in some digraph form. These elements make up a rich set of primitive functions that can be further defined by selecting properties of each of the elements. Metamodeling allows the user of such an iconic model to build new representations from these primitives to better represent their problem under study.

The problem of reuse and repository science is still an open issue. The metamodel philosophy allows us to tailor generic techniques to match individual needs. This would require developers to work on the

environment supporting the users doing the development rather than performing the development themself. The user is working more like in the first model which had the highest level of communication and the developer is working more like the second (perceived need) rather than the third model (user to developer). The communication between the developer and the customer should be of higher quality and in shorter spurts since communicating will focus on construction of a particular class rather than an entire system.

Developers will work more on building up class libraries of information for the customer to use to build the system. This is similar to how hardware development occurs now. A hardware vender, using business model two, sees a need for a particular hardware device. They build the device and market it to their customers. After a while whole catalogs of these devices are available, like a TTL data book, and users will build their new systems from these catalogs. Work has already been done in this area for VHDL, where they are building libraries of models of devices for simulation tools.

These are not new concepts but technologies such as blackboards, parallel computers to speed up searches, a new cultural attitude towards reuse and better graphical capabilities make these ideas possible. The framework we spoke of earlier can become the repository of this new assembler technology.

Summary

Our research has lead us to not have a single 'static' representation of the system but rather to express the problem domain in a dynamic fashion. This has led to developing a unified informal/formal paradigm. We attempt to express the information, the same information, in numerous forms and allow for various methods of information retrieval. It's important that this representation be in a form that all people involved are comfortable with and basic enough to allow for communication across different domains.

As reuse/reengineering/repository technologies mature, new business models will emerge. Visualization/Commercial Off-the-Self hardware will allow for users to develop more of their system than previously done.

This paper addressed a simple wish list of what a system engineering framework should do. It is heavily biased on comments made from a small set of system engineers and my own preferences. In no way is this a comprehensive list of what is required to meet the system engineer's needs but it is a step in the direction of specifying what an engineering station could look like.

References

[1] Press,L. Collective Dynabases, Communications of the ACM, Vol. 35, No.6, June 1992, pp26-31

[2] Baylog, J., Zile, S. and Rumbut, J. TARSIA: An Intelligent System for Underwater Tracking,Proc.of the Expert Systems in Government Conference (Philadelphia Oct. 86).

[3] Engelmore, R. and Morgan, T. (Eds) Blackboard Systems, Addison Wesley, Wokingham England,1988

[4] Page-Jones,M. The CASE Manifesto,Computer Language, June 1993

[5]Bowen, T.F. , Gopal,G.,Herman,G., Hickey, T.,Lee, K.C. , Mansfield,W.H., aitz, J. and A. Weinrib, The Datacycle Architecture, Communications of the ACM, Vol. 35, No. 12, December 1992, pp71-81

[6] Belkin, N.J. and Croft,W.B., Information Filtering and Information Retrieval: Two Sides of the Same Coin?,Communications of the ACM, Vol. 35, No. 12, December 1992, pp29-38

[7] Selfridge, O. Pandemonium: A Paradigm for Learning, Symposium on the Mechanization of Thought Processes, H.M. Stationery Office, London, 1959

Requirements Metrics: The Basis of Informed Requirements Engineering Management

Robert J. Halligan

Technology Australasia Pty Limited
1010 Doncaster Road
Doncaster East Vic 3109 Australia
Fax 61-3-841-8374

Abstract

Available data demonstrates that defective requirements are a dominant cause of cost and schedule overrun in defense and aerospace programs. This paper presents a structured methodology for measuring the quality of requirements, individually and collectively. It is shown that requirements may be characterized by ten quality factors, each with an associated metric, and by an overall requirements quality metrics. In addition, the requirements engineering process itself can be instrumented by means of five process-related metric. The paper describes the author's experience with application of both types of metric to engineering decision making. A tool which automates aspects of metrics collection is presented.

1 Introduction

Requirements engineering deals with the capture, analysis, expression and traceability of requirements. Requirements engineering may commence at the level of a broad statement of military need, and will continue through the definition of the system solution, right down to the lowest levels of specification of elements of that solution, for example, C, D and E specifications in the hardware world and minispecs in the software world.

Requirements engineering does not simply happen, it requires management. Classically, management is considered to comprise planning, organizing, staffing, monitoring and controlling. If we accept that "that which cannot be measured cannot be controlled", the role of requirements metrics is readily apparent.

But which metrics? Should we instrument the product (the requirements) or the process (the requirements engineering process) or both? How can requirements metrics be used to help the project team satisfy project success criteria? These and related issues are addressed below.

2 The State of the Requirements Art

Data from TRW developed in the early 1980s showed that, on a range of representative projects, 30 per cent of design problems requiring correction were due to erroneous or incomplete specifications. Another 24 per cent of errors were due to conscious deviation from product and process requirements. Other studies [1] have shown that the cost to correct an error typically increases by a factor of between 20 and 1000 over the life cycle of a system acquisition. System solutions which satisfy the contract, but not the need, are, unfortunately, commonplace.

Engineering practitioners have come to regard improved requirements engineering as one of the challenges of the 90's. The responses to this challenge have included:

- early, concurrent development of product and process requirements covering all product life cycle phases, from concept through to disposal [2];
- improved analysis of requirements by the use of operational requirements languages and associated tools, for example RDD [3]; and
- management of requirements through integration of text processing and relational database (or similar) support [4], resulting in improvements in requirements traceability and in the productivity of requirements analysis and flowdown activities.

This latter trend has brought with it a tendency, highly beneficial in the author's view, to manage all program requirements as a single set. Requirements may be readily allocated across all elements of the program, for example the prime mission products(s), project management, system engineering, test and evaluation, production, etc, and their interfaces. Within each of these program elements, requirements may be decomposed and allocated to lower level elements, product interfaces and functional interfaces.

3 Users of Requirements

Since requirements define the product or process to be realized, it is axiomatic that the success of any program is closely linked to the adequacy of definition and communication of requirements:

- users rely on requirements as a precise expression of their need;
- the program office relies on requirements for eliciting offers;
- both the customer and the contractor rely on requirements as an expression of their agreement as to what is to be delivered; and
- the functional elements of the project organizations of both the customer and the contractor rely on requirements as an expression of what they are to deliver to their respective internal customers.

4 Requirements Quality

Requirements, to satisfy their users, must, in their expression, exhibit certain attributes. We refer to these attributes as requirements quality factors. The author has found that a set of ten requirements quality factors is necessary to adequately define the quality of requirements, individually and collectively.

Correctness refers to an absence of errors of fact in the statement of requirement.

Completeness requires that the requirement contain all of the information necessary, including constraints and conditions, to enable the requirement to be implemented such that the need will be satisfied.

Consistency requires that a requirement not be in conflict with any other requirement, nor with any element of its own structure.

Clarity requires that the requirement be readily understandable without semantic analysis.

Non-Ambiguity requires that there be only one semantic interpretation of the requirement.

Connectivity refers to the property whereby all of the terms within the requirement are adequately linked to other requirements and to word and term definitions, so causing the individual requirement to properly relate to the other requirements as a set.

Singularity refers to the attribute whereby a requirement cannot sensibly be expressed as two or more requirements having different subjects, verbs and/or objects.

Testability refers to the existence of a finite and objective process with which to verify that the requirement has been satisfied.

Modifiability requires that:

a. necessary changes to a requirement can be made completely and consistently; and
b. the same requirement is specified only once.

Feasibility requires that a requirement be able to be satisfied:

a. within natural physical constraints;
b. within the state-of-the-art as it applies to the project; and
c. within all other absolute constraints applying to the project.

5 A Requirements Structural Model

Requirements are most commonly expressed as natural language statements, although graphical and formal mathematical requirements languages are widely used.

For the natural language type of expression, requirements quality metrics may be developed through the parsing of each requirement statement into the elements of a structural model of a sound requirement, a template. A template found to be suitable for English language requirement statements is illustrated in figure 1 [after 5]. Figure 1 also shows an example requirement parsed into the template.

Original Requirement:

In the Combat Zone, an HQ Switch, which is identical to a trunk node switch, shall be given two (2) independent links to at least two (2) other nodes in the network.

Element	Text
1 Actor	an HQ Switch
2 Conditions for Action	in the Combat Zone
3 Action	shall be given
4 Constraints of Action	
5 Object of Action	two (2) independent links
6 Refinement/Source of Object	
7 Refinement/Destination of Action	to at least two (2) other nodes in the network
8 Other	which is identical to a trunk node switch

Figure 1 - Requirement Structural Template

Elements of the template are defined, generally in accordance with Fuji (5), as below:

Actor/Iniator of Action. This is the subject of the sentence - the thing being specified. Examples are: "the system", "the interface", "the function", ...

Action. This is a verb - the action to be taken by the actor (subject). Examples are "shall calculate", "shall display", "shall fly", ...

Object of Action. This is a noun, and is the thing acted upon by the actor. Examples are: "the message", "the input signal", ...

Conditions of Action. This defines the conditions under which the action takes place, for example"upon receipt of a message", "in high resolution mode", "within 10 minutes of power-on", ...

Constraints of Action. This qualifies the action, for example "at a resolution of 400 x 1000 pixels", "within limits imposed by vehicle speed", ...

Refinement/Source of Object. These qualify the object, for example (refinement): "of flash priority", for example (source): "from DISCON".

Refinement/Destination of Action. These further qualify the action, and may be additional to Constraints of Action. Examples are "within 10ms", "to DISCON".

Other. This element collects non-requirements material.

6 Requirements Quality Metrics

A strong requirement will have each applicable element of the requirement, and the requirement overall, satisfying each of the quality factors described earlier. This ideal provides a basis for the development of requirements quality metrics.

Figure 2 illustrates the construction of a set of metrics based on parsing of a requirement into the template.

These metrics are defined below.

IRQ Individual Requirement Quality

This metric for a single requirement is a number between 0 and 1, 1 representing a "perfect" requirement and zero representing a totally defective requirement. The metric is constructed from the parsed version of the requirement by:

a. determining which of the possible seven elements of the structure are applicable and assigning a value of 1 to each applicable element (most requirements have 5-7 applicable elements);
b. assessing each element of the parsed requirement against the quality factor criteria, and scoring each applicable element as 1 (satisfactory) or 0 (unsatisfactory). An element may be unsatisfactory because it is missing, or because it is defective in some other way. A variant on the approach is to permit individual element scores between the limits of 1 and 0, although it is doubtful whether this refinement offers any significant benefit;
c. calculating the metric by dividing the sum of the applicable element values into the sum of the element scores.

IQF1-IQF10 Individual Quality Metrics

Ten individual (requirement) quality metrics correspond to the ten requirement quality factors, as follows:

IQF1	Correctness
IQF2	Completeness
IQF3	Consistency
IQF4	Clarity
IQF5	Non-Ambiguity
IQF6	Connectivity
IQF7	Singularity
IQF8	Testability
IQF9	Modifiability
IQF10	Feasibility

Structural Element	Applicability	Score		Metric Name	Metric Value
Actor	1	0	Correctness	IQF1	0
Conditions of Action	1	0	Completeness	IQF2	0
Action	1	0	Consistency	IQF3	1
Refinement of Action	0	0	Clarity	IQF4	1
Object of Action	1	0	Non-ambiguity	IQF5	0
Refinement/Source of Object	1	0	Connectivity	IQF6	0
Refinement/Destination of Action	1	1	Singularity	IQF7	1
TOTAL	6	1	Testability	IQF8	0
Metric IRQ	0.17		Modifiability	IQF9	1
Omission Ratio	5.00		Feasibility	IQF10	1

Figure 2 - Construction of Requirement Quality Metrics

These metrics assume, for an individual requirement, a value of 1 or 0 depending on whether the requirement overall has a defect of that type (0) or not (1). Again, scoring between these range limits may be used if desired.

The metrics for individual requirements rarely directly serve a useful purpose. It is necessary to aggregate individual requirements metrics to form metrics for groups of requirements in order to serve our objective of control. In making this transposition to aggregate metrics, we have consistently found the need to adjust completeness to allow for requirements which are missing altogether, not just incomplete in the sense of missing a condition or a refinement.

Requirements which have been omitted may be accounted for by estimating an *omission ratio* for each requirement that *is* present. The omission ratio is the number of new requirements that would be created if all possible areas of omission suggested by the requirement that *is* present were pursued to resolution. The omission ratio must be constructed such as to support aggregation of requirements having different omission ratios.

The quality metrics for sets of requirements correspond to, and are produced from, the individual metrics, as follows (for n requirements):

RQ *Requirements Quality*

$$RQ = \frac{\sum IRQ}{n}$$

QF1 *Correctness*

$$QF1 = \frac{\sum IQF1}{n}$$

QF2 *Completeness*

$$QF2 = \frac{\sum QF1}{n} - \frac{\sum omissionratio}{n}$$

Note that completeness may have a negative value.

QF3 to QF10 are derived as for QF1.

7 Application of Requirements Quality Metrics

A metric is only of value if it assists in decision making.

Areas of application of the metrics described above are summarized in Table 1.

Metrics should only be used where they contribute positively to the degree of satisfaction of project goals, including cost goals.

Metric	Application
RQ Requirements Quality	• estimation of requirements-related bidding risk/opportunity (depending on the type of contract) • estimation of requirements-related contract risk/opportunity • determination of the skills and level of resources required for requirements analysis • measurement of the quality of the product of requirements analysis, in relation to decisions such as: a. termination of formal requirements analysis; b. whether the project is ready for System Requirements Review (SRR), Software Specification Reviews (SSR) and other requirements reviews; c. whether system requirements are sufficiently mature for establishment of the functional baseline; d. whether CI requirements are sufficiently mature for establishment of the allocated baseline; • assessment of the specification writing skill levels of project team members • estimation of requirements-related subcontract risk/opportunity • use as a technical performance measurement (TPM) parameter
QF1-QF10 Requirements Quality Factors	• identification of aspects of requirements which are unsatisfactory • identification of requirements-related skills in which training of project personnel is needed • use as a TPM parameter

Table 1 - Application of Requirements Quality Metrics

8 Typical Values of Requirements Quality Metrics

Our experience in use of the metrics suggests the typical relationships between values of the metrics and requirements quality shown in Table 2.

9 Requirements Process Metrics

Table 1 indicated the application of requirements *quality* metrics. We have also found it beneficial to use, for engineering management purposes, requirements *process* metrics, derived for requirements analysis tasks such as system requirements analysis, software requirements analysis for CSCIs and hardware requirements analysis for HWCIs.

Useful metrics include:

RSTA Percent Started

This metric indicates the percentage of source requirements currently under development, the "work in progress".

RTBD Percent "To be Determined"

This metric indicates the percentage of requirements containing TBDs, ie, requirements for which the resolution of incompleteness is beyond the resources of the analyst and which have been referred to other individuals, organizations or phases for resolution of missing information.

RCOM Percent Completed.

This metric indicates the analyst's view of that analysis of the source requirement has been completed.

RAPP Percent Approved.

This metric indicates the percent of source requirements for which the results of analysis (child requirements) have been approved for incorporation in the destination document.

In addition, the need to control the process of formally decomposing and allocating requirements of an element in the system hierarchy to its subordinate elements has led to an additional metric:

RALL Percent Allocated.

This metric indicates the percent of parent requirements of an element at one level of the WBS for which the corresponding child requirements have been allocated to the applicable lower level elements.

All of the above process metrics provide data for earned value measurement within project cost/schedule control systems. In addition, RTBD has proved to be a useful parameter for incorporation into a technical performance measurement (TPM) program [2].

Metric	Very poor set of requirements, requiring substantial development	Fair set of requirements, may just be suitable for purposes of solicitation, depending on the SOW and type of contract envisaged	Requirements at SRR suitable for carrying forward into development	Requirements suitable for establishment of the Functional Baseline
RQ-	0.01-0.3	0.3-0.7	0.95-0.99	0.99+
QF1-Correctness	0.9	0.98	0.99	0.99+
QF2-Completeness	-5	0	0.95	0.99+
QF3-Consistency	0.9	0.97	0.99	0.99+
QF4-Clarity	0.9	0.97	0.99	0.99+
QF5-Non-Ambiguity	0.3	0.7	0.9	0.98+
QF6-Connectivity	0.3	0.9	0.99	0.99+
QF7-Singularity	0.1	0.3	0.99+	1
QF8-Testablity	0.1	0.7	0.99	0.99+
QF9-Modifiability	0.1	0.5	0.99	0.99+
QF10-Feasibility	0.95	0.99	0.99+	0.99+

Table 2 - Typical Values of Requirements Quality Metrics

10 Computer Support to Metrics Generation

Requirements management benefits substantially from the use of computer based tools which facilitate, in particular, efficient text handling, rigorous requirements allocation and the creation and maintenance of peer and parent-child relationships for requirements traceability purposes. Metrics prove to be most easily calculated where a CASE environment is in use for those other aspects of requirements management.

One CASE tool for requirements management with which the author has experience is Document Director ReqMgr, produced by Bruce G. Jackson & Associates, Inc. A prototype software package which automates storage of metrics-related requirements quality and process data and which progressively builds up requirements quality and process metrics has been built for use with Document Director ReqMgr.

Proprietary tools known to the author are also being utilized in a similar way by other organizations.

11 Conclusions

Numerous best practice standards (ISO 9001, Software Engineering Institute criteria, MIL-STD-499B) emphasize a closed loop process as a key to effective technical management. The metrics described in this paper are a means of implementing closed loop control over the requirements engineering process.

The cost of implementing these metrics within a suitable, existing CASE environment appears to be around two percent of the cost of the total requirements engineering effort. The engineering manager must decide whether the resulting payoff will exceed this cost. Sufficient data to conclusively answer this question has not yet been developed by the author, nor has it been identified from other sources.

Assessment of the cost-effectiveness of the use of requirements metrics must therefore, for the present, be subjective. It is the author's assessment that requirements metrics, developed on a sampling basis, used within a suitable CASE environment, provide considerable leverage in satisfying the goals of complex systems development.

Greatest leverage is obtained where sampling techniques are used in metric development. Such sampling may focus on, say, every nth requirement, or on areas of perceived risk.

References

[1] B.W. Boehm, *"Software Engineering Economics"*, Prentice-Hall, Englewood Cliffs, N.J., 1981

[2] MIL-STD-499B Draft, *"Systems Engineering"*, 6 May 1992

[3] M. Alford, *"Strengthening the Systems/Software Engineering Interface for Real-Time Systems"*, Proceedings of the Second Annual International Symposium of the National Council on Systems Engineering, Seattle, 1992

[4] M.B. Pinkerton and F.R. Fogle, *"Requirements Management / Traceability: A Case Study - NASA's National Launch System"*, ibid

[5] R. Fuji course notes on *Independent Verification and Validation*, 1989

Engineering and Analysis of Real-Time Systems

Jay K. Strosnider[1]
Department of Electrical & Computer Engineering
Carnegie Mellon University
Pittsburgh PA 15213

January 7, 1993

Abstract

The following paper presents a unified framework for reasoning about timing correctness on arbitrary serially reusable resources. The proposed approach bridges the gap between real-time scheduling theory and its implementation on physical resources via scheduling models. We define scheduling models as abstractions that can be used to reason about timing correctness on physical resources. We argue that a consistent set of scheduling models for all shared resources encapsulated in the System Engineering Workbench (SEW) enable the real-time systems architect to quickly explore the system design space, to establish and maintain a firm performance baseline, to facilitate system resource partitioning and management, to quantitatively evaluate hardware/software boundary issues, to optimize system configuration parameters, and to explore the impact of new technologies. Further, we argue that scheduling models operate at the right level of abstraction not only for system performance validation, but also for interfacing with the Software Development organization. Scheduling models of CPU/Operating Systems, backplane buses, disk subsystems, and local area networks have been developed, including the following specific models: Real-Time Mach, MWaveOS, Futurebus+, Microchannel Architecture, FDDI, and IEEE 802.5 token rings.

[1]This research is supported in part by grants from the Office of Naval Research and the Naval Ocean Systems Center under contract N00014-91-J-1304

1 Introduction

Developing large, complex, distributed systems is a trying evolutionary process fraught with technical and financial difficulties for both the contractor and the customer. Aggressive development schedules, coupled with the inherent complexity of these systems, forced systems development into concurrent engineering practices, long before there was such a thing as concurrent engineering. Projects moved from conceptual design to detailed design to unit hardware/software integration and test, and onto systems integration and test as much by programmatic definition as by technical maturity at each stage. Schedule pressures generally result in inadequate up-front Systems Engineering, which leads to incomplete and inadequate development specifications for the hardware and software design/development organizations. As the system moves towards sell-off, the problems inevitably snowball. The development process often degenerates into an interactive fire-drill between the customer, Systems Engineering and the development groups.

Given this preamble, the approach advocated in this paper will go a long way towards alleviating the problem by providing methodologies that can be encapsulated into System Engineering friendly tools. These tools will enable the System Engineer to quickly explore the systems-level design space in a quantitative manner and answer the question, "Will the system work?" at any point in the development process. Further, the System Engineering Workbench (SEW), with its associated analytical framework, directly supports concurrent engineering by providing the appropriate level interface to the software and hardware development organizations. The Systems Engineer thus works directly with the development organizations reacting, resolving, and validating the inevitable changes, while simultaneously being the resource management/allocation watchdog.

The proposed solution strategy has three core components: first develop a consistent set of scheduling models for all system resources, second develop a set of the appropriate figures of merit (FOMs) to support quantitative design decisions, and third encapsulate the scheduling models and FOMs into a user friendly tool, the System Engineering Workbench. We do not discuss the figures of merit in the paper. Section 1.1 presents a generic framework for reasoning about timing correctness of physical resources that forms the basis for the scheduling models. Section 1.2 provides an overview of the SEW tool and Section 2

summarizes how to design real-time systems using the proposed approach.

1.1 Scheduling Models

Scheduling theory holds great promise as a means to *a priori* validate the timing correctness of real-time applications. However, there currently exists a wide gap between idealized scheduling theory and the implementation realities of building large, distributed systems composed of real processors with real operating systems communicating over real buses and real networks supported by real disk subsystems. Specifically, we propose a set of consistent scheduling models which accurately model the timing and concurrency behavior of system services software supported by the underlying system hardware resources.

We define system services software to be all the software which provides the infrastructure upon which the application software runs. This includes: operating systems, database management systems, network management systems, user interface/window management systems, etc. Each of these system services software components is associated with managing its associated hardware resource, i.e. central processing unit (CPU), disks, networks, displays, etc. Currently, scheduling theory assumes an idealized resource with zero overhead and perfect preemptability. The scheduling models reported here extend scheduling theory to address the overhead and limited preemptability costs of scheduling application software via system services software running on real hardware assets. One simply cannot answer the question, "Will the system work?" without correctly addressing the overhead and limited preemptability issues.

Consider the ideal scheduling equations for fixed and dynamic priority scheduling summarized in Table 1. The fixed priority scheduling equations are general to any fixed priority assignment. The dynamic priority utilization-based equation applies for either earliest deadline or least slack time algorithms. The dynamic time-based formulation applies specifically to the earliest deadline algorithm [1]. A similar algorithm for least laxity could also be developed. In general the time-based, exact case equations provide necessary and sufficient conditions for schedulability whereas the utilization based equations provide less tight sufficient conditions for schedulability. The exception is the 100% utilization bound for idealized earliest deadline scheduling which is a utilization-based necessary and sufficient schedulability bound.

Note that Table 1 does not include the full conditions for checking schedulability for the time-based, dynamic priority case due to space limitations. The cumulative work $W_{ik}(t)$ across the interval defined by

Ideal (Time-Based) Scheduling Equations

	Time
Fixed	$\forall\ i=1,2,\ldots,n \quad \min_{0 \le t \le D_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \le 1$
Dynamic	$W_{ih}(t) = \sum_{j=1}^{n} \min\left(\left\lceil \frac{t}{T_j} \right\rceil, \left\lfloor \frac{hT_i}{T_j} \right\rfloor\right) C_j$ $B(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i$

Ideal (Utilization-Based) Scheduling Equations

	Utilization
Fixed	$\forall\ i=1,2,\ldots,n \quad \sum_{i=1}^{n} \frac{C_i}{T_i} \le i(2^{\frac{1}{i}} - 1)$
Dynamic	$\forall\ i=1,2,\ldots,n \quad \sum_{j=1}^{i} \frac{C_j}{T_j} \le 1$

Table 1: Ideal Scheduling Equations Summary

the busy period length, $B(t)$ must be evaluated to check that each job of each task meets its deadline. The algorithm that performs this check is given by:

```
Find the busy period of the task set.
For each task in the task set,
   For each job in the busy period,
      Find the completion time of the current job
      If the completion time > deadline, test fails, exit
      Adjust the idle time to the deadline of the next job
      While the current time < the arrival of the next job
         Increment the idle time to the next arrival
         Iterate forward
```

The dynamic, time-based check is a much more complicated test that than the dynamic utilization-based check and will yield the same result for the ideal case. It is included here for completeness and as a precursor to the dynamic, time-based scheduling models presented later.

We now extend the equations summarized in Table 1 to include the implementation effects of overhead and blocking. The extended equations summarized in Table 2 constitute a set of generic scheduling models that will be used to reason about timing correctness on the various resources. The generic scheduling models have three additional components:

- $Overhead_i$ which captures the task dependent scheduling overhead that can be directly bound to the application task.
- $Overhead_{sys}$ which captures the task independent system level overhead encountered on some resources.
- $Blocking_i$ which captures the time task τ_i is delayed executing by lower priority tasks due to imperfect preemptability.

The $Overhead_i$ component effectively increases the run-time C_i. Other than increased loading, it has no other effects on the schedulability of the task set. The $Overhead_{sys}$ component often shows up as a periodic system level task that can be readily incorporated into the scheduling framework. The $Blocking_i$ component due to imperfect preemption degrades the fixed priority, time-based formulation and the dynamic priority utilization-based formulations from necessary and sufficient schedulability conditions to

Generic (Time) Scheduling Models

	Time
Fixed	$\forall\ i=1,2,\ldots,n \quad \min_{0 \le t \le D_i} \sum_{j=1}^{i} \frac{C_j+Overhead_j}{t} \left\lceil \frac{t}{T_j} \right\rceil + \frac{Overhead_{sys,i}}{t} + \frac{Blocking_i}{t} \le 1$
Dynamic	$W_{ih}(t) = \sum_{j=1}^{n} min\left(\left\lceil \frac{t}{T_j} \right\rceil, \left\lfloor \frac{hT_i}{T_j} \right\rfloor\right)(C_j + Overhead_j) + Overhead_{sys,i} + \frac{Blocking_i}{t}$ $B(t) \sum_{j=1}^{i} C_j + Overhead_j \left\lceil \frac{t}{T_j} \right\rceil + Overhead_{sys,i} + Blocking_i$

Generic (Utilization) Scheduling Models

	Utilization
Fixed	$\sum_{i=1}^{n} \frac{C_i+Overhead_i}{T_i} + \frac{Overhead_{sys}}{T_{sys}} + max_{1 \le i \le n} \frac{Blocking_i}{T_i} \le i(2^{\frac{1}{i}} - 1)$
Dynamic	$\sum_{i=1}^{n} \frac{C_i+Overhead_i}{T_i} + \frac{Overhead_{sys}}{T_{sys}} + max_{1 \le i \le n} \frac{Blocking_i}{T_i} \le 1$

Table 2: Generic Scheduling Model Summary

sufficient schedulability conditions. With imperfect preemptability it is no longer necessarily true that the response time of the first job of a task in critical zone phasing is always the longest [2]. Thus only the time-based dynamic priority formulation remains a necessary and sufficient condition for schedulability. We further note that the time-base approach [1] was specifically developed to address the impact of overhead and blocking in dynamic priority scheduling algorithms. For a rigorous proof of the sufficiency conditions for the generic scheduling models see [3].

The generic scheduling models summarized in Table 2 can be applied to develop schedulability criterion for concurrent real-time task execution of arbitrary system resources. Table 3 summarizes the dominant effects that contribute to $Overhead_i$, $Overhead_{sys}$ and $Blocking_i$ for CPU's/OS's, Buses, Disks and Local Area Networks. Note that the scheduler's functionality shows up in very different forms on the various resources. Operating systems schedule CPUs. Buses are scheduled using hardware implemented arbitration protocols. Disks are scheduled with a combination of hardware and software in their controllers, and LANs are scheduled via their Media Access Control (MAC) protocols. Scheduler/arbiter implementations range from fully software implemented for operating systems to full hardware implementations on buses and networks.

The $Overhead_i$ component for CPU scheduling is composed mainly of the scheduler overhead along with associated Interrupt Service Routines (ISRs), synchronization protocols, device drivers, etc. The bus $Overhead_i$ component tends to be dominated by the time for the arbitration lines to settle along with addressing overhead and miscellaneous control functions. For disks, the $Overhead_i$ component is dominated by the physical movement of the head, and to a lesser extent the disk controller overhead. On LANs, $Overhead_i$ tends to be dominated by propagation delays along with addressing and miscellaneous control functions.

The $Overhead_{sys}$ component shows up on all the resources in different forms as well. Operating system event handling and scheduling is typically done at discrete intervals defined by an underlying periodic timer interrupt. This system overhead component is not bound to specific tasks and may be readily modeled as one or more additional system level task(s), τ_{sys} with a run-time of $Overhead_{sys}$ and a period T_{sys}. Backplane buses often must support refresh of dynamic RAM memory components. The overhead associated with this function is readily modeled as a periodic $Overhead_{sys}$ component. Similarly disks have

Subsystem	Scheduler/Arbiter	$Overhead_j$	$Overhead_{sys}$	$Blocking_i$
CPU/OS	S/W Protocol: Flexible, RMS EDS, etc.	Scheduler, ISR, Synch Device Drivers, etc.	Time-Tic	Time-Tic Kernel-Funcs.
Buses	H/W Protocol:RR Pos. priority Msg. based priority	Arbitration, Addr. Sync, Misc. Control	DRAM refresh	f(max. tenure)
Disks	S/W Protocol RMS, EDS, Pscan Tscan, etc.	Head Movement Dominated, control S/W	Scan overhead	f(max. tenure)
LANs	H/W Protocol RR, Fixed priority	Prop. Delay, Addr. & Misc. control	TTRT task etc.	f(max.pkt,prop.delay)

Table 3: Overhead and blocking for various sub-system components

an $Overhead_{sys}$ component due to the overhead from scanning. Although LANs typically do not have an $Overhead_{sys}$ component, FDDI is an exception. Assuming prioritized scheduling at each FDDI node and the use of the synchronous mode protocol, the scheduling affects of the other nodes can be modeled as an $Overhead_{sys}$ component.

The $Blocking_i$ component arises on each of the resources due to imperfect preemptability. On CPUs the $Blocking_i$ component is generally closely tied to the timer interrupt rate that drives the scheduler. On buses, the $Blocking_i$ component is a function of the maximum transaction time on the buses. Similarly, on networks, $Blocking_i$ is a function of the maximum packet size as well a propagation affects. Disks are generally nonpreemptable. Thus $Blocking_i$ is generally a function of the maximum single transaction size permitted.

The previous section introduced generic scheduling models as a means for bridging the gap between idealized scheduling theory and the implementation realities associated scheduling real processes on real resources. Two new terms were introduced to account for implementation overheads: $Overhead_i$ to ac-

count for overhead components that can be bound to each individual task, τ_i, and $Overhead_{sys}$, to account for system overhead components that cannot be directly bound to individual task executions. A $Blocking_i$ term was introduced to account for the time that task τ_i is delayed execution due to imperfect resource preemptability. We then summarized the dominant components of these overhead and blocking terms for CPUs, buses, disks and LANs.

1.2 System Engineering Workbench

Designing an effective tool at the system level is a difficult task. We are currently building the System Engineering Workbench to encapsulate the models, methodology and figures of merit. The following paragraphs summarize the design of the SEW tool.

The SEW tool is an X-Window based system, with a user friendly interface that will allow the Systems Engineer to interactively reason about real-time systems design via the complex theoretical models. The philosophy of the tool is provide an interface that will allow the engineer to easily add new models within a consistent framework, to modify system parameters and quickly view the results, and experiment with system design ideas and verify that the timing correctness of the system will hold.

SEW consists of the following components: task set editor, subsystem and system editors, subsystem and system simulators, and the scheduling analyzer. The following subtasks describe the function of each, and the process that we will go through to develop each. Note that an X-Window prototype is currently being developed, and that the bulk of the analytical software that operates behind the user interface is already in place. A high level view of SEW is provided in Figure 1. The following paragraphs provide a brief summary of the form and function of each of the major components of SEW.

- **Task Set Editor:** This component of SEW provides a window for editing the application task description. This window will allow the user to graphically draw interconnected bubbles representing tasks and their interactions. Each task can also be individually edited, allowing the user to specify task resource requirements, including timing, communications, etc. Eventually, this window will be hierarchical in nature, allowing common tasks to be grouped together.

- **Subsystem and System Editors:** This component of SEW provides a common window interface

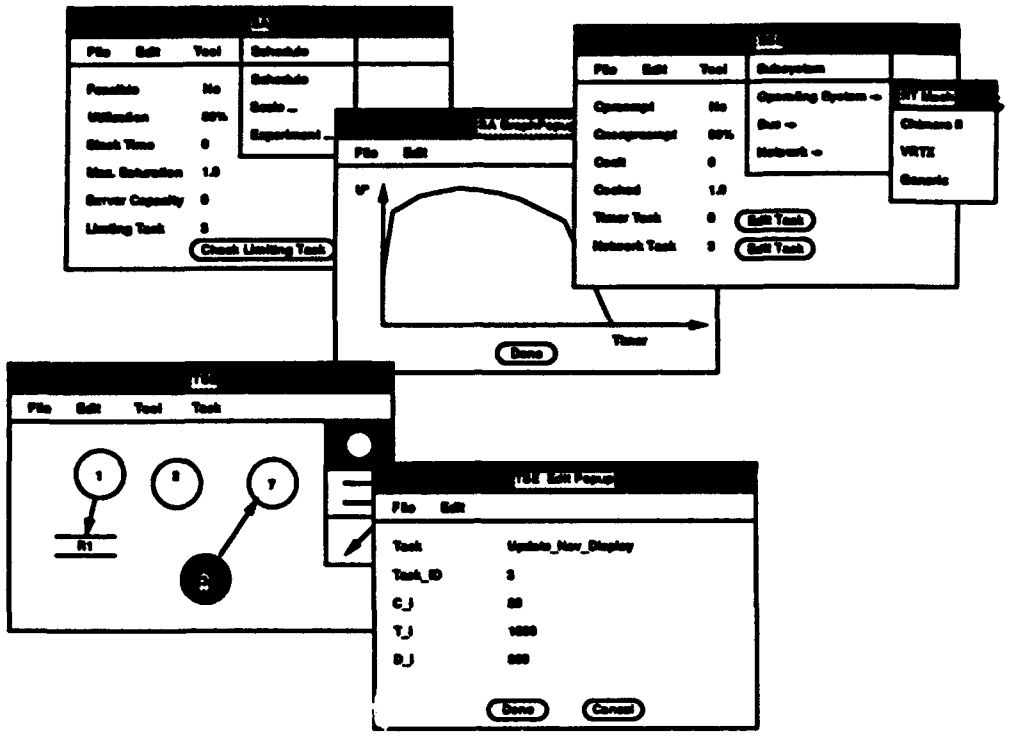


Figure 1: Systems Engineering Workbench Overview

for creating and modifying models of the various subsystems and the complete real-time system. The various subsystems will be edited here, allowing the user to change system parameters, such as blocking and overhead measurements. The system editor will specify the physical interconnects between the various subsystems.

- **Subsystem and System Simulators:** An important part of the SEW tool is the ability to view the expected task interactions. This window will allow the user to take the task set that has been created and simulate its execution on either a particular subsystem or a combination of subsystems into a system. This simulation is done at a high level, so that the user can see how the tasks interact.

- **Scheduling Analyzer:** This final window is perhaps the most important; through it the user can test the schedulability of the created task set running on the given subsystem or system. This window will allow the user to test the schedulability under any fixed or dynamic priority scheduling criterion. The scheduling analyzer will also allow the user to experiment with the system design space, by varying system parameters and viewing the resultant effect against quantitative figures of merit. Through this window, the user can view common real-time system metrics for the task

Sub-System Type	Models Available
Operating Systems	RT Mach (68030, R3000, i960) MWaveOS Chimera II
Busses	Futurebus+ Microchannel
Disks	Generic Model
Networks	IEEE 802.5 Token Ring FDDI IEEE 802.6 DQDB ATM Switches

Table 4: Scheduling Models Summary

set, such as utilization, breakdown utilization, server capacity, slack time, as well as find optimal operating points for such things as operating system timer interrupt rate or minimum packet size on a network.

The SEW tool is currently being developed and is reported here to emphasize the need for appropriate tools to support the real-time systems engineer. This tool supported by the underlying scheduling model and FOMs can greatly can help move real-time systems engineering from a practice to a sound engineering discipline. In table 4 we summarize the scheduling models that have been developed to date.

2 Engineering Real-Time Systems via Scheduling Models

This paper presented a unified framework for reasoning about timing correctness on arbitrary, serially reusable resources. The proposed approach bridges the gap between real-time scheduling theory and its implementation on physical resources via scheduling models. We defined scheduling models as abstrac-

tions that can be used to reason about timing correctness on physical resources. We argued that a consistent set of scheduling models for all shared resources together with relevant figures of merit encapsulated in the Systems Engineering Workbench (SEW) enable the systems architect to:

- **Validate system level response time performance.** This core capability of the scheduling models facilitates all the capabilities listed below. Given a set of task run-times, I/O and synchronization requirements, the scheduling models allow the Systems Engineer to determine whether all response time requirements will be met.
- **Quickly explore the system-level design space, and establish a firm baseline.** Given a consistent set of scheduling models of the system's components (CPUs, buses, networks, disks, etc.), the systems architect can then quickly evaluate the viability of arbitrary system configurations. We provided examples of that showed how the scheduling models could be used to decide which bus arbitration scheme or LAN was most appropriate for a specific application.
- **Expose the system overhead costs.** The scheduling models directly expose the system overhead costs associated with supporting various application functions. By exposing the costs of all system functionalities, the system architect can then make sound engineering decisions as to whether he can afford the additional functionality for his application.
- **Facilitate system resource partitioning and management.** The scheduling models provide the right level of abstraction for the System's Engineering organization to interface with the Software Development organization. Initially resource budgets can be assigned to the individual application programmers which specify limits on CPU time, synchronization requirements, device I/O etc. The Systems Engineering organization validates the system's performance relative to these initial budgets. As the development cycle proceeds and actual resource requirements become available, the System Engineering organization can then check to make sure that the system level timing performance is maintained. In this way the Systems and Software Development organizations can work together and have confidence at any point in time that the current baseline system is feasible.
- **Quantitatively evaluate hardware/software boundary issues.** Systems Engineers for high performance embedded systems are often required to evaluate the cost/performance tradeoffs associated with special purpose hardware. Mraz [4] used a set of scheduling models as an initial

evaluation vehicle in his work developing a RISC-Based architecture for real-time computation. He developed scheduling models for a conventional CPU/OS pair, a processor supported by an operating system coprocessor, and dual threaded RISC processor that injects the operating system as a non-interfering execution stream in application pipeline stalls. Using these three models he was able to quickly quantify the expected gains associated with the differing approaches.

- **Explore the impact of new technologies.** The science of building scheduling models for arbitrary technologies is maturing. Thus, we expect to be able to fairly quickly develop consistent scheduling models for any emerging technologies. For example, there is currently a lot of work in packet switching networks. Although, we have focused most of our effort on shared media LANs, we believe that we can readily model most packet switching networks.

- **Optimize system configuration parameters.** One of the most important capabilities that scheduling models provide is that they allow the System or Software Engineers to optimize the system configuration parameters.

The above work represents a solid first step to moving real-time systems development from an art to an engineering science. However, there is still much work to be done, such as folding in the work associated with jointly scheduling aperiodic and periodic tasks [5, 6]. The System Engineering Workbench needs to be completed. The capability to address fault tolerant requirements and provide graceful degradation properties also needs to be incorporated.

References

[1] D. Katcher, J. Lehoczky, and J. Strosnider, "Scheduling models of dynamic priority schedulers," *CMU/ECE Technical Report*, November 1992.

[2] M. Harbour, M. Klein, and J. Lehockzy, "Fixed priority scheduling of periodic tasks with varying execution priority," *Proceedings of the 1991 IEEE Real-Time Systems Symposium*, vol. 1, pp. 116–128, December 1991.

[3] E. Snow and J. Strosnider, "Implications of overhead and imperfect preemption on real-time scheduling models," Tech. report, Electrical and Computer Engineering, Carnegie Mellon University, 1993.

[4] R. Mraz, *A RISC-based Architecture for Real-Time Computation*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, 1992.

[5] B. Sprunt, *Aperiodic Task Scheduling for Real-Time Systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, August 1990.

[6] J. Strosnider, J. Lehoczky, and L. Sha, "The deferrable server algorithm," *To appear in IEEE Transactions on Computers*, 1993.

On the Structure and Dynamics of a Deeply-Integrated Information System

Bruce I. Blum

Johns Hopkins University/Applied Physics Laboratory
Laurel, MD 20723-6099
bib@aplcomm.jhuapl.edu

This paper examines the structure and dynamics of a large, complex information system that has been in operational use for more than a decade. The paper begins by explaining how the study of this system is related to the development of complex systems. It then introduces the modeling philosophy used in the system's design, provides an overview of the system's architecture, and evaluates 13 years of experience with its development and maintenance. It is shown that, even though the organization of the system defies all current standards of good practice, the resulting system is reliable, maintainable, useable, and inexpensive to support. The paper closes with some questions about the true nature of the conduct of systems engineering for complex systems.

Introduction

For the past half-dozen years, my research has focused on the nature of the software process and how we may best implement and control it. To me, software engineering (which is the study of the conduct of the software process) is a form of systems engineering. Moreover, with our growing reliance on software tools for design, test, and manufacture, software engineering is coming to provide an excellent model for equipment fabrication. Thus, even though the topic of this meeting is "complex system engineering," it is reasonable to accept a large software system as a specialized instance from the target class in the expectation that it will provide insight into the role of automation in the development of the large, complex, fault tolerant, distributed, real-time, time-critical systems of the future.

This paper examines the 13-year history of a large, complex, distributed, clinical information system [1]. The system is of special interest because it was developed and maintained using an environment that hides most of the implementation details [2]. The result is an architecture that is very different from those employed in typical information systems. The system also is of interest in this conference because it is used to make life-threatening decisions, it must be fault tolerant, and its data must be timely. Of course, the technology used for an information system make these last properties much easier to implement than they would be for

This work was supported in part by the U. S. Navy, Space and Naval Warfare Systems Command (SPAWAR) under contract N00039-91-C-0001, task VMAR9 with the Office of Naval Research (ONR) and the Naval Surface Warfare Center (NSWC).

a Navy tactical system. Thus, this paper does not suggest that experience in a medical setting is directly transferrable to a military application. Rather, the paper argues that if characteristics of this clinical information system seem to violate conventional wisdom, then perhaps our perceptions regarding those characteristics were poorly founded. It would follow that, if we must adjust our understanding of the structure of information systems, then perhaps we also should alter our conventions regarding complex, real-time tactical systems.

On Integration and Modularization

Elsewhere I have argued that software development is a modeling activity [3]. The models range in granularity from how a computer-supported product can resolve some domain need (i.e., the requirements) to how the computations should be carried out (i.e., the source code). An information system design is based on a model of some external reality. Surrogates for selected entities in that reality will be maintained as data in the system's database. Because the external reality is deeply-integrated, the database should reflect its integrated character. In many cases, however, the difficulty in implementing a system that operates within a complex environment forces the designers to abstract away many of the important properties (and complexities) of the external environment. This reduces risk and improves the processing of selected functions. Unfortunately, the result is a distortion of the design model, which restricts it to a segment of the real world model. As a consequence, it is difficult to extend system-supported activities beyond the boundary imposed by the constrained system model.

For example, a clinical system may ignore all attributes of a patient other than the patient's schedule. The resulting patient scheduling system will use some common patient identifier so that it may link patient schedule activities with, say, billing activities. Furthermore, an integrated patient scheduling system will associate providers (e.g., doctors, nurses) with the patient visits, and they may even maintain individual provider schedules including normal office hours, holidays, and other absences. Yet, even with a highly complex appointment system, the basic activity being modeled is that of scheduling; there will be little about the system that is unique to a health care setting. In fact, the basic system would be equally useful for a beauty salon, if only the cost per visit were high enough to justify the investment.

The Oncology Clinical Information System (OCIS) was designed to assist the patient-oriented activities in the Johns Hopkins Oncology Center. The model used to design OCIS is derived from a model of the Center. Patients are treated as inpatients and outpatients; services are provided by physicians, nurses, laboratories, the pharmacy, and registrars; therapy is organized by protocol (both research and standard practice), and the therapy plans, status, and analysis must be documented; visits, admissions, and resources are scheduled; and clinical data are combined and displayed to aid decision making. This model is highly integrated. For instance, the patient record must contain a history of all clinical results, diagnoses, and scheduled activities; nursing information must indicate patient activities, individual schedules, projected staffing needs, and resource assignments. In fact, the complexity of the model precludes its complete definition. The model evolves as experience with OCIS builds; moreover, as the health care team becomes familiar with OCIS, use of the system alters the real world model of Center operations.

I believe that deep integration and evolving understanding are characteristics of all large-scale, complex systems. When the complexity exceeds our ability to manage it, or when our understanding does not permit us to approach the complete problem, we decompose the problem by breaking it down into smaller, more manageable components. The principle of information hiding is used to abstract away details external to the component of interest. As experience accumulates, new uses for the modularized components are recognized and new decompositions are experimented with. This, of course, is how hardware has evolved. Simon suggests that this hierarchical approach is the essence of the design process [4]. Yet the jumble of the previous paragraph suggests no such structure. Everything seems to be connected to everything else; there is no natural order. Imposing an order by decomposition would introduce interfaces that serve to disintegrate.

The question is, can we develop a highly integrated design that evolves as our understanding builds, and—if yes—can we implement and maintain that design? This paper asserts that the OCIS project experience answers both those questions in the affirmative. How this has been accomplished has been documented elsewhere. What is important here is to recognize that it can be done. Just as Bannister opened the way to a series of sub-four-minute mile records, I would hope that the OCIS example would motivate others to examine this approach and exceed our accomplishments.

Overview of the System

In its present configuration, OCIS operates on a network of 5 computers, maintains a distributed database, and supports 250 terminals located throughout the Center. The database provides online access to some half-million days of care for the 20,000 cancer patients treated at the Center. The programs comprise a million lines of code, but this code is generated from compact specifications. In early 1992, the OCIS design consisted of specifications for 9257 programs and a data model comprised of 2273 relations (called tables) and 3823 attributes (called elements). The program specifications are very compact, averaging 15 lines in length. The functionality delivered by a program specification is roughly the same as that provided by a 300-line program in a more verbose language such as COBOL. Examples of program functions are the management of a menu (together with help messages and error checks), the processing of a request for a report, the listing of a report, and a reasonably complex computation involving data retrievals. OCIS was first installed in 1976; this version of the system was reengineered from the original system starting in 1980, and it has been in continuous operation since 1983. Thus, the current implementation of OCIS is a large and complex information system that has been used for a decade in supporting life-threatening decisions. The remainder of this paper examines the structure of that system and how the structure has evolved.

Unlike the organization of a system whose design was guided by decomposition, the structure of OCIS is best characterized as holistic. The specification is maintained in an integrated database (called the *application database*, or ADB) in which the elementary items are stored as *fragments* [5]. A program generator operates on these fragments to produce the executable programs. (A program specification may cause the generation of more than one executable program.) Examples of higher level fragments include program specifications, relation (table) definitions, and data structures composed of multiple tables. Within the ADB there is no concept of either module or file. A program specification may reference tables and types

(e.g., attributes or elements) without explicitly "including" them. In other words, all knowledge within the ADB is shared by all components of the ADB. The program generator organizes the ADB contents into modules for execution.

By way of contrast, a procedure-oriented implementation builds modules that define the procedure, and it uses data structures to support the processing. The process has no access to knowledge not explicitly specified or included within the module. An object orientation uses a different modularization technique in which the module is organized around the data type; operations on the data type are appended explicitly, and data type attributes in the inheritance chain are included implicitly. In both these module implementations, the module serves to hide details, and one of the benefits of the module formalism is its application independence, which is seen to foster reuse. With the environment used to generate the OCIS programs, on the other hand, modularization is viewed as an implementation (and not a design) concern. Thus knowledge of how to transform a design into an implementation is reused, and the system design (i.e., the ADB) is organized as a single, coherent unit. (The ADB can be organized into families of applications, and segments of applications may be copied [6].)

How well does this approach work? The design of the present version of OCIS began in 1980. Since that time, productivity has averaged one production program per effort day. In a 1992 analysis, the net productivity was .7 production programs per effort day (computed by dividing the number of specifications in the production system by the number of effort days expended on the project since 1980). All maintenance and new development is the responsibility of a staff of six. During the past 5 years, that staff has added new functionality to OCIS at the rate of 20 programs per week. The number of program specifications in the design has increased by a factor of 2.5 since the system was installed in 1983. Most of these programs represent functions not available in the initial implementation; some of the new functions augment or replace system features that were poorly understood at the time of original installation. Thus, the design of OCIS has evolved as the users' understanding of its operation and potential matured [7].

The claim just has been made that the design of OCIS reached its current state in response to a changing understanding of its objectives. That is, as the model of the real world adjusted itself to take advantage of the facilities provided by OCIS, the design model of OCIS reacted to accommodate the modified needs. Because this operational experience took place over a ten year period, one would expect to find that changes would take one of two forms:

> The addition of isolated, modularized features. Here a new requirement is identified, and a set of programs is specified to meet this requirement. The new function is relatively independent of the remainder of OCIS; exchange is managed through well-defined interfaces. Most extensions to traditional architectures are of this type.
>
> The editing of existing features. In this case, new requirements are met by altering the existing functionality and adding new programs as necessary. The resulting design cannot differentiate between what previously existed and what was just added. The old and new are deeply integrated, thereby reflecting the structure of the external reality served by the system.

I now demonstrate that the growth of OCIS is an example of the second form.

Demonstration of Integration

Table 1 contains a matrix that counts the number of program specifications by the year of their initial definition and last editing. If the system additions were highly modular, then one would expect to find few program specifications edited after the year of their initial definition. Yet the table presents a very different pattern. In the 13 month period from January 1, 1991 to February 2, 1992 (the date the data were collected), a total of 4,094 programs were brought into the editor. Of these, 1,049 programs were new programs defined during that period and the remaining 3,045 programs were edited to accommodate the new programs. That is, in a 13 month period 44% of programs in the 9,257-program system had been either edited or added by a staff of six full-time equivalents. (In an 1988 study, 1,084 programs were added and 2,170 programs were edited during an 18 month period, which meant that 49% of a 6,605-program system had been altered.) By way of a side comment, 44% percent of the program edits were made by someone other than the original designer, and the median number of program edits over a 13 year period was 11. Thus, the design team did not experience undue difficulty in working within this degree of integration.

Year	Year of last update											
Defined To	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	Total
To 1982	454	173	111	68	123	81	157	118	132	416	171	2,004
1983		204	156	48	63	58	64	41	75	188	63	960
1984			323	70	69	90	43	44	76	235	47	997
1985				115	97	48	39	34	65	259	52	709
1986					250	131	103	61	100	215	57	917
1987						186	87	72	83	201	73	702
1988							112	30	74	140	51	407
1989								208	156	288	86	738
1990									271	363	140	774
1991										643	254	897
1992											152	152
Total	454	377	590	301	602	594	605	608	1,032	2,948	1,146	9,257

Table 1. Distribution of OCIS programs by year defined and year last updated.

Table 2 provides another view of the integration of OCIS. It summarizes the relative ages between a program and the programs it calls, the tables it reads, and the tables it writes. As can be seen in the table, about 20% of all references to a program or table are to items that were defined at least six months after the referencing program; moreover, half of those references are to items defined 2 or more years after the referencing program's initial definition. Thus, older programs are edited (e.g., a menu program is modified to provide access to a new feature), and new capabilities are integrated with those that already exist.

Figures 1-3 depict the same data in a historical context. The relative ages of the referenced items are grouped by the year that the referencing program first was defined. The

Number	Years					*Months*			Years	
	2 <	2-1	1-½	*6-1*	*±1*	*1-6*	½-1	1-2	> 2	Items
Called Program	14	5	5	7	39	8	5	6	10	14,285
Read Table	29	9	8	8	20	6	4	4	12	13,898
Written Table	14	8	7	9	33	7	5	4	13	5,919

Table 2. Relative ages of associated OCIS items as percent of total.
(Date of program definition relative to the date of referenced item definition).

items are organized into five categories: those defined at approximately the same time (± 6 months), those between 6 months and 2 years (before and after), and those more than 2 years (before and after). References to items defined before the program may be thought of as a kind of reuse of existing system resources; references to items defined after the program was defined represent retrofits for integration. The time period covered may be divided into three activities: 1980-1983, development of OCIS; 1984-1986, evolution with constrained resources during user orientation; and 1987-, mature evolution. Naturally, there were no older items to be referenced during the first two years of development, and there can be few retrofits for the newer programs. The data provide the following insights into the structure of OCIS and its evolution.

Relative ages of programs called (Figure 1). Although most program calls are to programs defined at about the same time, a relatively large number of programs in the original OCIS system (1983 and before) were updated to reference programs defined significantly later. In fact, approximately 20% of the calls in the original system are to programs written at least 2 years after the calling programs. Once OCIS is in production, there tends to be significant use of "utility" resources that are part of the existing baseline (e.g., the calls to programs defined two or more years earlier). Nevertheless, in the mature period, the use of older utilities is almost balanced by the retrofitting of existing programs.

Relative ages of tables read (Figure 2). As one might expect, the data show that there is a strong tendency for new programs to read existing tables. The database is an OCIS resource, and new programs ought to combine existing data with the data defined for the new features. (For example, new patient-oriented functions will reference the patient name, which was defined as part of the initial OCIS increment.) It is not intuitively obvious, however, that the definition of new data structures will affect older programs. Yet, the data for the development period show that many of the baseline programs were modified to read data defined considerably after the program (e.g., 23% of the 1983 system's reads were defined 2 or more years after the program). This retrofitting of programs to read newly defined data continues at a lesser degree throughout the life of the system.

Relative ages of table written to (Figure 3). In a highly compartmentalized system, one would find that the dates of definition for the tables and the programs that write to (i.e.,

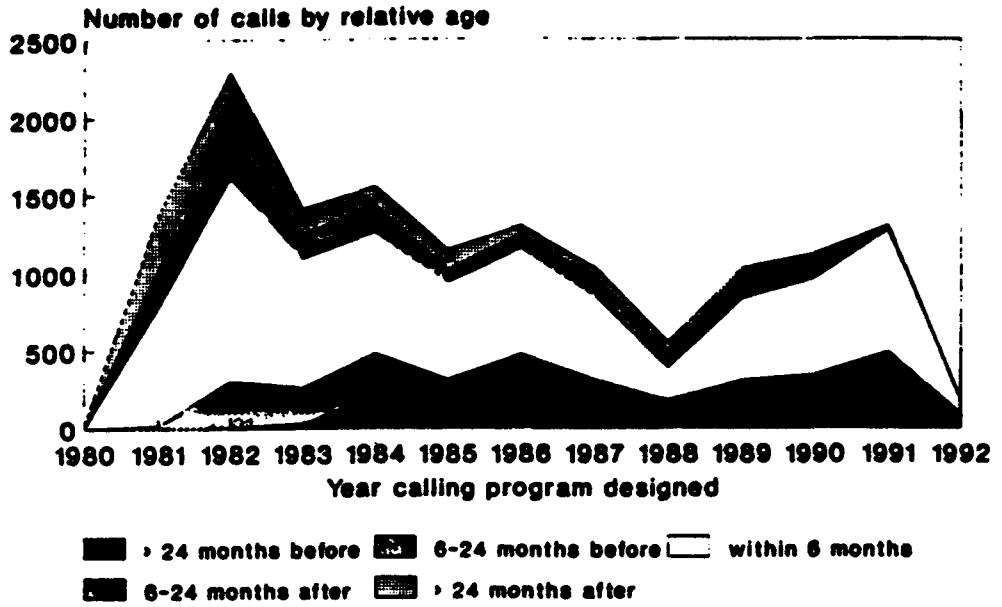


Figure 1. Relative age of programs called by date of calling program definition.

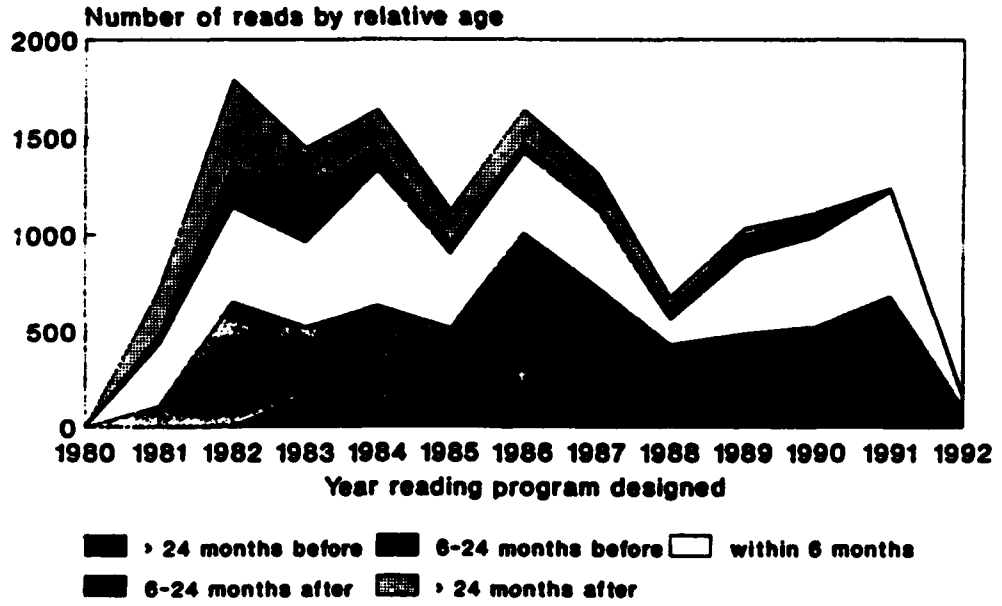


Figure 2. Relative age of tables read by date of reading program definition.

update) them are roughly the same. (For example, in an object-oriented environment, the dates of object and method definition would tend to be the same). Yet, as shown in the figure, one fourth of the original system was modified to update tables defined 2 or more years after the program's definition. Furthermore, the data indicate that new programs often write to tables developed considerably earlier (e.g., of the writes in programs defined in 1986, 35% were to tables defined 2 or more years earlier).

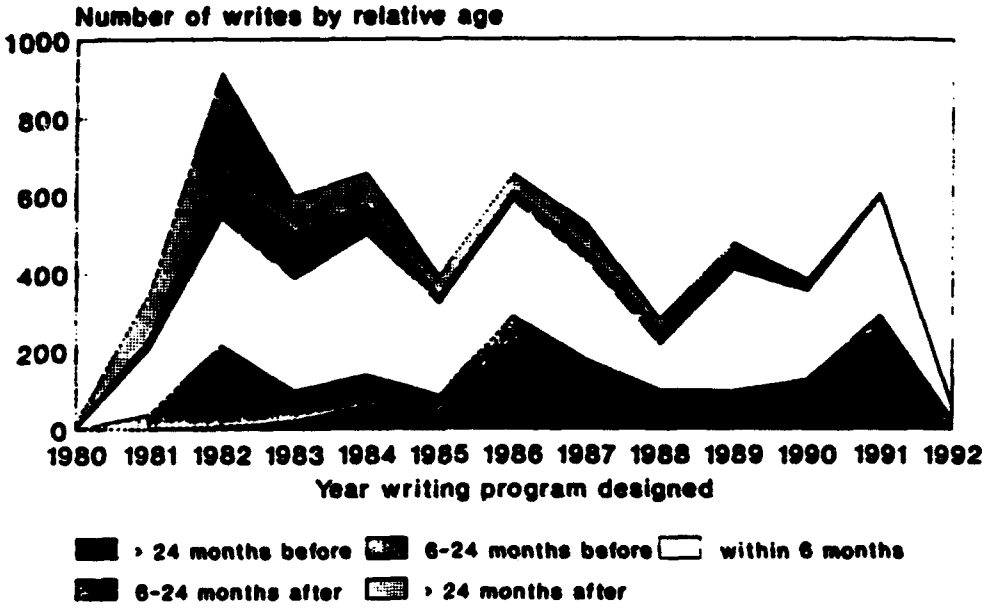


Figure 3. Relative age of tables written by date of writing program definition.

This analysis of the relative ages of the OCIS programs and the items with which they interact demonstrates that OCIS is a deeply integrated system. There are few changes that do not exploit existing system features, and few existing features are unaffected by the introduction of new capabilities. I assert that this degree of integration is implicit in an information system specification, but it is the lack of adequate support for development and maintenance that inhibits its realization. After all, if maintenance is difficult and expensive, then it would be prudent to localize change and establish clear module boundaries.

Conclusion

This paper has shown that there is at least one large-scale, complex information system that has operated effectively in a critical environment *and* that exhibits a non-modular, deeply integrated design. By conventional standards, the amorphous nature of the system architecture would be considered a poor design; one that would be very difficult to maintain. Yet the history of this project is one of high productivity, easy maintenance, and flexibility in responding to changing needs. Based in the widespread difficulty exhibited by other organizations in maintaining clinical systems, I do not believe that the success of the OCIS project is related to the specific domain in which it operates.

In much of my work I have tried to distinguish between the modeling of the software system intended to meet a need and the modeling of the implementation that will satisfy that need. Two very distinct classes of model are involved, and if we begin with the wrong model, then we will be forced to structure our reasoning within the context of that model. Modeling a problem domain should be holistic when we are concerned with complex, integrated problems; modeling the solution's implementation will be modular. Our heritage of hardware development has taught us to use decomposition to reduce large problems into smaller, more solvable

problems. This approach was reinforced by the early models of human information processing. Newer, connectionist models recognize the parallelism that permeates the universe. Computer architecture is moving to exploit this parallelism; the emerging concept of concurrent design is another of these transitions from a serial to a parallel process.

The point of this paper, therefore, is that perhaps we are looking at the problem the wrong way. For many years we have imposed a hardware-oriented discipline on software production, and the results have been satisfactory. Clearly, we have accomplished a great deal. Yet, with OCIS, an apparently chaotic organization also seems to work very well. Our challenge is to understand why it works, so that we can apply its lessons to the tactical systems that the Navy needs. That, of course, is the primary focus of my research.

References

[1] J. P. Enterline, R. E. Lenhard and B. I. Blum (eds), *A Clinical Information System for Oncology*, Springer-Verlag, New York, 1989.

[2] B. I. Blum, *TEDIUM and the Software Process*, MIT Press, Cambridge, MA, 1990.

[3] B. I. Blum, *Software Engineering: A Holistic View*, Oxford University Press, New York, NY, 1992.

[4] H. A. Simon, *The Sciences of the Artificial*, MIT Press, Boston, MA, 1969.

[5] B. I. Blum, Representing Open Requirements with a Fragment-Based Specification, *IEEE Trans. Sys., Man, and Cyber.*, (in press).

[6] B. I. Blum, The Fourth Decade of Software Engineering: Some Issues in Knowledge Management, *Int. J. Intelligent and Cooperative Information Systems*, (in press).

[7] Blum, B. I., The Dynamics of a Clinical Information System, *MEDINFO 92*, pp. 168-173, 1992.

INTEGRATION COMPONENTS, SPACES, AND CELLS

Jeffrey O. Grady
General Dynamics Space Systems Division

Setting the Stage for Integration Decomposition

The specification, concept development, design, and verification activities for complex systems involving hardware, software, and human action requires intensely cooperative work among many talented specialized engineers none of whom are capable of understanding the total problem that the system must solve in complete depth across the complete system. Specialization carries with it the need for integration of the many small problem solutions into the total system solution.

The rebirth of the systems approach in the form of IPD or concurrent engineering stumbles in many companies due to failure to upgrade communication techniques and integration processes to the needs of the IPD environment. These two failures are inter-related and must be unraveled by clear lines of authority and responsibility for the teams focused on the organization of the product and a clear and universal understanding of what system integration, the subject of much of the communication that must take place, is.

It would be convenient if we could describe all of the facets of system integration in terms of a single entity. The author is convinced, after years of work, observation, and study in this field, that many of us accept incorrectly that integration is a single activity. Few of us are able to describe how that single activity is performed, however. Many people assign the term "system integration" a mystical quality. Whatever it is, it is the answer to every system engineering problem and seemingly just connecting the term in the same sentence with a problem is sufficient to define an approach to the integration process in our proposals and conversations. We will see in this paper that the integration process can be decomposed into several parts and these parts explained in an uncomplicated way. System integration then becomes the sum of those parts.

We begin with an acceptance that an engineering organization that must deal with multiple system development activities, should organize in a matrix structure. The matrix has the advantage of focusing day-to-day work on specific program problems while providing a good environment within which to improve the organization's skills, methods, tools, and knowledge through continuous process improvement. We should also accept the good sense that we cannot beat the odds on specialization. Our engineering organization must select its personnel from the same pool as everyone else, humanity. We humans are knowledge limited and we solve the problem caused by that limitation through specialization.

Therefore, we will organize our personnel into functional specialties (departments) led by functional department Chiefs. These Chiefs will be responsible for providing all of our programs with qualified personnel, skilled in using a particular toolset and following the standard department procedures proven effective on past programs. We insist on standards, that are continuously improved, because we wish to take advantage of the practice-practice-practice template used by great athletes.

On a given program the product will be organized into sub-elements that can be worked on by one or more Integrated Product Development (IPD) Teams and we will assign our personnel to these teams which will form the principal personnel supervisory structure within the program. The work of all of the IPD Teams on a given program will be coordinated by a System Engineering & Integration (SEI) Team.

We will organize all program work into process steps linked to product entities under the responsibility of one or more IPD Teams. Each process step will have a set of goals and simple task description. We will map our IPD Team responsibilities to the processes and identify leaders for each process. All of the processes will be laid into our integrated schedule with clear start/stop dates and budgets which reflect back into the IPD structure. Each process will have clearly identified information and/or material product outputs that are needed in other processes as inputs. Also, each process step will have associated with it a simple criteria by which it is possible to determine when the task is complete in the form of a completion criteria.

Integration Components

These assumptions and selections leave us with the problem of combining, or integrating, the work of many people in different functional disciplines, working on different product system components, in many different process steps over time. We define these three fundamental integration components as function, product, and process respectively. Be very careful that

you understand that we have used the word function here to mean functional organization and not product system function.

In each of these components, we have to concern ourselves with two fundamental kinds of integration: co-component and cross-component integration. In addition, we have to account for the possibility that one or two of the components is not involved. This means we have a three-valued situation: co, cross, and null values. With three variables, each with three values, it is obvious we are working with 27 (3 cubed) different integration possibilities. It is helpful to have a picture of this problem to better understand all of these possibilities.

Unfortunately, we can't easily illustrate the three-valued relationship for each of our three components, so our visual model will disregard the null possibility for each component. Figure 1 illustrates the three components as a three dimensional system. We can imagine that we can assign positions on the function axis to discrete organizations in our functional organization such as reliability, structural design, quality assurance, etc. Similarly, we can assign positions on the product axis to the elements of the system (avionics system, on-board computer, circuit card, transistor, etc.) in a hierarchical fashion, Finally, the third axis can have positions marked out corresponding to the processes on our program process diagram, such as identify item X requirements, design item X, and test item X.

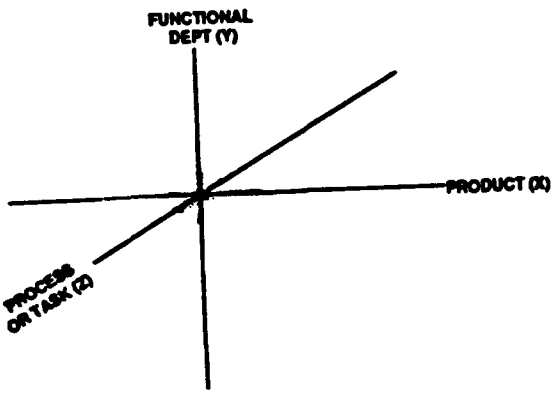


Figure 1 Integration Components.

For a given functional organization, all of the work that a specialty discipline does can be thought of as being within the plane passing perpendicular to the function axis at that function position. The task of integrating all of the work on that plane is called co-function integration, the integration of work accomplished by two or more specialists in that one functional discipline. The other two variables have null values in this limiting case.

Similar planes can be constructed perpendicular to the other two axes at different points on those axes to capture all of the co-process and co-product integration work. The careful reader will observe that every point in the three-space, then, corresponds to some combination of co and cross component integration for the three components.

Let us now define the three integration components in terms of their possible values. Three combinations are nulls, meaning no integration for that component: Null-Function, Null-Process, and Null-Product integration. The other six possibilities are co and cross combinations with each of the three components. Let us take each of these six cases in turn assuming, in each case, that the other two components have a null value for the moment.

Co-Function Integration coordinates the work of two or more persons from the same functional department (specialty discipline) to ensure they are all using the same tools, techniques, and procedures in an appropriate fashion and that their results are consistent with other work on one or more programs or systems. This task is the responsibility of the senior program functional specialist for the project or the functional supervisor, in the case where all company projects are the target of the integration work.

Cross-Function Integration coordinates the work of persons from two or more functional disciplines in search of sub optimal design and specialty engineering solutions needing re balancing, mutual conflicts between specialty requirements or the corresponding design solutions, available unused margin to be repossessed and applied more effectively, and wayward interpretations of the project or product requirements that may lead to conflict. This is a program responsibility that may fall upon an IPD Team Leader, task principal, or person from the System Engineering & Integration Team as a function of the relationship of the work to process or product and the integration level.

Co-Process Integration coordinates the work of two or more persons working in the same program process to ensure that they all are focusing on the same process needs within available budget and schedule constraints. This task is accomplished by the assigned task leader who is responsible to achieve the task goals on time and on budget.

Cross-Process Integration coordinates the work of two or more task teams working different processes. It seeks to

ensure that the work of the two or more teams is mutually consistent and driven by the same program goals. This integration task is the responsibility of a program person

Co-Product Integration coordinates the work of two or more persons developing the design solution for a particular product item. This may be to develop an integrated set of product item requirements, evolve an optimum synthesis of those requirements in terms of a design concept, ensure that all of the cooperating specialty views are respected in the design solution, or integrate the test, analysis, and design work associated with that product item. Where an item requires a special test article, such as a flow bench for a fluid system, this work should also ensure that the test article properly reflects the same requirements and design embodied in the product item.

Cross-Product Integration is the commonly conceived integration component that most people would first think of in response to the word integration. It focuses primarily on integration of interfaces between product items to ensure that all interface terminals and media are compatible both physically and functionally. This work could be as simple also as ensuring that all items are painted the correct color, have satisfied a particular specialty requirement as in being maintainable as defined by an ability to remove and replace in 10 minutes. It is difficult to talk about this form of integration without linking other integration components.

Integration Spaces

Integration work seldom falls into one of these pure cases. Commonly we have to deal with more complicated situations involving combinations of two or three components with mixes of cross, co, and null values. Figure 2 offers four particular examples of these possible combinations, or spaces, for further discussion. Once again, we disregard the null value case to enable simple graphical portrayal in three dimensions on two dimensional paper.

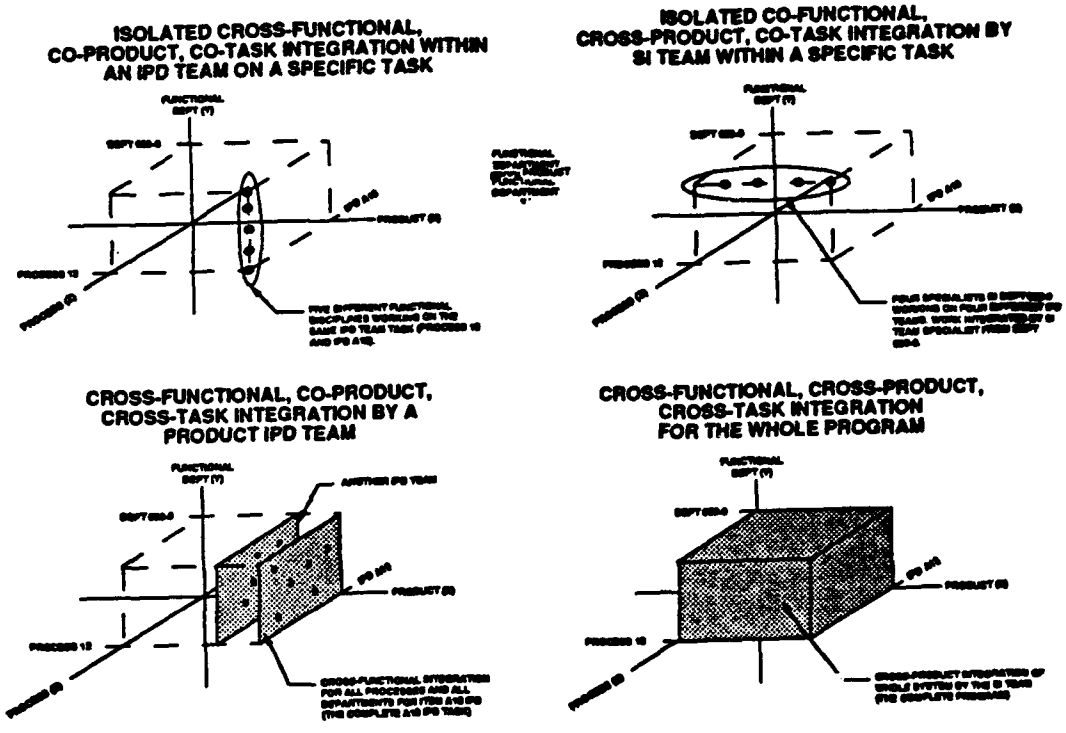


Figure 2 Integration Space Examples.

In Table 1 we use a simple tertiary counting scheme to ensure we have not omitted any combinations of the three variables. There can be no other forms of integration work than those listed in Table 1 given that

we have included every appropriate integration component. It would be possible to add one or more components to our list in addition to function, process, and product. The effect would be to multiply the number of different integration spaces. The number of integration spaces (S) is predictable as follows:

$$S = C^n,$$

where:

C = the number of integration components and

n = the number of values for each variable.

Table 4-1 Integration Space Identification.

ID	FUNCTION	PROCESS	PRODUCT	INTEGRATION TYPE NAME
0	NULL	NULL	NULL	INDIVIDUAL EFFORT
1	NULL	NULL	CO	ISOLATED CO-PRODUCT
2	NULL	NULL	CROSS	ISOLATED CROSS-PRODUCT
3	NULL	CO	NULL	ISOLATED CO-PROCESS
4	NULL	CO	CO	CO-PROCESS & PRODUCT
5	NULL	CO	CROSS	CO-PROCESS/CROSS-PRODUCT
6	NULL	CROSS	NULL	ISOLATED CROSS-PROCESS
7	NULL	CROSS	CO	CROSS-PROCESS/CO-PRODUCT
8	NULL	CROSS	CROSS	CROSS-PROCESS & PRODUCT
9	CO	NULL	NULL	ISOLATED CO-FUNCTION
10	CO	NULL	CO	CO-FUNCTION & PRODUCT
11	CO	NULL	CROSS	CO-FUNCTION/CROSS-PRODUCT
12	CO	CO	NULL	CO-FUNCTION & PROCESS
13	CO	CO	CO	ALL CO
14	CO	CO	CROSS	FOURTEEN
15	CO	CROSS	NULL	CO-FUNCTION/CROSS-PROCESS
16	CO	CROSS	CO	SIXTEEN
17	CO	CROSS	CROSS	SEVENTEEN
18	CROSS	NULL	NULL	ISOLATED CROSS-FUNCTION
19	CROSS	NULL	CO	CROSS-FUNCTION/CO-PRODUCT
20	CROSS	NULL	CROSS	CROSS-FUNCTION & PRODUCT
21	CROSS	CO	NULL	CROSS-FUNCTION/CO-PROCESS
22	CROSS	CO	CO	TWENTY TWO
23	CROSS	CO	CROSS	TWENTY THREE
24	CROSS	CROSS	NULL	CROSS-FUNCTION & PROCESS
25	CROSS	CROSS	CO	TWENTY FIVE
26	CROSS	CROSS	CROSS	ALL CROSS

If, for example, we concluded that there should be five components instead of the three covered in this paper, and three values (co, cross, and null) for each component, we would have 5^3=125 integration spaces. As you can imagine this could get out of hand very rapidly making the description of integration more complex than the work itself. Three components and three values appeared to the author to be a good compromise between completeness and understandability.

Each integration space involves a merger of a unique grouping of three particular values for the components and represents one integration mode useful in one or more particular situations.

Of the 27 integration spaces, we can dispense with INDIVIDUAL EFFORT integration immediately. We must assume that a single individual is fully capable in their field of specialized knowledge and able to effectively apply this knowledge to a specific product item and in a particular single process. This is why we specialized, after all, to create a human task that is within the power of a normal, single, specialized individual to master. We assume that each specialist can carry on an internalized conversation with themselves and use the power of their specialized discipline to solve the small problems we have tried to frame in our decomposition efforts.

Similarly, the ALL CO integration space is usually not very interesting to a system engineer since it involves people from the same functional department performing work in the same process step, for the same system element.

One other fairly simple integration space to explain, though the most complex of them all in practice, is ALL CROSS integration involving cross everything. When a member of the SEI team integrates the work of several

members of two IPD teams developing the design of two different product elements and the work of a facilities engineer responsible for the factory that will assemble these two elements and a tooling designer responsible for the manufacturing equipment that will hold the elements during mating, we have an example of this kind of integration. The reader will be able to imagine several other cases of this integration space.

We have given examples now for nine of the 27 integration spaces: including the six isolated integration cases, INDIVIDUAL EFFORT (or ALL NULL), ALL CO, and ALL CROSS. This leaves 18 remaining to be explained with at least one example. Some of these 18, you can see from Table 1, are very difficult to name so we will simply use the ID number for a name.

Integration Cells

In almost any given system development work situation, we will find it necessary for some combination of the 27 integration spaces to be applied. Very little system development work can be accomplished in total autonomy today. This phenomenon appears because we have had to specialize very finely to master enough of the available knowledge base to be competitive. The number of these combinations is finite but quite large for a large development program.

We have many options in grouping all program work into unique combinations of integration spaces but the best way to do this is probably driven by the program tasks that would appear on a program task network. Each program task has associated with it some set of functional disciplines performing work on some particular combination of product elements. Many of these tasks, possibly most, will require some form of integration in the context of some combination of the 27 integration spaces defined above. Let us call any one of these combinations of task, product, and functional organization an integration cell.

The more finely we divide the overall program into tasks, the more unique integration cells we will have. The more finely we assign IPD teams to develop the system, the larger the number of integration cells. The more functional disciplines that must be assigned to the program, the more integration cells that will be necessary. At the same time, the larger the number in any of these integration components, the more simply we can describe each integration cell because they will consist of a less complex combination of integration spaces.

Program World Line

So, the integration process is more complex than we might have first imagined, composed of a finer structure than we might have thought. But the complexity does not stop here. We must not only apply each of these integration spaces well within the context of the integration cells defined for the program, but we must apply them in a pattern coordinated with the program schedule. Figure 3 illustrates this need by placing the integration spaces coordinate system on a program world line. Throughout program passage on its world line (network or schedule), appropriate specialists must apply the appropriate integration spaces to the planned integration cells in accordance with in integrated plan. To the extent that we can reduce aggregate program work by early identification and resolution of inconsistencies between product and process elements, we encourage success in system integration. Ideally, we should be able to do all work error-free on the first pass.

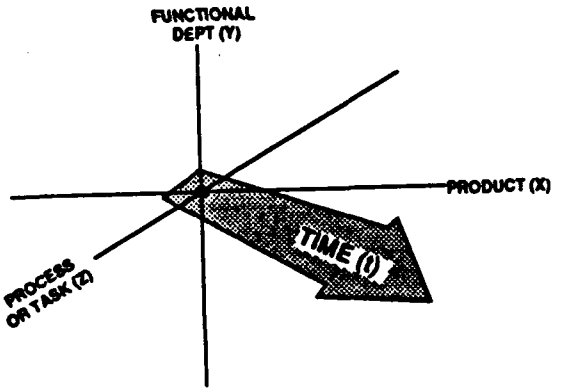


Figure 3 System Development World Line.

Now we can answer the question, "What is system integration?" It is the rich mixture of three integration components applied in combinations, defined by the resultant integration spaces, to the work confined to a finite number of integration cells across the program world line that actually comprises system integration on any given program.

Complex Systems Engineering Synthesis and Assessment Technology Workshop, July 20-22, 1993

An Efficient Approach to Systems Evolution (EASE)

Thomas C. Choinski, (203) 440-5391
John G. DePrimo, (203) 440-5723
Naval Undersea Warfare Center Detachment, New London, CT

"Today, we are in the final stages of a true "industrial revolution" in computer hardware and software which has totally transformed the industry. Navy has no choice but to adopt these changes. By leveraging the industrial revolution, Navy can reap great benefits in increased capability and decreased cost. If the Navy continues to develop unique product lines, it will likely face spiraling costs while falling further behind in capability."[1]

NRAC 91-1 Office of ASN RD&A

INTRODUCTION

Political changes throughout the world, fiscal constraints set by the United State Congress and a shift in corporate product planning have created the need for a new acquisition environment within the Department of Defense (DoD). The Director of Defense Research and Engineering (DDR&E) has set guidelines to direct these changes.[2] The guidelines consist of 7 Thrust Areas. The concept for transitioning technology described herein specifically addresses Thrust Area 4 (Undersea Superiority) and Thrust Area 7 (Affordability).

DoD must facilitate the process used to transition technology into warfare systems to achieve affordability objectives. No longer can DoD build warfare systems focused on delivering increased performance to oppose specific, pre-defined threats. Today's warfare systems must be able to adapt to unpredictable world situations similar to the Gulf War.

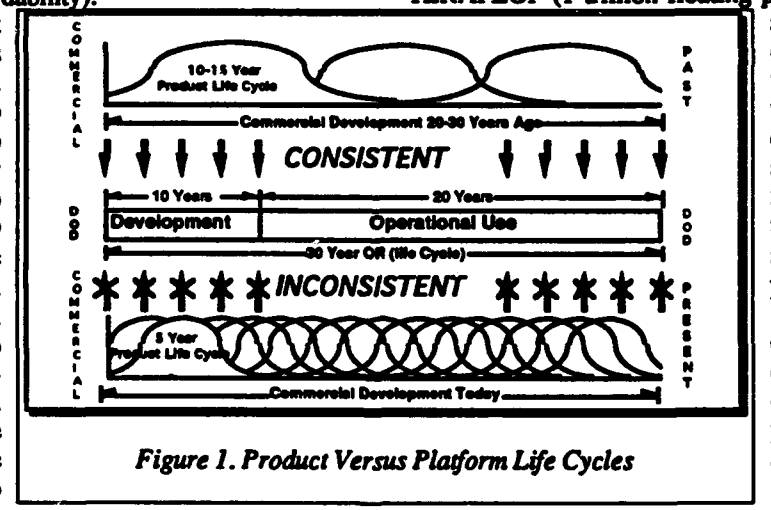


Figure 1. Product Versus Platform Life Cycles

The Efficient Approach to Systems Evolution (EASE) concept engendered a vision for developing warfare systems in the future. The concept seeks to make this vision a reality by capitalizing on key enabling technologies. EASE has set out to:

Foster the application of fast paced, emerging commercial technology to the development and acquisition of affordable warfare systems.

EASE will introduce new products and processes to transition technology in an affordable manner. The development of computer aided system engineering design tools and the strong emergence of high performance computing and communications (HPCC) technology, will make the transition feasible. This paper discusses these components further in the Vision and Technology Transition sections.

The HPCC Program, funded by the Advanced Research Projects Agency (ARPA) through the Federal HPC Program,[3] offers the potential for TERAFLOP (1 trillion floating point operations per second) computing capability. EASE will use HPCC technology as an enabling force to showcase potential, future alternatives to the military system acquisition process.

This paper outlines the EASE concept. The paper discusses: the three primary issues addressed by EASE, the new vision for transitioning technology, and the specific warfare system applications selected to showcase EASE.

ISSUES

The EASE concept focuses on three issues systems engineers confront when transitioning technology into warfare systems: commercial product life cycles, DoD policy, and the tradeoff between general commercial based processing and application specific (point technology) resources.

The first issue concerns the changing rate of new product development in the commercial sector. Twenty years ago the commercial sector distributed computer products with life cycles greater than 10 years. Digital Equipment Corporation's (DEC) VAX computer exemplifies this phenomena. DoD could transition products with 10 year life cycles easily, since the 10 year cycles minimized the number of technology upgrades for Navy platforms used for 30 years (Figure 1).

However, today's commercial computer companies have accelerated their time to market.[4] As a result, product life cycles have decreased to less than 5 years. In some cases, life cycles last less than 1 year. DoD has difficulty transitioning these rapidly emerging commercial products into warfare systems. Shortened commercial product life cycles require frequent technology upgrades. The current acquisition process can not keep up with the rate of change of technology.

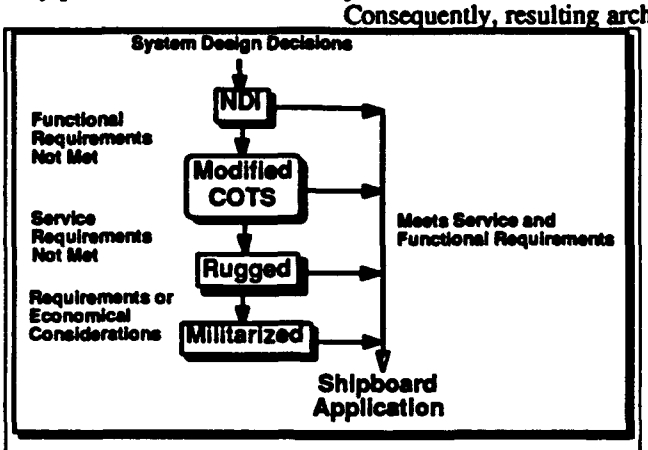


Figure 2. MIL-STD 2036 Policy

The second issue deals with DoD Policy. For example, the Navy has changed the policy concerning the utilization of commercial off-the-shelf (COTS) products. In the past, the Navy demanded militarization of equipment for shipboard use. The Navy granted waivers for using commercial equipment in military systems. DoD's aforementioned focus on affordability has reversed this policy. Accordingly, MIL-STD 2036 stipulates the requirement to use non-developmental items (NDI) and COTS. Program managers must justify using rugged and militarized equipment with an operational, service or economic consideration. This policy essentially compels a cultural change within the Navy. Figure 2 illustrates the process involved in MIL-STD 2036.[5]

The third issue addresses a trend in warfare system architectures. Analysis of warfare systems that have evolved over the last thirty years uncovers the trend toward the increased utilization of general purpose commercial based processors. Figure 3 illustrates this trend.

Warfare systems designed during the 1960's primarily used point technologies (i.e., special purpose designs) to achieve increased performance. During the 1980's, engineers designed warfare systems that incorporated commercial based processors, like Motorola's 68030; although, designers placed these processors on militarized circuit cards. Consequently, resulting architectures, consisted of an amalgam of commercial based technologies and special purpose designs. Once again engineers concentrated on performance.

The next step continues the trend towards general purpose commercial based processing technology. One alternative uses an emerging technology, like the one promoted by ARPA's High Performance Computing and Communications Program. In fact, the vision propagated by EASE uses HPCC technology as an enabling force.

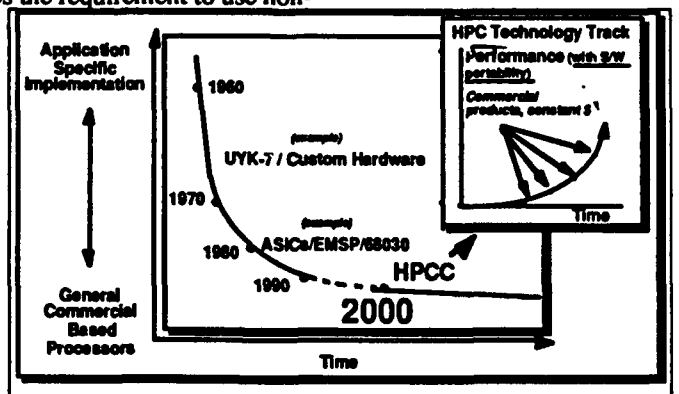


Figure 3. Commercial Based Processing Trend

VISION

The first step in achieving the EASE vision entails the creation of a "homogeneous" warfare system. HPCC technology offers one way to obtain this goal.

The homogeneous warfare system sets the stage for affordability. Figure 4 illustrates how each function in a warfare system (e.g., acoustics, communications, photonics, etc.) can use a generic HPCC based processor; hence, the homogeneous nature of the architecture.

Note, however, that heterogeneous processing resources do exist within a given function. For example, the acoustics function contains a receiver, HPCC computer and workstation. Similarly, HPCC technology also contains heterogeneous processing resources.

The homogeneous warfare system achieves affordability by using one software development environment for each function, and by minimizing the number of card types across the system. The warfare system also can reconfigure computing resources for low duty cycle functions. The applications section discusses reconfiguration.

General purpose commercial based processing fosters the transition of future technology. The warfare system can use either commercial or rugged equipment, but, land based development facilities incorporate commercial equipment. Scientists and engineers use the land-based facility to develop new algorithms and new functions while performing R&D. Under the scenario where only minor differences exist between the architecture in the laboratory and the system at sea, systems engineers can efficiently port the software to the warfare system. Utilization of a HPCC software development environment enables software portability into the warfare system described in figure 4. Figure 5 depicts this efficient process.

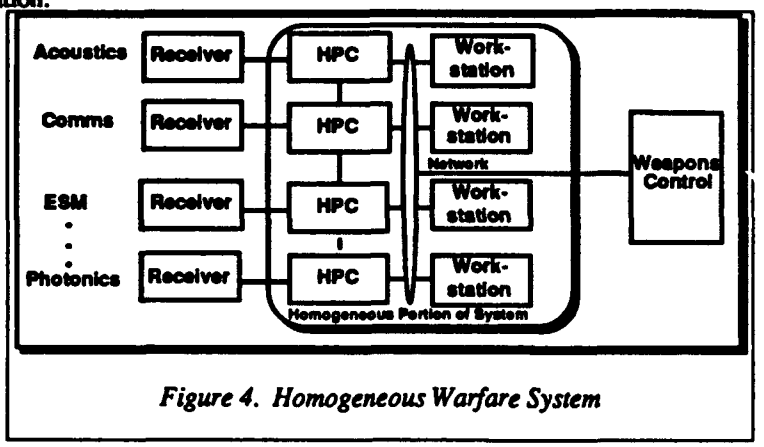


Figure 4. Homogeneous Warfare System

On the other hand, systems engineers can also transition new hardware and software advances from the commercial sector as shown in the lower right hand corner of figure 5. Succeeding generations of commercial product families can transition into the warfare system.

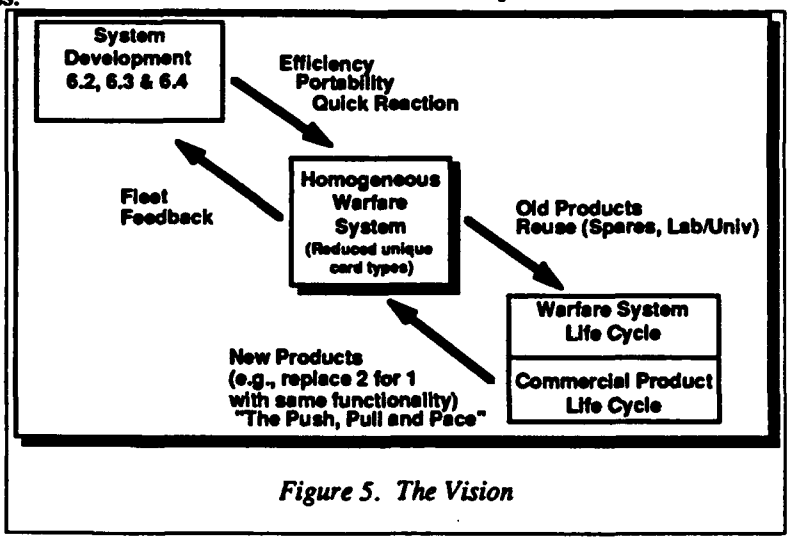


Figure 5. The Vision

The IBM PC serves as an analogy. IBM tied their personal computer to Intel's microprocessor family. The 80486 replaced the 80386SX; the 80386 succeeded the 80286. Over the life of the 486 product, Intel will introduce a host of derivative products, each offering some variation in speed, cost, and performance and each able to leverage the process and product innovations of the original product.[6]

The homogeneous warfare system provides the foundation to transition technology in the same sense as the personal computer analogy. However, transitioning technology demands more than an architectural foundation. Fiscal constraints mandate DoD's need to revitalize the transition process.

TECHNOLOGY TRANSITION

The DoD Research and Development Management Guide discusses the ***push*** and ***pull*** of technology.[7] The technology ***push*** involves advances in the state-of-the-art which occur in industry independent of emerging operational requirements. The technology ***pull*** encompasses advances in operational requirements which surface apart from the state-of-the-art in technology. The EASE concept introduces a third category into this paradigm, the ***pace*** of technology.

The ***pace*** addresses the rate at which the DoD can transition new ideas from the ***push*** and ***pull*** categories into the fleet. DoD needs to create an environment which simplifies the transition of emerging technologies. Figure 6 proposes a Technology Insertion Environment (TIE) to accomplish this objective.

Commercial trends suggest a 3 year transition cycle; in other words, technology should transition within 3 years of introduction from either the ***push*** or the ***pull*** categories. The 3 year transition period requires moving a warfare system product from the fuzzy front end of development to a well defined implementation decision. The fuzzy front end describes the beginning phase of product development. During this phase a

product requirements may be ambiguous. Moving to a well defined implementation decision will require the completion of the fundamental tradeoff analysis between development cost, time, life cycle cost and performance.[8] The Technology Insertion Environment expedites the tradeoff analysis.

The Technology Insertion Environment can transition research and development (R&D) that replaces older generations of technology. Likewise, the TIE can focus on technology fusion. Technology fusion refers to the process of combining technologies in a hybrid fashion.[9] This approach applies particularly to combinations from the *push* and *pull* categories, as well as hardware and software.

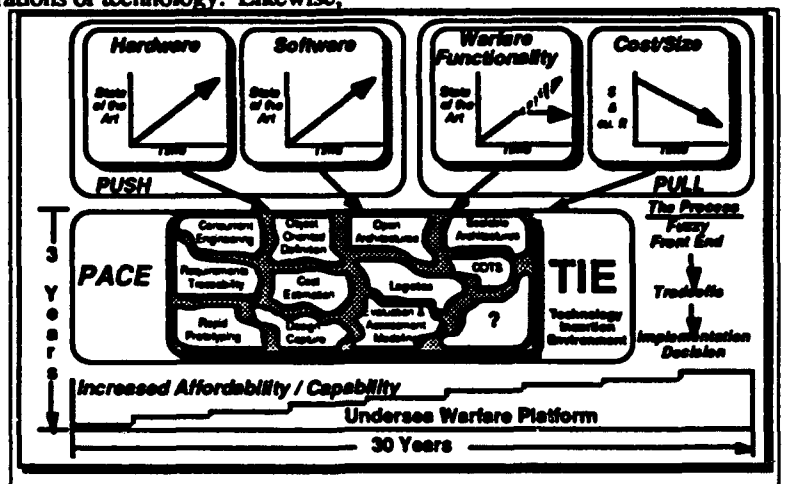


Figure 6. The Pace of Technology Transition

The picture in Figure 6 portrays the Technology Insertion Environment as pieces of a puzzle. These pieces provide the infrastructure to transition technology. The pieces include but are not limited to: logistics, concurrent engineering, rapid prototyping, requirements traceability and simulation. The EASE concept has identified numerous pieces to the puzzle, but has yet to determine the appropriate fit for the pieces. DoD needs to research this area to mature the concept. The Engineering of Complex Systems Block developed by the Naval Surface Warfare Center, and sponsored by the Office of Naval Research, has developed tools suitable for the Technology Insertion Environment. Successful technology transition depends on the Technology Insertion Environment.

The EASE concept provides a novel process for transitioning technology. Many products could showcase the EASE concept. The following section introduces two specific warfare system products which focus on affordability.

WARFARE SYSTEM APPLICATION

An assortment of warfare system applications could showcase the EASE concept. Figure 7 outlines the metrics used to guide the selection process. The EASE concept could have included (and ultimately may include) surface ship or air applications. Indeed, EASE has nearly universal military application. Exceptions include functions like cryptography which remain special purpose.

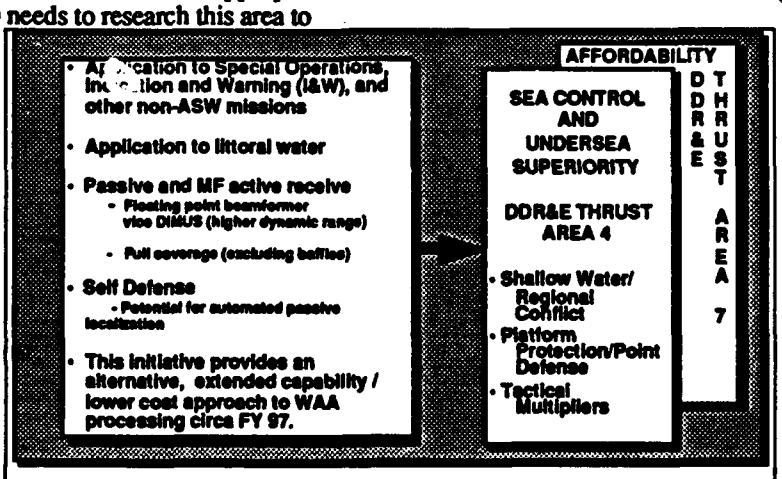


Figure 7. Application Selection Criteria

Examples of products considered include expanded TB-29 towed array processing. TB-29 focused beamforming and automated contact followers offer dramatic manning reductions relative to current processing techniques. This application would necessitate substantial processing throughput by concurrently processing thousands of sonar beams. ESM (Electronic Warfare Support Measures) presented another consideration, since its processing requirements are very similar to acoustics (i.e., A/D conversion, signal processing, and data processing). However, ESM's signal processing (e.g., FFT's, correlation, and feature extraction) require on the order of 100 MIPS. Some acoustic sensors require higher processing rates than ESM.

The processing required for the Wide Aperture Array (WAA) and photonics (i.e., periscope

processing) have optimum attributes to showcase EASE. These disparate, multi-GFLOP applications can demonstrate virtually any submarine or surface application through extrapolation. For example, one of three monochrome sensors in the ARPA developed non-penetrating periscope (NPP) forms an image 1024 x 1024 pixels by 10 bits/pixel, or about 1.3 Mbytes/image, and captures 30 images/second. Combined with advanced image processing algorithms involving enhancement and/or replica correlation, this massive quantity of data yields a 10 GFLOP minimum real-time requirement.

The WAA sensor consists of six panels of elements attached to the side of a submarine. The number of elements present a heavy computational load. Other reasons led to the selection of WAA besides the heavy computational load for beamforming and detection processing shown in Table I. For example, WAA has greater potential as a shallow water sensor than towed arrays. Towed arrays constrain the maneuvering and speed of a submarine to avoid dragging the array on the bottom. In addition, the frequency response of the WAA sensor is well "tuned" to emerging post-cold war contacts of interest. The implementation, sized in Table I, expands functionality: (1) full detection coverage excluding baffles in azimuth and D/E; (2) a floating point vice one-bit clipped ("DIMUS") beamformer to enable enhanced medium frequency active receive and improved detection in cluttered environments; and (3) interpolated detection beamforming to enable the addition of automated contact followers for reduced manning requirements. When referring these attributes back to the figure 7 application selection criteria, the reader will find close agreement.

Operation	Sustained	Efficiency*	Peak
Beamforming	7.50	32%	23.44
FIR Filters	0.54	25%	2.16
Complex FFTs	2.02	56%	3.61
Cross PSD	0.23	14%	1.64
Auto PSD's	0.16	14%	1.14
Integrate Auto Spectra	0.01	14%	0.07
Inverse Complex FFT	1.01	56%	1.80
Normalized XCOR/	0.25	14%	1.79
Upsample and Interpolate FIRs			
TOTAL GFLOPS	11.72		35.65

Table I. WAA Processing Requirements

NUWC and Intel engineers constructed the data in Table I. NUWC described the signal processing, and computed the sustained throughput requirements based upon sample rates, the number of beams, and the number of sensors. The sustained throughput requirement approaches 12 GFLOPS, most of which the beamformer consumes. Colleagues at Intel benchmarked the efficiencies. Intel executed "static" benchmarks to compute the efficiencies (i.e., computational nodes were not interconnected). Therefore, table I efficiencies do not account for operating system, I/O, and TADSTAND demands. NUWC's experience with massively parallel processing through the Advanced Sonar Signal Processor Architecture Project (ASPA) indicate systems engineers can expect to achieve 10% efficiencies in situ.

In any event, Intel did use FORTRAN instead of assembly language to benchmark their efficiencies. They used a high level language to point out the high level programming capability which lends itself to portability of application software. Figure 8 depicts an example involving "porting" applications from a commercial Intel Paragon to the Honeywell ruggedized unit with minimal rewrite of the software. The EASE concept includes the portability of software as a major feature.

The current Intel XP/S-35 has a peak throughpu. which matches the requirements outlined in Table I. Honeywell expects to increase processing density to 10 GFLOPS/cu. ft. in FY94, and to 100 GFLOPS/cu. ft. by FY98, through packaging and water cooling. With these processing densities: (1) the total size of a warfare system could shrink considerably; and (2) low processing efficiencies become less significant (e.g., 10% as stated above).

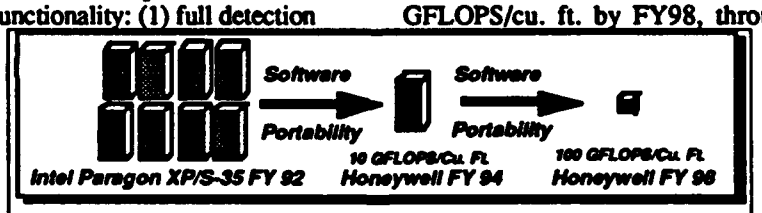


Figure 8. HPCC Packaging Payoff

The priorities of the Navy have changed, leading to a broad assessment of the future direction of United States' Maritime forces.[10] With the end of the Cold War, U.S. military strategy has shifted from deterrence of global conventional and nuclear war to the protection of vital national interests in regional crises, contingencies, and conflicts. While all elements of the Armed Forces have important roles to play, America's nuclear-powered submarines are a decisive component of the new military strategy. The submarine will help provide the flexible forward

presence and crisis response capabilities that have been the cornerstone of our national defense.[11]

The requirement for total warfare system flexibility provides the key to accommodating this evolution. Warfare system flexibility is the ability to provide mission enabling functional emphasis in those areas demanded by specific deployment assignments. Core requirements for ship's safety and self-protect require the applications in the left pie chart of figure 9. Shifting to an Indication and Warning (I&W) mission invokes specific requirements which could differ significant for example, from a traditional Cold War anti-submarine warfare (ASW) mission.

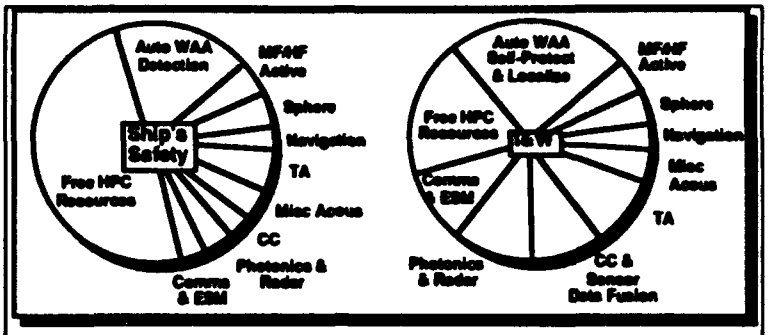


Figure 9. Warfare System Reconfigurability

The approach warfare system implementation provides the ultimate flexibility; the ability customize, through reconfiguration, the total available processing resources as mandated by its mission.

SUMMARY

The Naval Undersea Warfare Center has undertaken the EASE initiative to foster the application of fast paced, emerging commercial technology to the development and acquisition of affordable warfare systems. The initiative consists of new products and processes to foster the transition of new technologies into the fleet.

New processes center on the creation of a Technology Insertion Environment which will provide the tools to make technology transitions practicable. The environment will address concerns including: logistics, concurrent engineering, rapid prototyping, requirements traceability, and simulation.

To showcase the process, the EASE concept includes the development of two products using High Performance Computing and Communications technology. The Wide Aperture Array and the Non-Penetrating Periscope provide the applications for HPCC technology. These applications individually require more than 10 GFLOPS sustained processing throughput and together demonstrate the suitability of HPCC technology for a broad range of DoD real-time processing functions like image processing, beamforming, database management and traditional signal processing.

ACKNOWLEDGMENT

The authors would like thank Mr. Michael Amaral, the technical reviewer of this paper, as well as the people at the Naval Undersea Warfare Center who contributed to the EASE concept, especially Dr. John Short, Mr. William Coggins, Mr. Hal Watt, and Dr. José Muñoz.

REFERENCES

1 NRAC 91-1 Office of ASN RD&A, 1991 Summer Study on Open Systems Architecture for Command, Control and Communications, September 1991.

2 Director of Defense Research and Engineering, Defense Science and Technology Strategy, July 1992.

3 Office of Science and Technology Policy, Grand Challenges: High Performance Computing and Communications, A Report by the Committee on Physical, Mathematical, and Engineering Sciences to Supplement the President's Fiscal Year 1992 Budget, 1992.

4 Milton D. Rosenau, "Speeding Your Product to Market," Engineering Management Review, September 1989, pp. 27-40.

5 Department of the NAVY, MIL-STD 2036.

6 Steven C. Wheelwright and Kim B. Clark, "Creating Project Plans to Focus Product Development," Harvard Business Review, March-April 1992, p. 74.

7 Department of the Navy, RDT&E/Acquisition Management Guide, January 1987, p. 2-8.

8 Thomas C. Choinski, "Economical Development of Complex Computer Systems," Complex Systems Engineering Synthesis and Assessment Technology Workshop Proceedings, July 1992.

9 Fumio Kodama, "Technology Fusion and the New R&D," Harvard Business Review, July-August 1992, p. 70.

10 Department of the Navy, ... From the Sea, September 1992.

11 Department of the Navy, America's Nuclear Powered Submarines, Washington, D.C., 1992.

AN OVERVIEW OF THE PROCESSING GRAPH SUPPORT ENVIRONMENT

Roger Hillson
Naval Research Laboratory
Washington, DC 20375-5000

Abstract

The Navy has developed a data flow method for programming networks of processors. This approach, called the Processing Graph Method (PGM), is now being used to develop signal processing applications for the Navy's second-generation tactical signal processor. At the Naval Research Laboratory, a unified set of software tools has been developed to facilitate PGM programming. A Macintosh-based Graphic Entry Workstation (GEWS) can be used to iconically capture processing graphs which are then automatically translated into Signal Processing Graph Notation (SPGN). The Processing Graph Support Environment (PGSE) is a set of Ada software utilities for compiling, linking, and executing the processing graphs; it includes a large, user-extensible library of signal processing primitives. PGSE is now available for VAX systems running VMS and for Sun-4 workstations under SUN OS 4.1.3. A simple signal processing application is developed to demonstrate the utility of PGSE, and current enhancements to the system are discussed. GEWS and PGSE are both available from the Naval Research Laboratory.

1. Introduction

Many commercial products now support iconic data-flow methods for programming signal processing applications [1]. Over the past decade, the Navy has also developed a data-flow method for programming multiprocessors. This approach is called the *Processing Graph Method* or *PGM* [2, 3, 4, 5, 6], and is designed to reduce the cost and complexity of signal processing application development. In PGM, signal processing applications are developed as directed data-flow graphs (or *processing graphs)* and *command programs*. The numerical computations are realized by elementary signal processing functions underlying the processing graph nodes, while the command programs provide a mechanism for dynamically managing graph execution, including graph initiation, termination, and reconfiguration. Graphs are deterministic while command programs mediate inherently non-deterministic responses to operator input, including the addition or deletion of sensor channels.

The PGM specification [2] defines the grammar and syntax for *Signal Processing Graph Notation (SPGN)*, an intermediate language for specifying processing graphs in a text format. PGM also defines the necessary functionality for command program procedures which are implemented as extensions to a host-specific high-order language. Other parallel languages can also be applied to signal processing [e.g. 7, 8, 9], but PGM retains a number of unique advantages:

(1) PGM includes a mature and stable intermediate language for data-flow graphs.
(2) PGM functionally specifies control procedures which can be embedded in a high-order language appropriate for the host processor.
(3) Parallelism is implicit in the PGM description of a data-flow graph. The programmer does not have to explicitly indicate parallelism through the use of semaphore constructs, etc.
(4) PGM is architecture independent.
(5) PGM is the only data flow method to be fully implemented on a tactical military signal processor, the AN/UYS-2A [10].

The Processing Graph Support Environment (PGSE) is a complete implementation of PGM [11]. The run-time shell includes a large user-extensible library of over 125 signal processing primitives. PGSE is designed is to facilitate the multi-user development and testing of processing graphs. It requires as input a set of SPGN files, data files, and either a command program or a sequence of interactive control instructions. With the exception of an optional shell for executing AN/UYS-2A command programs, PGSE does not incorporate any machine-specific architecture models.

2. The Processing Graph Method

A complete description of the Processing Graph Method is beyond the scope of this article. Additional information can be found in the PGM Specification [2] and Tutorial [3]. The Pictorial Processing Graph Standard [5] describes the iconic conventions for PGM graphs.

2.1 Processing Graph Entities. A *processing graph* is a directed graph in which the vertices correspond to either *nodes* or *subgraphs*, and the directed edges correspond to *queues*. *Subgraphs* are themselves processing graphs.

Queues, which convey data between the nodes, are strongly-typed first-in first-out (FIFO) data structures. The head of the queue is designated by an arrow, which also specifies the direction of data flow through the queue. Data are placed on the tail of the queue, and removed from its head. A queue can be connected to only one node at its head and only one node at its tail. A *dynamic queue* is a queue created and controlled by a command program. A *graph variable (GV)* contains a single data element, although this element may itself be a multidimensional array. A GV can be simultaneously attached to one or more nodes, and is used to broadcast data throughout the graph. Its value can be modified at runtime by the output from a node or a command program.

Nodes are the scheduled entities within a processing graph. Each node must have at least one input port, although it is not required to have an output port. Queues and graph variables are attached to the node's ports, and must be of the same data mode (type) as the port. Each node has an underlying elementary computational process called a *primitive*. Examples of signal processing primitives are Fast Fourier Transforms (FFTs), filter and bandshift functions, and elementary vector and matrix operators. Data will be passed to and from the primitive during node execution.

Associated with each node port is set of *Node Execution Parameters* (NEPs). These parameters determine how the data on the queues are to be used. There are five NEPs for input queues: *threshold amount*, *read amount*, *consume amount*, *offset amount*, and *valves*. When the number of data elements placed on each of a node's input queues is greater than or equal to the *threshold amount* for each queue, the node is ready to execute. The *read amount* specifies the number of elements that are read by the node before the underlying primitive executes; *offset amount* is the number of elements which are skipped before reading the data. The *consume amount* specifies the number of data elements which are removed from an input after the node executes. A *valve* has some integer value; if this value is zero, no data is placed on the queue.

2.2 Command programs. Unlike processing graphs, command programs are implementation-dependent on the PGM host. Chapter 5 of the PGM specification specifies the Command Program (CP) functionality required for a PGM implementation [2]. A syntax is suggested, but it is not mandatory. CP procedures alone are not sufficient to write command programs, for these procedures must be incorporated into a high-order language (HOL) specific to the target machine. Command program functions permit the user to start and stop the graph, to enter operator specified parameters, to link and unlink queues and graph variables, and to flush the data from a dynamic queue prior to restarting the graph. The following list summarizes the major command program functions:

- Declare the initial CP or a spawned CP and its formal parameters.
- Spawn one or more instances of command programs from the initial CP or one of its children.
- Start or stop a graph.
- Completely reinitialize a graph instance, and resume its execution.
- Create and destroy dynamic queues and graph variables.
- Initialize a dynamic queue and add data to it, read data from it, or flush the data entirely.
- Read and write to dynamic queues and graph variables.
- Connect or disconnect a CP to the head or tail of a dynamic queue, or link or unlink a dynamic queue from the port of a graph or I/O procedure.
- Wait for the completion of a procedure to read a queue.
- Initialize I/O procedures, associate them and with dynamic queues, and start and stop the I/O processes.

3. The Processing Graph Support Environment (PGSE)

The Processing Graph Support Environment is set of Ada programs for compiling, linking, and executing processing graphs and command programs. PGSE was written and developed with the Digital Equipment Corporation Ada Compilation System (DEC ACS), and has been ported to the SUN-4 under the Telesoft TeleGen2 compiler. PGSE incorporates a compiler, linker, and run-time shell (Fig. 1).

The PGSE compiler performs extensive syntactical error checking on the source files, as well as verifying the conformance of the primitive arguments with a library of primitive profiles. The linker accepts the object modules produced by the compiler, and produces a single graph realization load module. Extensive checks are performed to verify that the interfaces between graphs and subgraphs are consistently defined.

3.1 The PGSE Run-Time Shell. The run-time shell permits the user to debug and execute the graph realization module created during the link step. The executing graph can read data from ASCII-formatted data files, process the data, and produce formatted output files. The PGSE run-time shell includes an extensible library of signal processing primitives compliant with the AN/UYS-2 primitive specification. PGSE is distributed with the source code for all supported primitives, and a set of Ada packages to assist the user in developing new primitives [11, part 2, chapter 7].

During graph execution, the shell evaluates run-time expressions and continually checks the interrelationship of the NEPs. The constraints include the requirements that all NEPs except valves are greater than or equal to zero; and that threshold amounts are greater than or equal to both the consume amount, and the sum of the read and offset amounts. Primitive specific constraints are likewise checked for data compliance. Warnings will also be issued if any graph system or primitive-specific constraint is violated.

3.2 Shell Commands. Commands to the shell can be entered either interactively, or by writing a formal Ada command program. The original Ada command program environment was developed in 1988; more recently, a shell has been implemented which will process Ada programs written in accordance with the AN/UYS-2A command program environment. Many of the interactive commands emulate command program functions. Commands are defined to create and destroy dynamic queues or graph variables; to link and unlink the queues or graph variables from a graph; to deposit, examine, and remove data from either dynamic and local queues or graph variables; and to start and stop I/O processes. The graph itself can be started and halted, and the scheduling of nodes suspended and resumed.

An executing graph can be halted by suspending node scheduling, by stopping graph I/O, or by setting a breakpoint for a node or primitive. The use of *breakpoints* provides a major tool for symbolic debugging of the graph. Breakpoints can be set to execute after some specified number of node or primitive executions, and prior to the next execution of the entity. A breakpoint will be either temporary or permanent; if permanent, it must be explicitly canceled or the scheduler will stop on every subsequent occurrence of the specified condition. A graph can also be executed one node at a time.

The *trace function* is identical to the breakpoint function; however, it only notifies the user when a given condition has been met, rather than halting graph execution. *Watchpoints* can be set to suspend graph execution whenever a particular NEP is modified outside of a specified data range. The watchpoint function is useful for analyzing graphs with variable NEPs.

When the graph execution is suspended, different entities within the graph can be examined. The user can display the status of the node, and the status of each of its input and output queues. All breakpoints can be listed. When a queue is examined, the data it contains can be read to a file, listed on the screen, or plotted to the screen. All formal parameters associated with nodes, queues, and graph variables can be displayed.

Log and history files capture the history of the graph execution. The log file records the sequence of interactive commands processed by the shell, and the history file saves the sequence of node executions, including the path to the node and the underlying primitive.

4. A Signal Processing Example

Fig. 2 illustrates an elementary four-node processing graph which performs filtering, spectral analysis, integration, and complex magnitude estimation. This representation was generated with the GEWS workstation. The nodes are shadowed, and the queues are in boldface: this convention, in compliance with the processing graph pictorial standard [5], indicates that each of these queue and node entities represents a *family* of graph instances. That is, there will be NC copies of the graph executing in parallel, each on a separate sensor channel.

In the case of the queues, the family structure is made explicit by the bracketed indices which precede the queue names ([1..NC]Q_IN, [1..NC]Q1, etc.). The ith node or queue in a family will be designated as [I]ENTITY_NAME. The syntax for the SPGN specifications is also straightforward. A family of NC queues of mode float can be specified as

```
%QUEUE ( [1..NC]Q_IN_FFT : FLOAT )
```

A node which calls the primitive *real-to-complex FFT* can be declared as

```
%NODE ( [I=1..NC]FFT
```

```
PRIMITIVE = FFT_RC
PRIM_IN  = N_IN,
     N_OUT,
     FLAG,
     BIN_NUM,
     [I]Q_IN_FFT
          THRESHOLD = N
          CONSUME = N*(1-OL/4)
PRIM_OUT = [I]Q_OUT_FFT
```

where N_IN, N_OUT, FLAG, and BIN_NUMBER are the number of points per transform, the number of output points per transform, the direction flag, and the starting bin number, respectively. [I]Q_IN_FFT and [I]Q_OUT_FFT are the ith members of a family of input queues and output queues, respectively. The SPGN for the example was generated on the Macintosh Graphic Entry Workstation (GEWS) [12], but it can also be written by hand.

The application can be started and managed either by a sequence of interactive commands, or by an Ada command program. In either case, the command sequence must be:

- define the formal graph variables and/or queues connected to the graph.
- define the I/O processes for the formal queues and associate the I/O processes with the input and output data files.
- start all I/O processes except for one input process.
- start the graph via the START command.
- start the remaining input process.

Figure 3 shows the transformed data on the input, output, and internal queues of the graph. The data was captured by setting breakpoints on the appropriate nodes and plotting the queue data to the screen. The input to the graph was simulated time-series data generated by summing a degraded sinusoidal signal and three of its harmonics. A practical difficulty is that PGSE cannot examine data on the formal queues. The original graph was therefore augmented by replacing the input and output ports by a family of concatenated nodes and replicated nodes, respectively.

5. The Status and Future of PGSE

GEWS, available from the Naval Research Laboratory [13], has been widely distributed, although it is no longer being maintained against the current PGM specification. The Navy-sanctioned workstation tools for AN/UYS-2 application development are the Sun-based gred/grail utilities, developed by AT&T Bell Laboratories [14]. Gred is the iconic capture system, and grail translates the gred output files into SPGN.

PGSE, like GEWS, has been widely distributed, both to industrial users and to Navy laboratories [13]. Aside from the AN/UYS-2, the VAX- and SUN-based PGSE system currently provides the only available means for developing PGM applications and primitives. New primitives can be inexpensively prototyped and tested in Ada. PGSE is of particular value to teams of software developers with limited access to the AN/UYS-2, which is effectively a single user machine, although the speed of PGSE graph execution is generally insufficient for real-time applications. PGSE is now being used by Hughes Aircraft to prototype applications for the Active Low-Frequency Sonar (ALFS). It has been employed at NRL and NAWC to prototype passive sonar applications.

NRL is currently testing a new release of PGSE which incorporates an AN/UYS-2A Ada command program shell [15]. A new interface will permit users to execute subgraphs as single nodes; this permits the user to simulate compound primitives ("chains") constructed from subgraphs with nodes using existing primitives. New tools to simplify the coding of Ada primitives have been proposed.

The development of PGSE has been driven by the Navy's need for rapid prototyping of AN/UYS-2 applications. Because the PGSE implementation is independent of any specific target machine, the system has wide utility. The domain of applications is a function only of the primitive libraries. Although the current primitive library is based upon the AN/UYS-2 primitive specification [16], primitives for C3I applications and Bayesian inference networks have also been written.

REFERENCES

[1] R. Robison, "Tools for embedding DSP," *IEEE Spectrum*, vol. 29, no. 11, 1992, pp. 81-84.

[2] *Processing Graph Method Specification: Version 1.0,* prepared by the Naval Research Laboratory for use by the Navy Standard Signal Processing Program Office (PMS-412), Dec. 1987.

[3] *Processing Graph Method Tutorial.*, prepared by the

Naval Research Laboratory for use by the Navy Standard Signal Processing Program Office (PMS-412), Jan. 1990.

[4] R. Stevens, "A Tutorial on the Processing Graph Method, "*Proceedings of the 1987 Summer Computer Simulation Conference (Montreal, Quebec, Canada, July 27-30)*, The Society for Computer Simulation, San Diego, CA 92117.

[5] *Pictorial Processing Graph Standard*, prepared by the Naval Research Laboratory and Evaluation Corporation International for use by the Navy Standard Signal Processing Program Office (PMS-412), March 1988.

[6] Information concerning PGM documentation and training courses should be directed to:

Naval Sea System Command
Room 11E28,
Attention: Mr. Clair Guthrie
Arlington, VA 22202
(Telephone: 202-602-0284)

[7] *OCCAM 2 Reference Manual*, INMOS Limited, 1988.

[8] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The Synchronous Data Flow Programming Language LUSTRE," *Proceedings of the IEEE*, vol. 79, no. 9, 1991, pp. 1305-1320.

[9] P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire, "Programming Real-Time Applications with SIGNAL," *Proceedings of the IEEE*, vol. 79, no. 9, 1991, pp. 1321-1336.

[10] G. Melcher, G.Thomas, and D. Kaplan, "The Navy's New Standard Digital Signal Processor, the AN/UYS-2," *The Journal of VLSI Signal Processing*, vol. 2, no. 2, 1990, pp. 103-109.

[11] *User's Manuals for the Processing Graph Support Environment: Parts 1-4*, prepared by Hughes Aircraft Company Ground Systems Group for the Naval Research Laboratory, Dec. 1989.

[12] *Graph Entry Work Station (GEWS) User's Guide (Beta Test Version 0.98)*, prepared by Hughes Aircraft Company Ground Systems Group for the Naval Research Laboratory, Dec. 1989.

[13] The GEWS software and documentation is distributed for free by the Naval ResearchLaboratory. The PGSE tools are available for a nominal distribution fee A written request, accompanied by a blank Macintosh disk, should be submitted to:

Roger Hillson or Judy Trenck
Code 5583
Naval Research Laboratory
Washington, D.C. 20375-5000
(Telephone: 202-404-7332/7334)

[14] *ECOS Workstation User Manual.*, prepared for Naval Sea Systems Command (PMS-412) by AT&T Bell Laboratories, March 1989.

[15] *AN/UYS-2A(V) ASIP Application Programmer's Reference Manual*, prepared for Naval Sea Systems Command (PMS-412) by AT&T Bell Laboratories, March 1993.

[16] *ECOS Primitives Specification Library (Floating Point: SEM B)*, prepared for Naval Sea Systems Command (PMS-412) by AT&T Bell Laboratories, September 1988.

Acknowledgments: The major sponsor for PGSE development was the DOD Software Initiative Software Technology for Adaptable Reliable Systems (STARS), with additional funding from the Naval Sea Systems Command (PMS-412 / PMO-4128) and the Naval Air Systems Command (PMA-264). Richard Stevens, David Kaplan, and Judy Trenck have facilitated many aspects of the work described here. Particular thanks is due to Bradley Logan of the Hughes Aircraft Company Ground Systems Group, technical director for PGSE development.

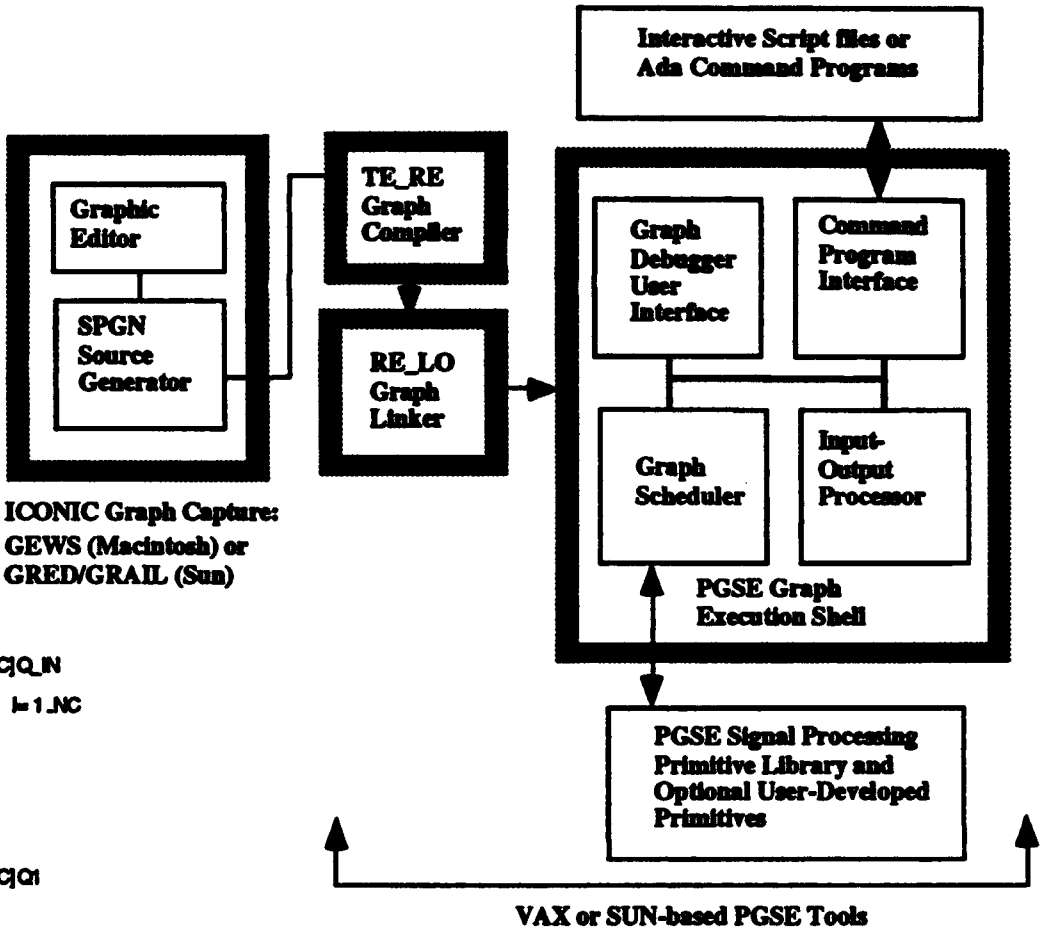


Fig. 1. **The Processing Graph Support Environment**

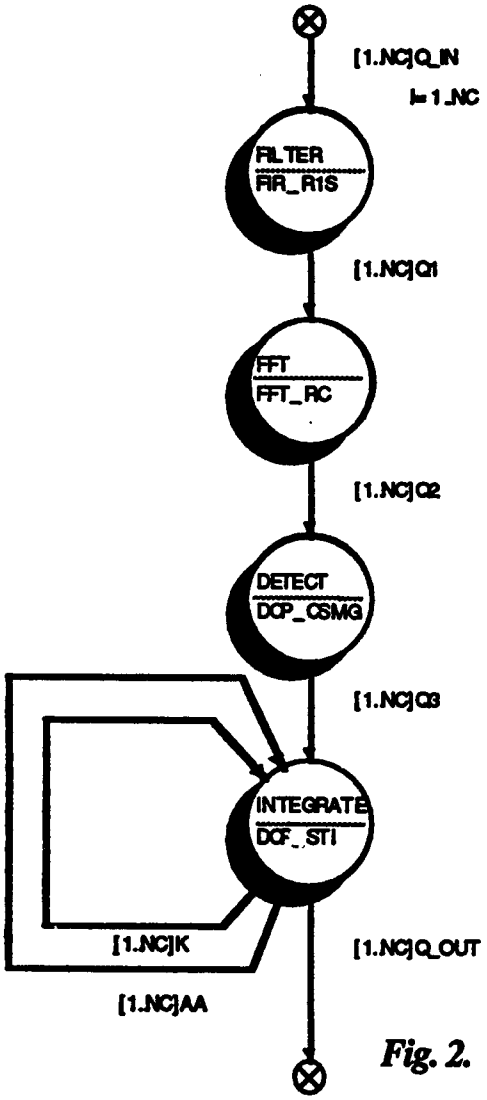


Fig. 2. **A Four-Node Graph**

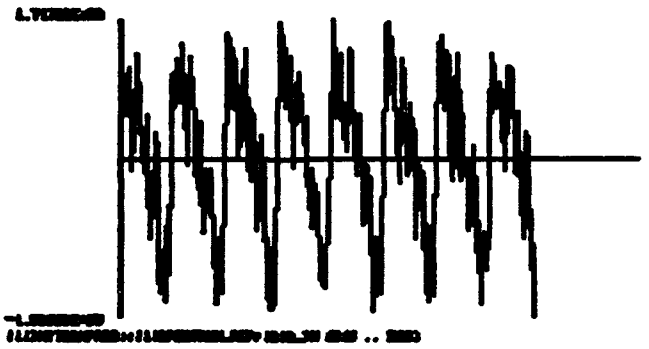


Fig. 3a. **Raw Time Series Data ([2]Q_IN)**

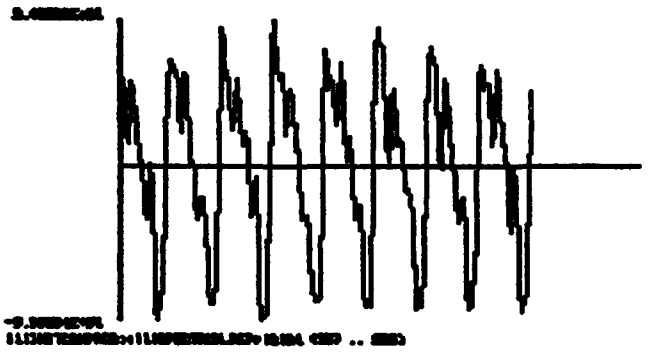


Fig. 3b. **Smooth Time-Series Data ([2]Q1)**

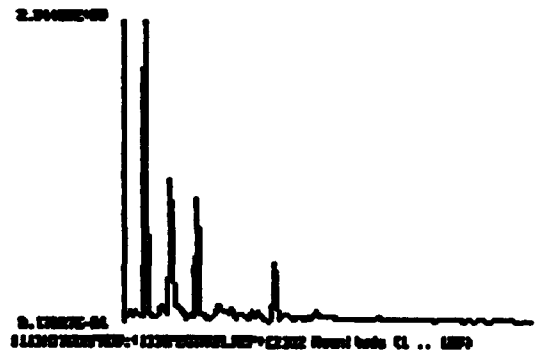


Fig. 3c. **Complex Spectral Data ([2]Q2)**

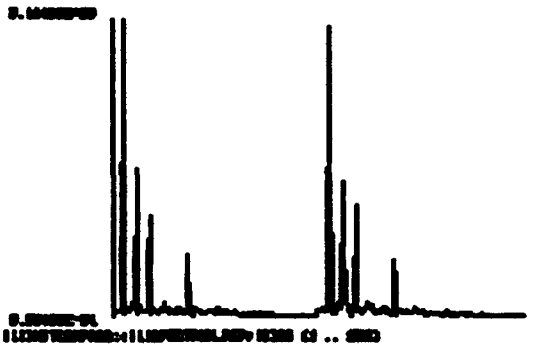


Fig. 3d. **Real Spectral Magnitude ([2]Q3)**

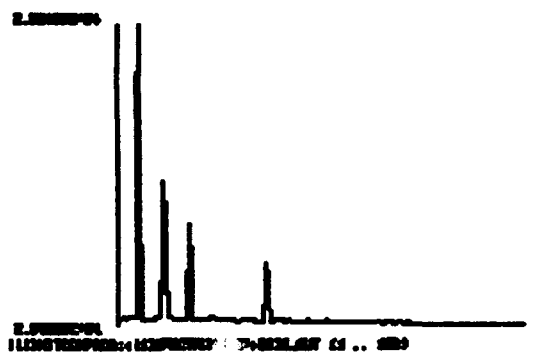


Fig. 3e. **Integrated Spectral Magnitude ([2]Q_OUT) (8 Spectral Scans/Integration)**

A Real-Time Object Model: A Step toward an Integrated Methodology for Engineering Complex Dependable Systems

K. H. (Kane) Kim and L. F. Bacellar
Dept. of Electrical & Computer Engineering.
Univ. of California
Irvine, CA 92717, U.S.A.

Abstract

The need to establish a coherently integrated methodology for engineering complex real-time computer systems has been taken seriously in recent years by both researchers and practitioners dealing with safety-critical computer-based applications. In our view, a missing cornerstone for developing such an engineering methodology is a system (and component) model which is effective not only for abstraction and stepwise refinement of complex real-time computer systems but also for representing and providing a basis for analysis of the application environments. Only with the establishment of such a modeling scheme can one hope to realize a method for requirements analysis and specification which will produce a solid information base for both systematic design and rigorous evaluation. The first author and his research collaborator, Hermann Kopetz, jointly formulated an initial framework of such a desirable system model a few years ago. It was named the *real-time object* (RT-object) model. The RT-object model, an extension of the conventional object model, possesses capabilities for precise handling of the timing behavior of modeled subjects. A basic thesis we have attempted to validate is that any application environment, not only control computer systems, can be modeled as a set of interacting RT-objects. The potential of the RT-object model as a backbone of an integrated methodology for engineering complex dependable systems is discussed.

1. Introduction

Techniques for engineering complex dependable real-time systems (DRS's) have been a subject of intensive research for more than two decades but by and large they have evolved in insufficiently integrated forms up to now. As a consequence, we perceive that the following conditions in the current practice in system engineering exist :

(1) Lack of rigor in requirements specification:
Particularly problematic is the specification of temporal behavior requirements and dependability requirements. As a result, the requirements-design traceability has been generally weak.

(2) Weak traceability among various system models:
The traceability among various system models used during high-level design, validation, and evaluation has been generally weak. The consequence has been the poor interoperability and coherence among various tools mobilized during system development and evaluation.

(3) Lack of integration in design techniques:
Most real-time fault tolerance techniques stand unnecessarily in isolation while in fact many of them are complementary in nature. Cost-effective integration of such techniques remains unachieved. Also, approaches have not been sufficiently developed for optimizing allocation of resources which would realize efficient implementation of real-time distributed computer systems (DCS's). Naturally, integration of effective resource allocation approaches with fault tolerance techniques has been delayed.

In our view, the key issue to be resolved is the *uniformity* and *achievable accuracy* in representation of both application environments and designs at different levels evolving during the system development cycle. That is, a desired representation (or modeling) scheme should be effective not only in the abstraction of real-time (computer) control systems under design but also in the representation of the application environments. Such a modeling scheme should allow variable-accuracy representations ranging from a full-detail functional specification to a high-level structural representation. In particular, it should be capable of handling temporal characteristics at various degrees of accuracy. Only with the establishment of such a modeling scheme can one hope to realize a method for requirements analysis and specification which will produce a solid information base for both systematic design and rigorous evaluation.

The first author and his research collaborator, Hermann Kopetz, jointly formulated

an initial framework of such a desirable modeling scheme in [Kop90a, Kop90b]. It was named the real-time object (RT-object) model. The RT-object model, an extension of the conventional object model, possesses capabilities for precise handling of the timing behavior of modeled subjects. Although the term "real-time object" has appeared in literature quite a few times and there is some common ground with what was referred to as a real-time object in previous literature [Bih89, Rum91], the real-time object model discussed in this paper possesses some concrete unique characteristics. In order to distinguish the real-time object model discussed here and in [Kop90a, Kop90b] from others, it is often denoted by RTO.k.

Besides the timeliness of the output, an attribute that is of particular importance in complex real-time systems is dependability which encompasses reliability, availability, and security [Lap92]. The current practice in handling dependability during the system engineering cycle lacks a coherent systematic process. The need for establishing an integrated methodology is especially acute in this area. Again, we argue in this paper that the RTO.k model is a promising framework for building up such a methodology.

Therefore, the main theses put forth in this paper are :
(1) The RTO.k object model has strong uniformity and unbound achievable accuracy in representing both application environments and computer control systems under design, and
(2) The RTO.k object model is a promising backbone of a coherently integrated system development methodology, in particular, a framework in which various complementary dependability assurance techniques can be integrated.

2. The RT-object model

2.1 Motivation

In our previous study dealing with the subject of DRS's, the need for a model such as the RTO.k object model was encountered in several different contexts. First, during the attempts to validate rigorously the effectiveness of certain real-time fault tolerance mechanisms in the context of real-time DCS's, questions such as the following were encountered:

How are time-out values to be chosen ?
How are the recovery time requirements to be determined ?

Secondly, in trying to extend the approaches formulated for scheduling hard-real-time tasks in single-node (including multiprocessor based) computer systems [Kim80] into the approaches applicable to DCS's, the following questions were encountered:

How is the urgency of a task to be determined ?
What is the relationship between stimulus-to-response deadlines and urgencies of tasks ?
How are the concurrency and contention for processing, storage, and communication resources to be reflected in determining an optimal allocation ?
How are dynamic system reconfiguration actions to be accommodated without endangering the output timeliness ?

It appeared that the approaches formulated for single-node systems were too simplistic to be of use as a baseline for deriving approaches applicable to sizable real-time DCS's.

Thirdly, in searching for approaches for the rigorous specification of timing requirements, the following questions were encountered:

Would it be sufficient to associate timing specifications with procedure-segments and messages crossing node boundaries ?
How are fault tolerance actions to be accommodated in timing specifications ?

Our conclusion was that all the above questions pointed to one basic law: rigorous handling of the temporal behavior of a DCS requires a *global view* of the system and a *top-down engineering*. Consequently, a model of a system or its components that facilitates a global view and stepwise refinement of the view was recognized as the key item needed in resolving the questions raised above. So, a search began and at the outset the object model appeared to be a natural building-block for any modular-structure system [Dah72, Boo91, Rum91]. However, conventional object models did not appear to have sufficient concrete mechanisms that provide power for accurate modeling of application environments.

2.2 The essence of the RT-object model, RTO.k

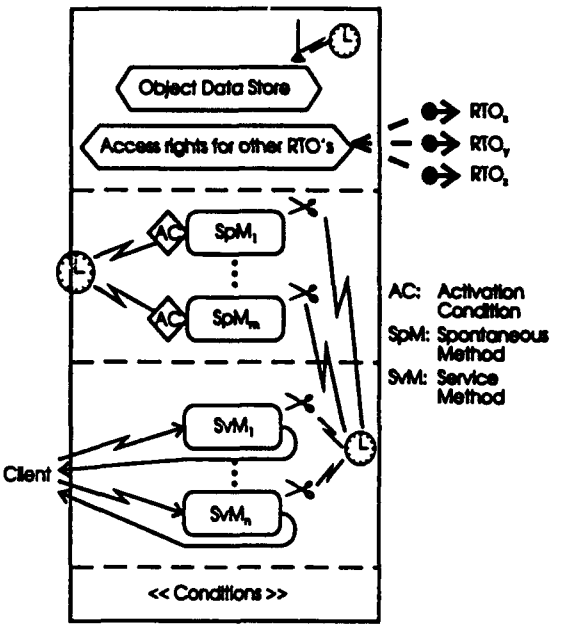
The RT-object model formulated by Kopetz and Kim in [Kop90a, Kop90b], often denoted by RTO.k, is an extension of the conventional object model(s). As an object model it is :
(1) independent of the language (textual or graphic) used to program or specify object designs,
(2) independent of the way inheritance is facilitated, and
(3) independent of the service-call mechanisms or the message protocols by which objects

exchange information on services requested and completed.

It is a stepwise refinable structure with abstract data type characteristics. It is an extension of the conventional object model(s) in at least three essential ways:
(1) For each execution of a method of an RTO.k object, a *deadline* is imposed;
(2) For some methods of an RTO.k object a real-time clock serves as the mechanism for triggering the method executions as a function of real time and such methods are called *time-triggered (TT-) methods*; and
(3) Real-time data contained in an RTO.k object become invalid after the interval called the *maximum validity duration* passes.

In addition, the following constraint on method executions is incorporated into the RTO.k model.
(4) A basic concurrency constraint which prevents conflicts between TT-methods and message-triggered methods is incorporated. In general, activations of object methods triggered by messages from external clients are allowed only when TT-method executions are not in place. To be exact, when a message-triggered method is not free of data conflict with a TT-method, execution of the former (message-triggered) method must not be allowed in a time zone separated by less than a certain system-dependent constant, say g, from a TT-execution of the latter method. This restriction is called the basic concurrency constraint or state accessibility constraint. Note that this basic constraint does not impose any restriction on concurrent execution of TT-methods or concurrent execution of message-triggered methods.

Figure 1 depicts the essential components of the RTO.k object model. The extension (1), namely the deadline imposed on a method execution, has been mentioned in every discussion of the term real-time object. In addition, the notion of calling for an object method as real-time reaches some predetermined time-points was mentioned in some of the literature in which the term real-time object was used, although the approach of clearly distinguishing between TT-methods and message-triggered methods adopted in the extension (2) is probably a unique feature of the RTO.k model. The extension (3) (use of the maximum validity duration to eliminate old useless data) and the extension (4) (basic concurrency constraint) are unique features of the RTO.k model. Therefore, on one hand, the RTO.k object model is general in that it is independent of the specification / design language, the inheritance mechanism, and the inter-object message protocols used. On the other hand, it has a new concrete structure in that



Essential components added to a basic timeless object

1 - Deadline for execution of each object method
2 - Clock-driven activation of object methods
3 - Maximum validity duration for the data set in object store

Figure 1. RTO.k - A real-time object model

it contains deadlines, TT-methods, maximum validity duration, and the basic concurrency constraint. The unique characteristics of the RTO.k object model together with its relatively recent formulation are also indications that the model as it currently stands is more of a framework of an evolving model, not a fully developed mature tool.

The proposition that an object model can be used to represent not only computer systems but also application environments is not new [Dah72]. This is originated from the belief that each real-time application environment can be viewed as a set of state variables that interact among themselves. However, unlike other object models the TT-method facility in the RTO.k object model enables representation of the application environments "to any degree of accuracy desired" within the limit of the user's knowledge about the environment. That is, the clock-driven activation of object methods is a natural mechanism for representing the concurrent and continuous changing of the state variables which are typical situations in the application environments. This will be illustrated in the next section. Additional mechanisms for specifying certain parallelism existing among the methods of RTO.k objects, e.g., mechanisms of COBEGIN or FORK-JOIN nature, are also

allowed in the RTO.k object model as in many conventional object models.

The basic concurrency constraint is not only necessary for keeping the semantics of the RTO.k model in an unambiguous form but also can be useful in ensuring consistency among RT-objects. The latter point is particularly acute when RT-objects are replicated while updating the object data store in each object is the exclusive responsibility of TT-methods. It is easy to design such object replicas with the assurance that they will never provide different values with the same time-stamp to their clients [Kop90b].

A fundamental notion established in association with the RTO.k object model is that of *temporal accuracy* which refers to the time gap between a state variable in the application environment and its representation in a computer-internal RT-object [Kop90a, Kop90b]. This temporal accuracy is a notion fundamentally related to the output timeliness. Therefore, we believe that much of the temporal behavior requirements imposed on control computer systems can be specified in natural and rigorous forms in terms of temporal accuracy bounds. Temporal accuracy bounds are then major drivers for determining many other temporal entities such as deadlines for object method executions.

3. An RT-object based approach to environment modeling and requirements specification

The object model was initially formulated and used in simulation applications [Dah72]. Therefore, use of the object model in modeling the environment objects, i.e., modular entities in the environment which have time-varying internal states, has been practiced for a long time. However, as mentioned in the preceding section, the recently formulated version of the object model, the RTO.k object model, supports more accurate detailed modeling of environment objects. This aspect and how the RTO.k object model can be used during the requirements specification and high-level design steps are discussed in this section with examples derived from the defense C^3 area.

Consider an anti-missile defense scenario depicted in Figure 2. The environment in this context means a sky space segment of interest, called the "theater", and any moving objects in that theater including a valuable target to be defended (e.g., a ship) and flying objects (e.g., hostile reentry vehicles (RV's) and non-threatening slow-moving objects).

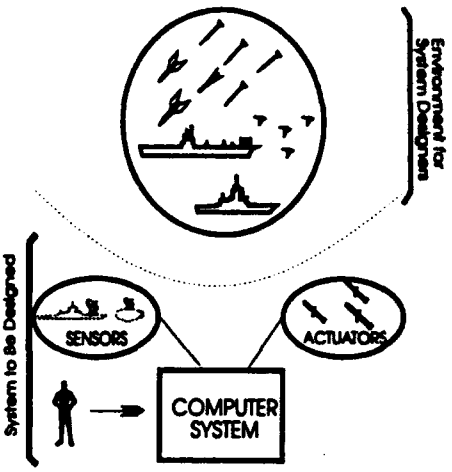


Figure 2. An anti-missile defense system

Initially the top-level requirements given by the customer who places an order for the defense system are as follows.

(1) Each RV should be intercepted if it is dangerous.
(2) If there are more dangerous RV's than the interceptors, then the early arriving RV's should be intercepted and the defense target should be moved toward a safer location.

The system designer will first decide on the set of sensors and the set of actuators (e.g., interceptors) to be deployed. Thereafter, the functions of the computer based control system will be determined based on the control theory logic adopted.

If some sensors and actuators chosen are located in the theater, then a representation of the application environment must be expanded to include these newly chosen environment objects. An RTO.k representation of the initial theater (before deciding on the sensors and actuators to be incorporated within the theater) is depicted in Figure 3.

The internal storage of this high-level RT-object basically consists of the space in the theater, a defense target (ship), and a dynamically varying number of RV's. Therefore, the information kept in this RT-object is a composition of the information kept in the defense target, the RV's, and any other object in the theater space. A noteworthy property here is that each of these components that are treated as components of the internal storage, i.e., the defense target and the RV's themselves, can in turn be represented as an RT-object.

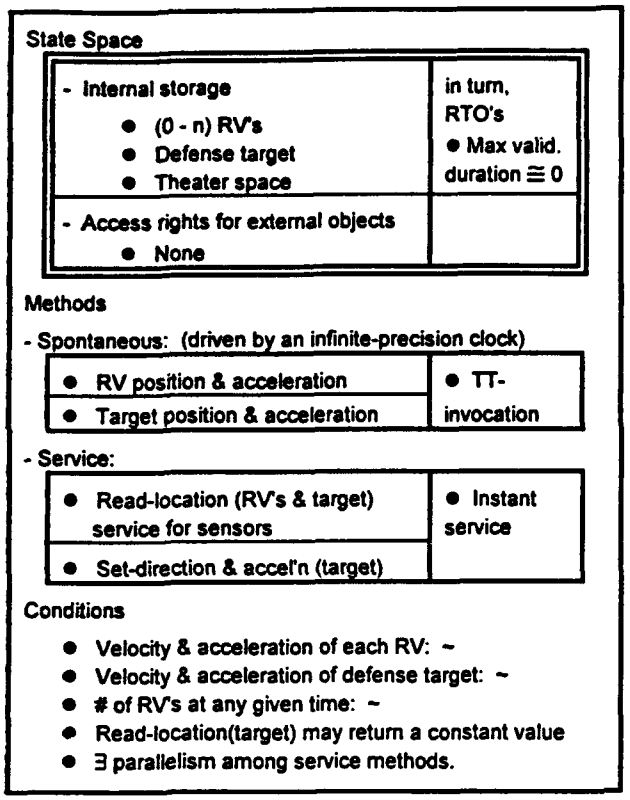


Figure 3. An RTO.k specification of the environment

The methods in this top-level RT-object model for the theater are classified into two types. One class of methods are the TT-methods and they are labeled *spontaneous methods* in Figure 3. Conceptually these methods are activated continuously and each of their executions is completed instantly. If we adopt the less precise version of the model in which the time domain is a discrete domain and the time gap between two instants adjacent in the domain is called a clock tick, then a less accurate representation of the environment results. That is, such view dictates the activation frequency of any spontaneous method to be no more than once per every clock tick while allowing each execution to be completed before or by the time of the following activation of the same method. Therefore, the spontaneous methods are the mechanisms for representing (or simulating) continuous state changes that occur naturally in the environment objects. The natural parallelism that exists among the environment objects is precisely represented by use of multiple TT-methods which may be activated simultaneously. In general, the accuracy of an RT-object representation of the environment is a direct function of the activation frequencies of TT-methods.

The other class of methods are the message-triggered methods that are found in conventional objects. They are labeled *service methods* in Figure 3. They are invoked upon requests from the clients. However, the clients here are unusual ones, i.e., sensors and actuators interfacing with environment objects. Figure 3 corresponds to the case where some sensors to be located outside the theater have been chosen but no sensors and actuators to be located inside the theater have yet been chosen. Sensors work to obtain information about the states of environment objects, primarily locations, movement directions, and some signatures of RV's and the defense target. This relationship between sensors and environment objects can be represented partially by the service methods such as "Read-location (environment object)" in Figure 3, with the understanding that such a method is executed at the instant at which a sensor makes an observation of the environment state.

Similarly, actuators work to make impact on the conditions and future courses of environment objects. In this example, the only possible impact that can be made on RV's by the system being designed is the collision of the interceptors against the hostile RV's. These interceptors are actuators produced at an early stage of the system design and once they are produced, they should be treated as environment objects in the theater as well. Although Figure 3 represents the theater before introduction of such actuators, there are some control points in the defense target which can be accessed from the computer system, typically structured as a control computer network (CCN), via a communication channel, e.g., a radio communication device. The "Set-direction & acceleration (target)" service method in Figure 3 represents such possibility.

The last important component of an object model is the set of constraints that governs both information states of the object and the computational results produced by object methods. In Figure 3, these constraints are listed in the section labeled *Conditions*. All the constraints listed are of the "laws of physics" type. Therefore, the system designer's knowledge of physics is expressed in the *Conditions* section. One of the constraints in Figure 3 specifies the parallelism that exists among service methods, i.e., the parallelism among the sensor activities and actuator activities. This is an example of a concurrency constraint. Concurrency constraints are integral components of RT-objects.

As mentioned earlier, a single RT-object specification of the environment can be refined into a network of RT-object specifications, each corresponding to a different environment object.

All the knowledge contained in the *Conditions* section of the single RT-object specification should be retained, most likely in a scattered form, in the network of environment object specifications. Additional knowledge may also be introduced during the refinement process.

Note: The defense C^3 scenario discussed above was chosen because it had been established in a LAN based distributed computer system testbed in the authors' laboratory several years ago. It is a rich scenario for use as a test ground for various advanced approaches to dependable real-time computing. The existing implementation is structured in the conventional object oriented style and not based on the RTO.k object structuring. A new implementation based on the RTO.k object structuring is under way.

4. The RT-object model as a potential backbone of an integrated methodology for engineering complex DRS's

There are many important implications of the RT-object based unified representation of both control computer systems and the applications environments. The RT-object model is a potential basic structure that can evolve,

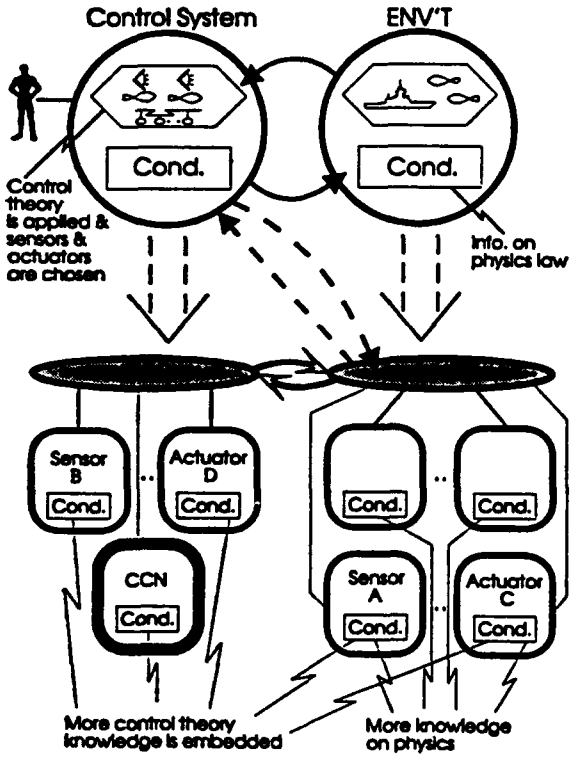


Figure 4. High-level design via elaboration of RT-objects

rather than being present temporarily, from the requirements specification step through the structured design step to the validation and evaluation step.

Figure 4 depicts the use of the RT-object model during the early steps in the system development cycle. Some time after an RT-object based specification of the environment is obtained, an RT-object based specification of sensors, actuators, and the control computer network (CCN) can be obtained on the basis of the control theory adopted. The *Conditions* section of the RT-object representing the CCN must contain information on the requirements imposed by the system customer.

For example, if a radar sensor and interceptors as actuators are adopted, then the *Conditions* section of the RT-object specifying the CCN must include the type of constraints shown in Figure 5.

All the specifications (of both the environment and the CCN) may go through further refinement which is essentially a high-level design activity. As the environment specification is refined, more knowledge of physics may be incorporated. Similarly, as the CCN specification is refined, more control theory knowledge may be incorporated.

So far, the discussion has been focused mostly on the role that the RT-object model can play in the requirements specification and high-level design steps which are the earliest steps in the system engineering cycle. Under the RT-object based engineering methodology envisioned here, detailed design means essentially a process of converting the high-level RT-object specifications into more detailed or even fully executable RT-object specifications.

A number of significant benefits are expected to accrue from such a practice. A few major ones are listed below. However, concrete demonstrations of these benefits are yet to be seen and thus the following should be treated as potential benefits.

(1) Requirements specification:

Requirements, in particular temporal behavior requirements and dependability requirements, can be specified in rigorous forms that can be detailed to varying degrees. As mentioned before, these requirements are expressed in the *Conditions* section of the RT-object specifications. Uniformity and unbound achievable accuracy in representing both the environment and the CCN are the fundamental ingredients enabling rigorous specification of requirements. Also, better information flow from the requirements specification step to the

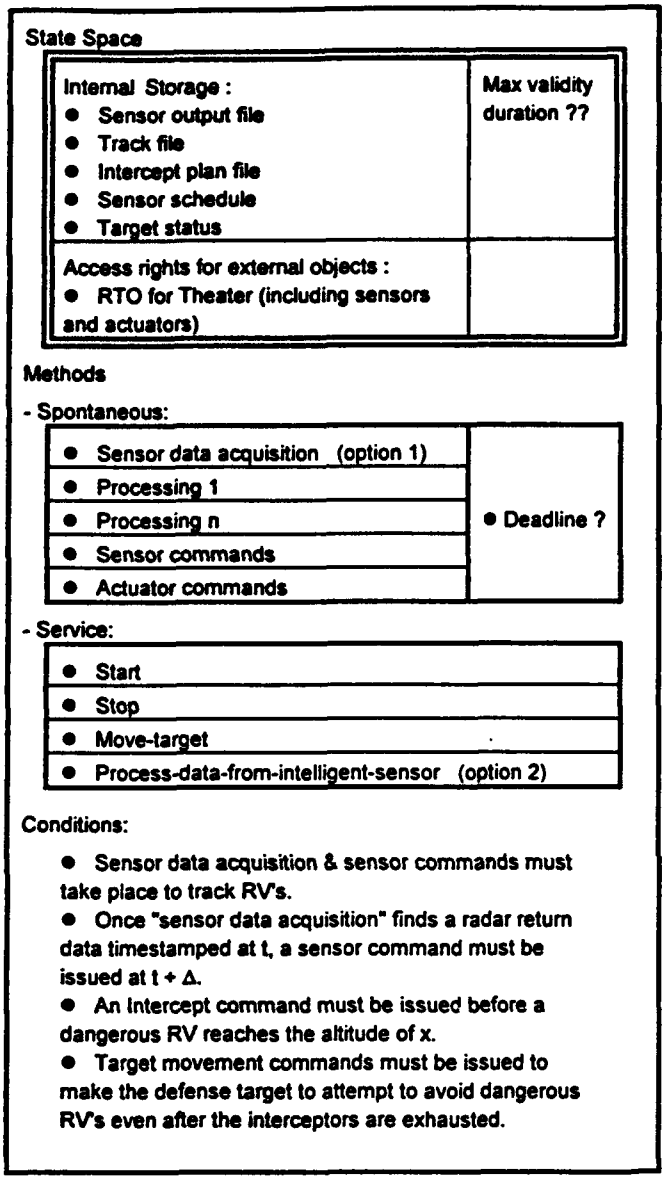


Figure 5. An RTO.k specification of the CCN

structured design and optimization step can result due to the common RT-object structure that is used in both steps.

(2) Resource allocation:

Top-down multi-level resource allocation can be facilitated in the course of stepwise creation of object hierarchy designs after starting with RT-object based requirement specifications. Such resource allocation approaches are expected to offer significant advantages over the bottom-up approaches of extrapolating conventional uniprocessor scheduling techniques to deal with DCS's. Obviously, resources can also be represented at varying degrees of accuracy as RT-objects before their allocation. Resource allocation can then become a process of converting each high-level RT-object into a network of software (or user) objects and hardware (or resource) objects.

(3) Rigorous validation:

Validation of both temporal behavior and dependability can be performed in a much more systematic and rigorous manner than before. This is again due to the improved rigor in requirements specification and the improved information flow during the system development cycle.

(4) Compatibility among modeling and evaluation tools:

Due to the improved information flow from the requirements specification step to the validation and evaluation step, the compatibility among the tools mobilized during various steps is expected to improve significantly.

It should also be noted that an RT-object based system engineering methodology can be gainfully enhanced by incorporating additional modeling and analysis techniques (e.g., process oriented models and data flow models [Rum91, You89]) as supplements to the RT-object backbone during various steps of the system engineering cycle.

5. Integration of dependability design techniques into an RT-object based system engineering methodology

Quite a few complementary techniques for designing highly fault-tolerant real-time DCS's have been developed over the years, but each has matured to a different degree in isolation without being integrated with others. Two most important basic classes of techniques are as follows.

(1) Techniques for realizing action-level fault tolerance :

These techniques aim for designing DCS's such that critical actions take place (i.e., each critical real-time task produces an output as specified) in spite of component failures. Therefore, they aim for much higher degree of dependability than those techniques aimed for merely aborting some tasks and cleansing system states upon component failures. For tolerating hardware faults only, the most basic techniques formulated are :

Voting TMR (triple modular redundancy) (more generally, voting NMR) [Toy87],
PSP (pair of self-checking processors) (of which a special case is the pair-of-comparing-pairs scheme used in the

Stratus system [Wil85]) [Kim92],
Temporary blackout handling [Kim92].

For tolerating both hardware and software faults, the most basic techniques are :

DRB (distributed recovery block) (which uses the recovery block scheme as a component) [Hec91, Kim89a, Kim92],
NVP (N-version programming) [Avi85].

More "expensive" techniques devised to supplement the aforementioned basic techniques include the distributed conversation scheme and others [Kim89b]. ■

(2) Techniques for DCS diagnosis and reconfiguration

These techniques aim for minimizing the periods during which sick organs are lurking in DCS's. This means to facilitate fast learning by each fault-free node of faults occurring in other parts of the DCS and fast reconfiguration including functional amputation of faulty components, reincorporation of repaired or new components, and redistribution of tasks. Basically the following three types of approaches are conceivable:

Centralized,
Decentralized,
Hybrid.

Centralized approaches are simple and have been considered from the beginning days of distributed computing. Yet its integration with the techniques for action-level fault tolerance has not been fully accomplished. Decentralized approaches are much less mature as a technology although again the basic concept is at least 20 years old. Hybrid approaches can be developed in rigorous forms only after decentralized approaches are well understood. ■

In integrating the aforementioned techniques and others for designing fault-tolerant dependable computing capabilities, the following major steps need to be accomplished. Our basic thesis here again is that the RT-object model provides a natural framework for specifying dependability requirements at varying levels of detail and also for integrating various fault tolerance mechanisms / techniques that have evolved in isolation.

(1) Rigorous specification of dependability requirements

This is not a sufficiently mature technology area in spite of the fact that the field of fault-tolerant computing is at least 30 years old. It has been mentioned for the following reasons that the RT-object structure offers a good framework in which specifications of dependability requirements can be incorporated. Initially, a dependability requirements specification will appear in the *Conditions* sections of high-level RT-objects. The specification can then be decomposed or refined as high-level RT-objects are converted into networks of smaller RT-objects. Uniformity of the representation structures maintained across the environment specification and the CCN specification and also maintained during stepwise refinement, is a major ingredient enabling the rigorous and coherent specification of dependability requirements and the design of dependable computing capabilities. Rigorous specification of dependability requirements in turn facilitates cost-effective integration of various dependability design techniques evolved in isolation.

(2) Integration of action-level fault tolerance techniques and DCS diagnosis and reconfiguration techniques

Since these techniques require certain allocation of hardware resources, a model that can represent both functions to be performed and the supporting execution resources can serve as a highly valuable guiding structure. The RT-object model is a highly desirable model in this regard although the use of the RT-object model for this purpose has not been practiced widely. Active research is under way in this integration area.

(3) Integration of fault tolerance design techniques with performance-guarantee oriented design techniques

Performance-guarantee oriented design techniques include both system structuring techniques and resource allocation techniques aimed for guaranteed response time. As mentioned in the introduction, such rigorous handling of the temporal behavior requires a global view of the system and a top-down engineering. The RT-object based system engineering methodology meets this requirement effectively. Therefore, for this integration step again the RT-object model can play an important role. In addition, similar arguments can be made for integrating security enforcement techniques in a coherent manner into the RT-object based system engineering methodology.

6. Conclusion

This paper has presented a proposition that the RT-object model is not just an attractive approach for complex system modeling and structuring and it is actually a potential structuring backbone of a coherently integrated methodology for engineering DRS's. As the first step on our part toward formulating concrete

examples illustrating the feasibility of such a methodology, a specific version of the RT-object model, denoted by RTO.k, was formulated several years ago, and an experimental specification and design of a simplified C^3 application system which is based on the RTO.k object model, is under way. There are indications that many other research organizations share our feeling about the appealing nature of object based approaches to design of complex real-time systems. This raises hope that an increasing number of demonstrations of the potential benefits of using RT-object models in various parts of system engineering will be seen in open forum in the future. In spite of the promising nature of the RT-object based approaches to system engineering, however, concrete demonstrations in this area are not expected to be simple research endeavors in terms of efforts and costs required.

Acknowledgment: The first author wishes to acknowledge the useful insights obtained in this area from the cooperative work with Hermann Kopetz in Austria over the past decade. The work reported here was supported in part by US Navy, NSWC White Oak Laboratory under Contract No. N60921-92-C-0204 and in part by the University of California MICRO Program under Grant No. 92-075.

References

[Avi85] Avizienis, A, "The N-Version Approach to Fault-Tolerant Software", IEEE Trans. on SE, Vol. Se-11, No. 12, Dec. 1985, pp.1491-1501.

[Bih89] Bihari, T., Gopinath, P., and Schwan, K., "Object-Oriented Design of Real-Time Software", Proc. IEEE CS 10th Real-Time Systems Symp., 1989, pp.194-201.

[Boo91] Booch, G., 'Object-Oriented Design', Benjamin Cummings, CA, 1991.

[Dah72] Dahl, O.J., "Hierarchical Program Structuring", in Dahl, Dijkstra, & Hoare eds., 'Structured Programming', Aca. Press, NY, 1972.

[Hec91] Hecht, M. et al., "A Distributed Fault Tolerant Architecture for Nuclear Reactor and Other Critical Process Control Applications.", Proc. IEEE CS 21st Int'l Symp. on Fault-Tolerant Computing, June 1991, Montreal, pp.462-469.

[Kim80] Kim, K.H., "Recent Developments in Safe Design of Hard-Real-Time Computer Systems", Proc. Int'l Computer Symp. 1980, Vol. II, Taipei, pp. 1229-1241.

[Kim89a] Kim, K.H. and Welch, H.O., "Distributed Execution of Recovery Blocks: An Approach to Uniform Treatment of Hardware and Software Faults in Real-Time Applications.", IEEE Trans. Computers, May 1989, pp.626-636.

[Kim89b] Kim, K.H., "Approaches to System-Level Fault Tolerance in Distributed Real-Time Computer Systems", Proc. 4th Int'l Conf. on Fault-Tolerant Computing Systems, Baden-Baden, Germany, Sept. 1989, pp.268-281 (Springer-Verlag) (Invited paper).

[Kim92] Kim, K.H., "Design of Real-Time Fault-Tolerant Computing Stations", presented as a lecture in the NATO Advanced Science Institute on Real-Time Computing, Sint. Maarten, Oct. 1992, to appear as a chapter in a volume by Springer-Verlag.

[Kop90a] Kopetz, H. and Kim, K.H., "Temporal Uncertainties in Interactions among Real-Time Objects", Proc. IEEE CS 9th Symp. on Reliable Distributed Systems, Huntsville, AL, Oct. 1990, pp.165-174.

[Kop90b] Kopetz, H. and Kim, K.H., "Real-Time Objects and Temporal Uncertainties in Distributed Computer Systems", Tech. Rept. UCI-ECE-90-7b, Dept. of ECE, UCI (& Tech. Rept. TUW-ITI-10a/90, Institut fur Praktische Informatik, Tech. Univ. of Vienna), Oct. 1990.

[Lap92] Laprie, J.C., "Dependability: a Unifying Concept for Reliable, Safe, Secure Computing", in J. van Leeuwen ed., Information Processing 92, Vol. 1, pp.585-593. (Invited paper)

[Ran75] Randell, B., "System Structure for Software Fault Tolerance.", IEEE Transactions on Software Engineering, June 1975, pp.220-232.

[Rum91] Rumbaugh, J. et al., 'Object-Oriented Modeling and Design', Prentice Hall, New Jersey, 1991.

[Toy87] Toy, W.N., "Fault-Tolerant Computing.", A chapter in Advances in Computers, Vol. 26, Academic Press, 1987, pp.201-279.

[Wil85] Wilson, D., "The STRATUS computer system.", Chapter 12 in T. Anderson ed., 'Resilient Computing Systems' Volume I, John Wiley & Sons Inc., 1985, pp. 45 - 67.

[You89] Yourdon, E., 'Modern Structured Analysis', Yourdon Press, NJ, 1989.

A Methodology for Complex Computer Systems Engineering

Alexander D. Stoyenko* **Harry Crisp†**
Lonnie R. Welch **Robert L. Harrison**

Abstract

Typical among modern applications, such as AEGIS, are (1) integration of multiple existing large systems, as well as development of new systems and subsystems, (2) complex and often conflicting objectives (security, robustness, coherence, real-time and physical distribution), and (3) dynamically adaptive behavior. Toward the goal of automated software synthesis, we define a model for development of complex system software. In our model, the phases of complex systems engineering are ongoing and cyclic, and include specification of requirements; reverse engineering of existing modules and data structures; reengineering of existing systems' designs and implementations; computation of module to table binding strengths, module frequencies, and other metrics necessary for system optimization and assessment; clustering of modules and tables based on bindings; partitioning of clusters among processing elements; assessment of configuration for conformity to requirements and optimality; monitoring and incremental adaptation. We address these concerns in an integrated framework based on a system description language called RT-Chart.

1 Introduction

Protoplasts of the complex systems of today are found among considerably simpler Navy and (non-Navy) real-time computer control systems of the 1940s. Since the 1940s and until the late 1980s, these systems have evolved as what has been understood as traditional or conventional real-time systems. A traditional real-time system is relatively small (or consists of a relatively small number of logical components) and static in nature. The structure of such a system is typically either a cyclic executive or a relatively small number of independent, coarse-grain processes. The system is executed on a small number of processors. In addition to the processors (and their associated computer resources, such as memories and peripherals), the system makes use of a relatively small number of fairly homogeneous non-computer resources (such as sensors or actuators). Finally, while traditional real-time applications have needed to satisfy fault-tolerance and other non-functional requirements, the mechanisms for incorporating these into the corresponding computer systems have been relatively straightforward (such as triple-modular redundancy).

While older Navy applications have necessitated real-time systems of the traditional kind, modern Navy systems are considerably more complex. Typical among modern applications —

*Stoyenko and Welch are with The Real-Time Computing Laboratory, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, NJ 07102 USA, E-mail: {alex | welch }@vienna.njit.edu. The work described herein was performed while Welch was a visiting researcher at NSWCDD. This work is supported in part by the U.S. ONR Grant N00014-92-J-1367, by the U.S. Army Grant DAAL03-91-C-0034, by the NATO Grant CRG-90-1077, by the AT&T UEDP Grant 91-134, and by the NJIT SBR Grants 421250 and 421290.

†Crisp and Harrison are with the Naval Surface Warfare Center, Dahlgren Division, { rdharri | hcrisp}@relay.nswc.navy.mil

such as AEGIS — are (1) integration of multiple existing large systems, as well as development of new systems and subsystems, (2) complex and often conflicting objectives (security, robustness, coherence, real-time and physical distribution), and (3) dynamically adaptive behavior. Consequently, the computer systems that control these applications are required to act accordingly. Specifically, the resulting systems often are very large, and are expected to adapt in a timely, rapid and correct fashion to frequently changing environment variables and conditions. The systems are expected to run on modern computer architectures, which often are (highly) parallel and utilize many heterogeneous resources. Another aspect of such systems is that they are expected to exist for decades, due in part to the tremendous cost of their development. Finally, the systems need to incorporate — in accordance with the requirements of the applications they control — a wide variety of often conflicting functional and non-functional objectives. Given all these requirements, it is natural to refer to these systems as not merely *real-time*, but as *complex*.

In this article we present a methodology for engineering complex computer systems. The methodology views a complex system as an aggregate of software components (processes and objects). The methodology is based on (1) a system-, process- and object-oriented language for specification, design and implementation [11, 9], (2) an umbrella of conceptual system views (one for real-time, another for fault-tolerance and so forth), (3) a software component manager [3, 5], and (4) a set of integrated techniques for synthesizing complex systems from existing and new software systems, for execution on distributed and parallel hardware platforms [9]. Synthesis considers the constraints of the integrated multiple view requirements, while searching for high quality clusterings and assignments of modules to processors. For software systems (or subsystems) being implemented, alternate software component implementations are considered to determine the most suitable for the application. Additionally, alternate hardware configurations are considered.

The remainder of this article is organized as follows. In Section 2 we define a methodology for complex computer systems engineering. Section 3 illustrates the methodology with the AEGIS system. Finally, Section 4 summarizes what has been achieved and provides directions for future work in this promising technology.

2 Complex Computer Systems Engineering

The first portion of this section provides a complex systems engineering framework that resulted from studying the development process employed in the AEGIS system. The second portion of the Section presents a complex systems engineering model that captures this framework. The section concludes with the presentation of a system description language that allows the model to be put into operation in an organized fashion in any complex system engineering project.

2.1 A Framework for Complex Systems Engineering

A complex system is a long-lived and evolving system of systems, having functional requirements and non-functional requirements. Complex systems require coherent behavior at the *macro* level. In fact, we assert that determinism is only needed at the complex system level, in contrast to the contemporary real-time lore, which dictates that all components of a computer system must be deterministic in order for the complete system to be deterministic. Additionally, there must be a high degree of distribution of the systems and their components, yet at the same time there must be a coordination of the systems' activities so that they are smoothly connected. The coordination task is further complicated by the fact that a complex system must exhibit robustness in the presence of faults, maintaining reasonable response times and acceptable functionality. Thus, there is a need to manage redundant subsystems.

We also believe that new classifications are needed for real-time systems, so that the needs of complex systems can be expressed. While in many real-time systems there are processes with genuine hard deadlines, the traditional notion of hard real-time — which states that no process can ever miss a deadline — is an idealistic view of reality. We define hard real-time systems to be those in which (1) the physics of the problem domain defines the timelines, (2) timing constraints must be guaranteed without exception, and (3) there is no slack for timeliness. Hard real-time requirements occur in sensor and actuator systems. Soft-real-time systems are those in which (1) there is coherent behavior at the macro level, (2) timing constraints must be guaranteed without exception, and (3) timeliness must be within specified bounds. Soft real-time requirements occur in macro-level (complex) control systems. A new class of real-time systems is defined to be *easy*, in which (1) there is reasonable performance, (2) exceptions on timing guarantees are permitted, and (3) there are no timeliness requirements. Auxiliary systems are classified as easy real-time.

Complex systems engineers also require the ability to "throttle down" the system, by shedding tasks of low importance during times of overload. It is frequently the case that the exact mix of tasks in a complex system cannot be determined statically. Thus, some system resources may become saturated and a dynamic response to the situation is warranted. By allowing system developers to specify the policy for handling overload, their task is simplified.

Complex systems must be synthesized from many autonomous hardware and software systems, since they bring together many resources that must be joined into a coherent and efficient whole. Toward the goal of automated software synthesis, we define a model for development of complex system software. In our model, the phases of complex systems engineering are ongoing and cyclic, and include the following: specification of requirements; reverse engineering of existing modules and data structures; reengineering of existing systems' designs and implementations; computation of module to table binding strengths, module execution frequencies, and other metrics necessary for system optimization and assessment; clustering of modules and tables based on bindings, and partitioning of clusters among processing elements; assessment of conformity to requirements, optimality, load balancing, etc.; monitoring and incremental adaptation. The remainder of this Section briefly discusses each of these items.

2.2 Requirements Specification

During the first phase, the requirements of the system are specified [4]. This is accomplished by stating the required functionality [4], as well as non-functional requirements such as timing behavior, fault tolerance and security. During this phase, necessary system features are stated in implementation-independent terms, using a technique such as system design factors (SDF) [1]. For example, requirements may state that the data from a particular sensor should be sampled once per second, transformed via FFT, and matched against a set of relevant patterns which cause the task to transmit a message to be sent to a handler routine. Additionally, the reliability of the task could be stated as 0.11 probability of failure, and the output of the task could have the security classification of secret.

2.3 Reverse Engineering and Reengineering

Requirements specification is followed by the task of reverse engineering, which captures the designs of existing systems. The goal of this phase is to identify the essential features of the systems, as defined by the reengineering, design, implementation, and optimization phases. The reverse engineered design must allow reasoning about design and implementation tradeoffs. Thus, it must be multiple leveled, supporting a design-time view and an implementation-time view, with perhaps multiple views within each of these to deal with specific design and imple-

mentation attributes (such as timing, cost, dependability or parallelism). Reverse engineering is a very difficult task to perform, since the system to be reverse engineered may be implemented in non-structured programming and with low-level languages (such as assembly language). Furthermore, the system is likely to be implemented in multiple languages. The use of language constructs such as pointers and memory overlays further complicates design capture. Reverse engineering is a multiple pass activity, proceeding from analysis of low level details, to synthesis. Analysis begins with the examination of single program statements and data elements. Synthesis builds on the analysis results by examining intercomponent relationships (such as a statement accessing a data element). The result of synthesis is an aggregation of the components (into units such as procedures). Successive passes of synthesis result in larger aggregate components or in additional intercomponent relationships, since the output of one or more synthesis phases serves as the input to a later synthesis phase.

The goal of reengineering is to produce a system design and corresponding implementation that satisfy the requirements. It is performed with the goals of (1) correctness (such as deadlock and race condition avoidance and synchronization of access to data shared among processes generated by the reengineering) (2) efficiency (via techniques such as parallelism and code cloning), (3) analyzability and testability (for all system requirements — functionality, timing, etc. — even in the presence of aliasing and unbounded loops), (4) portability (thus avoid platform specifics), and (5) maintainability and adaptability.

2.4 Configuration, Optimization, Assessment, and Metrics Collection

There are many degrees of freedom for which choices must be made in the design, implementation and configuration of a complex system. The configuration refers to the choices of processors and memories, their interconnection, and the distribution of software components and data among the processors and memories. The choices made affect the quality of the resultant system, in terms of meeting constraints and also in terms of the closeness to optimality. Due to the complexity, selection of choices cannot be performed manually. Note also that there are constraints of varying classes. User-defined constraints (such as deadlines and periods for processes) are fixed and cannot be altered by the optimizer. Configuration-defined constraints are flexible, and include items such as processor and link speeds, interconnection topology, and software component versions. Configuration and optimization may consider the following degrees of freedom: clustering of modules and data to be assigned to a processor as a unit; software component to processor assignment [10, 12]; hardware configuration; load balancing; parallelism; communication; and software component version selection [3, 5]. A multipass configuration and optimization strategy is appropriate for complex systems due to the magnitude of the problem. For example, in one pass a clustering of modules based on one criterion such as communication binding can be performed using a fast, coarse greedy algorithm. In a second pass, the clusters can be assigned to processors using an accurate, detailed, high quality optimization technique such as simulated annealing or neural networks. The first pass has the effect of reducing the complexity of the assignment problem, since the number of units to be assigned has been reduced significantly.

To determine the acceptability of a particular system configuration, and to guide the optimization phase, the system's characteristics are assessed with respect to the system requirements [2, 6, 7, 8, 9, 11]. Conformity to timing, dependability, security and other requirements is checked. A detailed set of techniques for checking timing conformity of complex systems is presented in [9, 11]. The paper presents techniques for predicting response times of independent real-time processes that are distributed over many processing elements (PEs). The techniques estimate contention for PEs (CPUs) and network communication links. The contention is combined with utilizations (based on frequency of execution) to compute a rate of progress experienced by clients of each device. Response times are computed as the product

of rate of progress and computational demand. The predictions have recently been enhanced to apply to systems in which asynchronous remote procedure calls are allowed, thus permitting parallelism within a task to be modeled accurately.

To evaluate the quality of a particular set of system designs and implementations, under a chosen configuration, the values of various system attributes need to be collected [9, 10, 11]. The collection of the metrics is (in part) a static process, wherein compiler-based tools extract information such as the set of global tables read or written by each procedure, or the interprocedure call graph. Additional information is collected by monitoring the behavior of the system during execution. The dynamic monitoring can provide such valuable information as the amount of data read from a particular global table by a given procedure, or the probability of taking a particular branch of a conditional.

2.5 System Description Language

To meet the challenge of complex systems engineering, we propose an integrated multi-view methodology. The notion of multiple views is not new. In particular, five essential *conceptual* views are presented in [1]. However, our views are *operational* (they correspond to operational requirements) rather than conceptual, and include (but are not limited to) the following views.

1. The Functional view presents a system in terms of active processes and their dependencies (through usage of passive resources and direct interactions).

2. The Timing view presents time-constraints of each process, and each resource or interaction requirement (of an action). For instance, a particular process may be strictly periodic with a deadline at the end of each period and may require a particular resource for a certain amount of time in every period.

3. The Fault-Tolerance view presents the fault-tolerance and reliability requirements of each process and each resource usage or interaction. For instance, a particular process may need to be replicated and run on two physically separated CPUs.

4. The Security view presents security requirements of the system. For instance, it may require a particular degree of clearance to access a particular resource.

As the development of a system matures — throughout specification, design, implementation, maintenance, and even during execution of dynamically adapting systems — the treatment of each operational requirement will naturally include increasingly more complexity. To accommodate such complexity, the corresponding view will support a hierarchical definition of the operational view of the requirement. To express the view, a rigorous description language called RT-Chart is provided.

Each of the operational views can be mapped onto and from each conceptual view (though not every map may necessarily be equally useful). In judging an overall design, it is in fact be useful to map different operational views onto and from a corresponding conceptual view. We will provide "useful" maps from operational views onto and from conceptual views. Note that RT-Chart can serves as an intermediate form for which analysis and optimization techniques exist [11, 9], however, the actual language used to express the system properties may be RT-Chart but need not be RT-Chart. One can define maps from other description languages to RT-Chart, and the mapping can be automated. In fact, we have defined maps from Ada-package-based and object-oriented systems to RT-Chart.

Different requirements of a system may be addressed by different teams of engineers. Each team should thus typically develop and appreciate its own, specific operational system view.

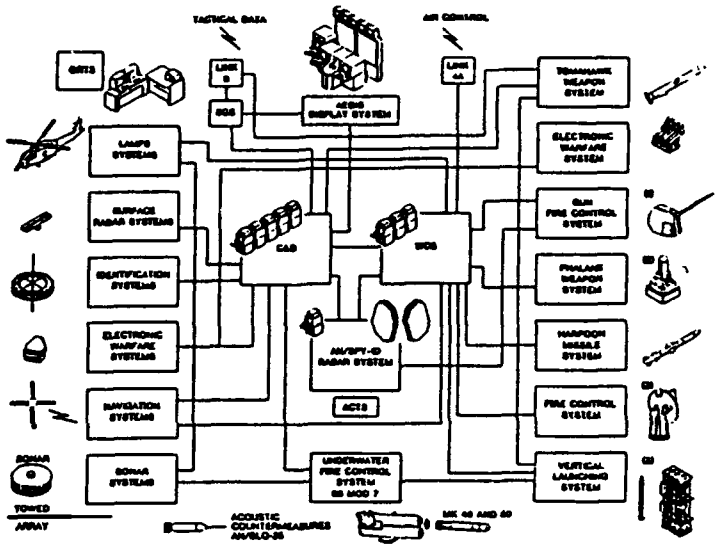


Figure 1: AEGIS Combat System.

To integrate and maintain consistency among the different operational views, another set of maps — this time from one operational view and its corresponding description language into another—are provided, along with consistency checking rules. The actual implementation of these maps uses the maps between operational and conceptual views. Furthermore, to support the hierarchy within each operational view, the hierarchical maps are used along with additional mapping rules to ensure that maps across methodological views involve "compatible" levels of the hierarchy.

RT-Chart specifications represent a system as a set of periodic and event-drive processes, each composed of a set of actions. The first action of a process is performed at the start of each process activation, as determined by the beginning of the process's period or by the occurrence of the event driving the process. Upon completion, the initial action invokes a successor action, passing some data. RT-Chart provides a resource algebra for stating the modes in which resources can be used: (1) may be used concurrently (2) must be used concurrently (3) cannot be used concurrently and (4) cannot be used concurrently and must be used in a particular order. Additionally, and-gates allow the specification of parallel actions, and or-gates indicate conditional execution of one among a set of actions. Furthermore, in RT-Chart, the actions may be hierarchical, to enable macro-level reasoning and specification. We have found it useful to describe detailed implementations of RT-Chart actions as objects and packages. In addition to functionality, timing and parallelism, there are other important system aspects that RT-Chart allows to be expressed. Security classification levels can be indicated for information flows, code (actions), resources, levels of hierarchy, and implementation details. To allow dependability to be dealt with, degree of redundancy or reliability can be specified for actions or processes. Another aspect of systems is relative criticality, which can also be expressed for actions and processes.

3 A Military Example

The United States Navy has successfully developed and deployed complex systems. An example is the AEGIS combat system (see Figure 3). AEGIS is a complex system, composed of many systems. The main categories of systems composing AEGIS are (1) detect (2) control, and (3) engage. The detect category consists of the LAMPS, surface radar, identification, electronic sensing, navigation, and sonar systems. The control category of systems includes C&D,

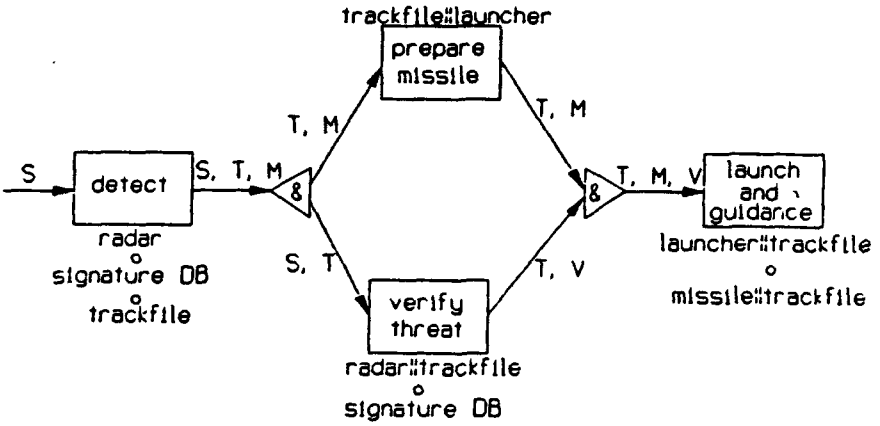


Figure 2: Auto-special control.

WCS, SPY radar, ACTS, and ORTS. In the engage category of systems, there are the LAMPS, electronic warfare, GUN weapon, fire control, vertical launching, advanced tomahawk weapon control, phalanx weapon, and underwater fire control.

To illustrate the methodology of the previous section, consider the anti-air warfare (AAW) engagement control system of AEGIS. It provides the capability of automatic identification and engagement of quickly evolving threats. Obviously, these tasks must be performed according to rigid timing requirements, with little variance. An early indication of of possible quick reaction targets is given by the radar system. Detection of such targets triggers a high priority set of actions, consisting of missile preparation, further classification of threat, missile firing and guidance (or abortion of firing sequence if the target is determined to be nonhostile).

This set of actions can be modeled in an RT-Chart specification as shown in Figure 3. The process is activated by a radar signal (S) arriving at the detect action. To detect, the radar unit, signature DB, and trackfile are used sequentially (as indicated by the "o" resource usage operator). Upon detection, a triple (S, T, M) is sent to the &-gate which forks control to concurrent actions. The triple contains information describing the radar signal (S), the track identifier (T) and the missile launcher and missile type (M) to be used. The prepare missile action can use the trackfile and launcher concurrently. The verify threat action, running concurrently to the prepare missile action, uses the radar unit and the trackfile concurrently and then uses the signature DB. Threat verification sends the value (V)—a confirmation or a negation of the preliminary detection—to the launch and guidance action. The launch and guidance action uses the launcher and trackfile concurrently, and then uses the missile and trackfile concurrently.

Following the firing of a missile, uplink commands are periodically sent to the missile to control its intercept trajectory. The appropriate trajectory is determined by considering the speeds and positions of the target and the missile, and extrapolating the position of the target. The missile guidance action breaks down further, into a set of actions: measure target and missile positions, calculate missile correction factor, build uplink command to be sent to missile, transmit uplink command to missile. The missile guidance must be performed periodically to allow the missile to intercept the desired target. The deadline for the uplink command to be received by the missile is stringent, since it is programmed to self-destruct if it receives no such command (to avoid destroying the wrong object). Missing of such a deadline is highly undesirable, since it results in the waste of a missile and also places the combatant at considerable risk. Parallel processing is appropriate to meet such timing requirements, since there may be

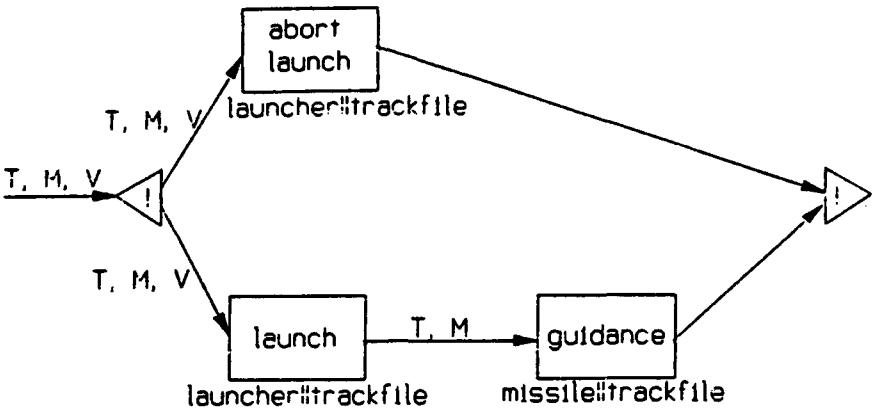


Figure 3: Missile control process.

multiple missiles in flight and since the speed of missiles is ever improving and therefore timing requirements become more stringent.

This launch and guidance actions can be modeled in an RT-Chart specification as shown in Figure 3. The task performed is either (as indicated by the !-gate) an abortion of launch if the hostile target was not verified, or is a launch and successive guidance of the missile.

4 A Last Word

The engineering of complex systems requires the integrated solutions of problems related to parallel and distributed processing, real-time, security, and dependability. The framework, model, and methodology embodied in this paper addresses these concerns by building on the experiences of the AEGIS development team and the members of the Real-Time Computing Laboratory at NJIT. The result is a framework and corresponding approach that allow complex systems to be specified, designed, configured, evaluated and maintained.

In our work so far, we have addressed the Functional and Timing operational views and have explored their relationship with the Implementation conceptual view. We have also defined a specification language RT-Spec and a design and implementation language RT-Chart for expressing operational and conceptual views. So far, we have not incorporated a hierarchy into the Functional and Timing views nor into the RT-Spec/RT-Chart semantics, though work on this is in progress. Additionally, the reengineering and metrics collection areas are in their infancy, requiring much additional research. Another problem we are continuing to address is the integration of the solution to the optimization problem into the complex systems model described in this document. We are also exploring techniques for managing libraries of reusable software components. Tools incorporating the techniques are also being evolved with the research.

We would like to thank all members of the SSSWG, NJIT RTCL members, and the HiPerD team for the influences they have had on the ideas embodied in this document.

References

[1] N. T. Hoang, "The Essential Views of Systems Development," *Proceedings of 1991 Systems Design Synthesis*

Technology Workshop, Naval Surface Warfare Center, Silver Spring, Maryland, pp. 3–9, September 1991.

[2] E. Kligerman, A. D. Stoyenko, "Real-Time Euclid: A Language for Reliable Real-Time Systems," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 9, pp. 940-949, September 1986.

[3] W. Rossak, A. D. Stoyenko and L. R. Welch, "The Component Manager: A Hybrid Reuse Tool Supporting Interactive and Automated Retrieval of Software Components," *Proceedings of 1992 Complex Systems Design Technology Workshop*, Naval Surface Warfare Center, Silver Spring, Maryland, July 1992.

[4] M. Sitaraman, L. R. Welch, D. E. Harms, "Influences of a Component-Based Industry on the Expression of Specifications of Reusable Software," *The International Journal of Software Engineering and Knowledge Engineering*, June 1993.

[5] R. A. Steigerwald and L. R. Welch, "Reusable Component Retrieval for Real-Time Applications," *Proceedings of IEEE Workshop on Real-Time Applications*, N. Y., N. Y., May 1993.

[6] A. D. Stoyenko, V. C. Hamacher, R. C. Holt, "Analyzing Hard-Real-Time Programs for Guaranteed Schedulability," *IEEE Transactions on Software Engineering*, pp. 737–750, SE-17, No. 8, August 1991.

[7] A. D. Stoyenko, T. J. Marlowe, "Polynomial-Time Transformations and Schedulability Analysis of Parallel Real-Time Programs with Restricted Resource Contention," *Journal of Real-Time Systems*, Volume 4, Number 4, Fall 1992.

[8] A. D. Stoyenko, T. J. Marlowe, W. A. Halang, M. Younis, "Enabling Efficient Schedulability Analysis through Conditional Linking and Program Transformations," *Control Engineering Practice*, Volume 1, Number 1, 1993.

[9] A. D. Stoyenko, L. R. Welch, "Response Time Prediction in Object-Based, Parallel Embedded Systems," to appear in *Euromicro Journal*, 1993, Special Issue on Parallel Processing in Embedded Real-Time Systems.

[10] L. R. Welch, *Architectural Support for, and Parallel Execution of, Programs Constructed from Reusable Software Components*, Ph.D. thesis, Department of Computer Science, The Ohio State University, December 1990.

[11] L. R. Welch, A. D. Stoyenko, T. Marlowe, "Modeling Resource Contention for Distributed Periodic Processes," *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, December, 1992.

[12] L. R. Welch, A. D. Stoyenko and S. Chen, "Assignment of ADT Modules with Random Neural Networks," *The Hawaii International Conference on System Sciences*, IEEE, Jan. 1993.

An Assessment Control Board (ACB)

and a

System Integration (SI) Program

as complements to

The Configuration Control Board (CCB)

Richard Evans

May 1993

An Assessment Control Board (ACB)

and a

System Integration (SI) Program

as complements to

The Configuration Control Board (CCB)

Overview

Systems development, especially in DoD programs, typically includes a basic program control board known as the Configuration Control Board (CCB). CCB membership is often a combination of both contractor and user/government personnel, or else each organization chairs their own CCB, with participation/attendance open to the other. The roles of the CCB can vary depending on whether it is developer or user/government managed; but in both organizations the CCB has the common function of control of program configuration change, particularly for the baseline documents that establish the requirements, design, and testing. A developer CCB controls the change nomination. A user/government CCB controls the changes and their associated funding and schedule adjustments, if any. CCBs act after-the-fact in the sense that they receive formal change proposals in specific formats, some of which may have been in prior preparation for months.

A key to achieving assessments is a management complement to the traditional Configuration Control Board (CCB) that takes the form of an Assessment Control Board (ACB). An ACB is a complementary and contrasting program control board that has been applied on several major system [software and hardware] development projects. A summary of the complementary nature of, and contrast between, an ACB and the traditional CCB might be equated to the quote on talent [CCB] and tact [ACB] in an anonymous quote published in McGuffey's Sixth Eclectic Reader, Van Nostrand, circa 1878--pp 113:

Talent and Tact

"Talent is power, tact is skill. Talent has weight, tact is momentum. Talent knows what to do, tact knows how to do it. Talent is wealth, tact is ready money. Talent sees its way clearly, but tact is first at its' journeys end. Talent convinces, tact converts. Take them to court and talent feels its weight, tact finds its way. Talent commands, tact is obeyed. Unless they are combined we have successful pieces which are not respectable, and respectable pieces which are not successful."

The thesis of this paper is that there is a need for both talent and tact: both assessment control [ACB] and configuration control [ACB]. Assessment makes discoveries, the CCB disciplines the application of those discoveries. Assessment anticipates and plans, the CCB operates after-the-fact and regulates. While CCBs are as essential as talent, they are as equally in need of the balance of tact [assessment]. CCBs alone can be deficient in three ways:

1. The CCB members involve themselves more in the detailed change proposed rather than in the control of the process;

2. The CCB's scope is typically inappropriately constrained to program change, and that is all too often, when presented, essentially defacto—as the majority of the investigation resources [both time and money] have by then been used; and

3. The CCB does not meet early enough to recognize and address risks as they develop and are, at that point in time, still controllable [time is the development's one inelastic resource].

An ACB, on the other hand, would meet weekly, and only briefly, and focus on the control of the assessments, in the form of one-page Assessment Plans (APs) and the results that are also one-page Assessment Reports (ARs). The ACB focus on assessments would contrast with and complement the CCB approval of proposed changes. As illustrated in Figure 1, the ACB controls the work before it ever starts; the CCB controls the implementation of change, where the design of the change has already occurred.

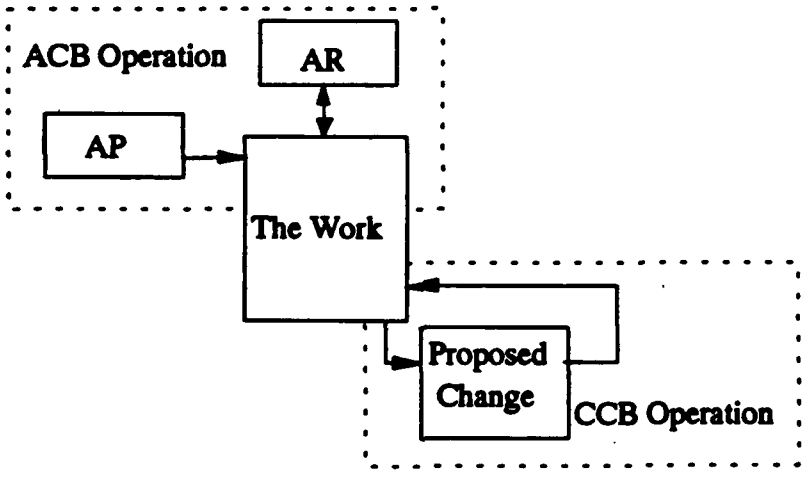


Figure 1 ACB and CCB Operations

Assessment Plans (APs)

Assessment Plans (APs) are one page, with the following attributes:

1. Scope

   The work and the associated products to be assessed

2. Assessment Criteria

The criteria to be applied in assessing the work and the products. This is one of the hardest element of a plan to devise, and accordingly one of the most critical program controls. .

3. Approach

How will the assessment itself be assessed,

How will the assessment be conducted--the format and process

Who will be on the "separate/independent" assessment team--their names

4. Schedule and cost

The assessment milestones and the proposed investment in assessment

The ACB, by approval of the one-page APs, exercises the essential "tact" influence of a project, as a complement to the CCB "talent". By directing the plans for assessment, the ACB directs the future, in addition to controlling it.

System Integration (SI) Program

A parallel development concept that can be applied to strengthen the CCB is for the ACB to also sponsor an SI program for CCB control. The objective of an SI Program is to assure that proposed changes are well prepared for CCB consideration. Changes may be changes to the configuration of the program architecture, and schedule as well as a change to the design. There are three primary dimensions of an SI program, collectively they are known as the three Is, they are illustrated in Figure 2

1. Identification
2. Investigation
3. Implementation

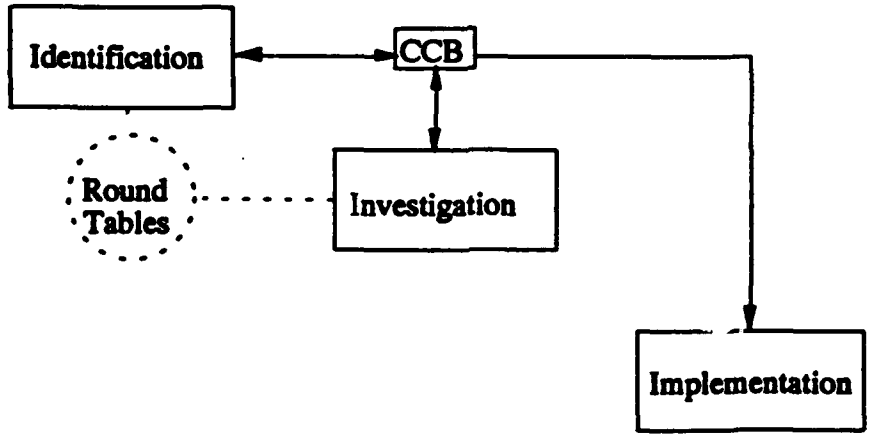


Figure 2--ACB-Sponsored and CCB-Controlled SI Program

The driving influence of the SI Program is in the first two "Is": Identification and Investigation. These are supported by a four-part SI Program structure that is illustrated in Figure 3:

1. Use of System Reports (SRs) as individually numbered records of every problem, suggestion, insight, or idea. An SI Database is built on the ever-accumulating set of SRs. SRs are recorded as symptoms, so to speak, without prejudice. They are not filtered by any criteria, such as who said, or how they were reported, or whether they were validated. They are accumulated and honored by a unique SR Number that is never reused. Thus, while the SR may be placed in an inactive file, its identity, its number, always remains unique to that SR.

2. The use of non-representation Problem Area (PA) teams to assess the overall program handling of the SRs. The team members are drawn from both the user/government and the developer and serve as professional collateral assignments and not as representatives of their parent organization's management priorities or interests. The PA Teams assess, they do not have responsibility for solutions. They recommend initiatives, but they do not sponsor changes to the CCB--with the attendant responsibility to implement approved changes. The PAs monitor the process, both its design and operation

3. Candidate Program Initiatives (CPIs). CPIs are temporary homes for potential program initiatives. CPIs are unfunded and without a designated management responsibility. They are the initial planning framework, neural territory, for the allocation of SRs. Noting that SRs are allocated redundantly, with one primary allocation and multiple secondary assignments.

4. Program Objectives (POs). POs are funded, have assigned implementation responsibility, and are the formal vehicles for configuration change. POs are assembled as the implementation packages from the array of CPIs. They may be one entire CPI or include portions of many.

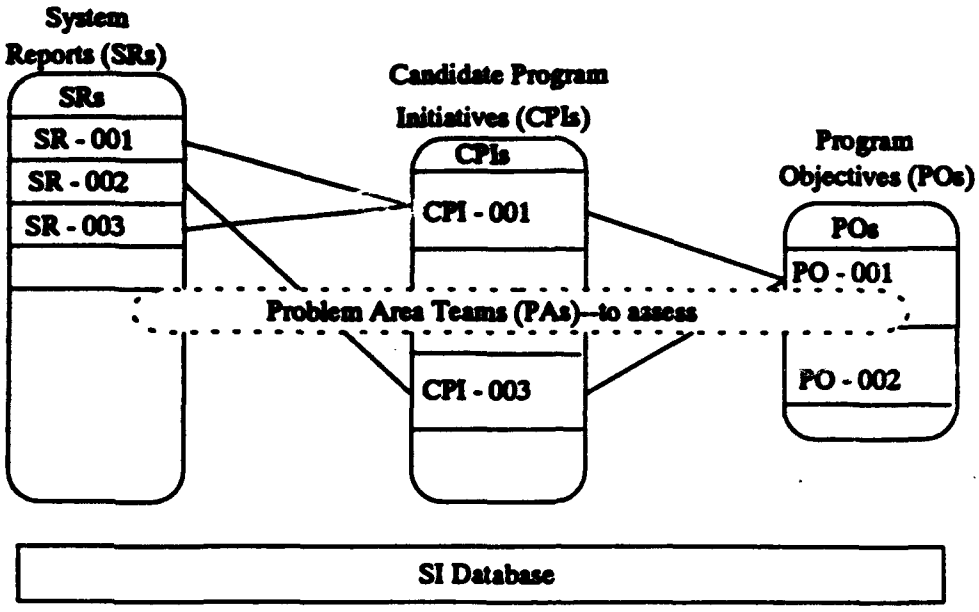


Figure 3 SI Program

The Investigation Process has a central feature of the use of Round Tables (RTs) to strengthen the investigation. The Investigation Plan (IP) is considered by a panel of three to five experts. The RT is the assessment team for the planned investigation. The IP includes [as item 4] an Assessment Plan [approved by the ACB] as a key part of the Investigation process. The IP defines, for the RT consideration, the following top-level attributes of the planned investigation:

1. Problem/Opportunity A summary of the problem or opportunity to be addressed.

2. Approach The key features of the approach to the investigation effort.

3. Resources Staff-Months, Schedule for the activity, and an estimate of the "ranges" of the Implementation/Life Cycle Costs or Implementation Resources Requirements Range (IR3) that span the high-low for three parameters: Duration [how long to build], CY [calendar year for start] and costs.

4. Assessment The assessment plan, identifying the proposed assessment team members, the plans and products to be assessed [assessment milestones] and the principal criteria to be applied in those assessments.

5 References. Available background material--what is already known about the problem and alternative solutions and approaches.

6. Products and Schedule The principal deliverables--the tasks to be accomplished--with their milestone schedules.

The CCB controls the transition from Investigation to Implementation. By operating an SI Program, the ACB supports the CCB with proposed changes that have been appropriately considered, with up-front review by an RT, that can include a public "hearing" format where all interested parties can be apprised of the planned investigations.

Summary

Assessment plans and reports are necessary complements to program control through CCBs. CCBs are essentially totally after-the-fact. The resources [both time and money] to prepare the proposed changes has already been invested by the time the CCB receives them for consideration. The operation of an ACB is management working up front, where the leverage is greatest. The ACB works with one-page Assessment Plan (AP) summaries for the planned assessment of all work. The scope of some APs may be very large, others very small, as the ACB approves. Some work may be small in dollar scope but large in significance. The ACB influence of how work is to be assessed is a prime lever on what is to be done. The criteria for "goodness" and the names of those who will prepare the assessment report are key management influencers. The Assessment Report (AR) is generally succinct, not belabored. Does the work / product meet the pre-established criteria? The plans for the AR is itself part of the AP that the ACB controls.

Operation of a planning process, called an SI Program, based on SRs as the primitives for all planning, is a potential added aid to the CCB. With an SI Program, the CCB receives proposed changes that have been identified and investigated with care--by pre-planned assessments. Further, an SI Program works with bipartisan Problem Area (PA) Teams [that include both customer and developer members-- not as representatives of their organizations, but for their expertise only] to monitor the implementation and the planning from the perspective of their "problem area". They particularly concern themselves, working on a very low duty cycle--an hour a week or less, with the planning to address their assigned SRs. What is being done with the symptoms [a synonym for system report or SR] they have monitoring cognizance for.

Investigations are as critical as the assessment of assigned work. How investigations are to be accomplished is a key concern of the ACB. A central feature of the Investigation Plan (IP) is the Assessment Plan (AP) for that investigation. A built-in feature of every Investigation AP is the use of a Round Table to provide up-front assessment of the planned investigation--an assessment of the IP itself. The one-hour or less [or not as a face-to-face meeting if preferred] meeting of the RT members assures that the planned investigation is well-considered. While the primary need for an ACB type focus is on the plans for Investigations, nominated by one-page Investigation Plans (IPs), an ACB-controlled operation of assessment teams could also be effectively applied across the full span of the software lifecycle. The idea is that, at least in addition to, [and possibly in some cases even in lieu of] the after-the-fact milestones of System Design Reviews (SDRs) and Preliminary Design Reviews (PDRs) there could be a concentrated focus where the leverage is greatest: on the plans for the conduct of each phase.

A Generic Object Oriented Conceptual Pivot Model

Naoufel Kraïem

Laboratory MASI/CRI, University of Paris I
17 rue de Tolbiac, 75013 Paris
naoufel@masi.ibp.fr.
Phone :+(33).1.44.24.93.65
Fax :+(33).1.44.86.76.66

Abstract: *The emergence of the object philosophy in the new software development techniques gave birth to many object models. The object-oriented approach enables the improvement of software quality, the reduction of future maintenance requirements, the reuse and the adaptation of specification and developments. However the difficulty lies in the transition between the conceptual specification and the implementation because of the disparity of the formalism proper to each level. To resolve the problem, we propose an object oriented interface supported by a software tool and based on a pivot model and a set of mapping rules. This research has been developed within the framework of the ESPRIT II project named Business Class*[1]

Keywords: object-oriented analysis, object-oriented design, software engineering environment, information systems development.

1- INTRODUCTION:

The Object-Oriented approach emerges in certain number of data processing domains, such as programming, software engineering, data base, DBMS, analysis and design of data base and information system. The paradigms underlying the computational object-oriented are stabilized enough to consider that they are providing a unifying approach for information system development.

Object-oriented requirements, analysis and specification models are henceforth the subject of an active research effort. Indeed, object-oriented design techniques (e.g. [Booch87], [Heitz89], [Meyer90], [Wirfs-Brocks90]) do not resolve all the problems of object-oriented development in the information system domain. Object-oriented design deals with the solution space, the "how" of technical design, and generally emphasize the organization and reuse of code. Object-oriented analysis is concerned in the problem space, the "what's" of requirements' specification, and centers on the semantics of the phenomena, postponing technology-dependent choices.

Object-oriented design methodologies are focusing on system design as a later stage of the application life cycle, implying that the earliest stage leading to requirement's specification and conceptual design, have been performed.

Object-oriented analysis methodologies are still under investigation. Three main approaches are being proposed:

- the functional approach uses traditional DFD based techniques to derive object specification
- the data driven approaches are influenced by E/R modelling to define objects
- the object based approaches recommend the use of the object concept right from the beginning of the system life cycle. The concept of object is then the basic element the system relies on.

The claim of these approaches is that enhancements and extensions of the computational object concept are required to make it relevant to conceptual modelling.

O* [Brunet91], MCO [Castellani93], (OOD, GOOD) [Booch86,87] HOOD[Heitz89], OMT [Rambaugh & al.91], OOA[Coad & Yourdan91], and OOSA [Shlaer&Mellor91] are examples of approaches to support conceptual modelling in an object-oriented way.

The paper aims at presenting an oriented conceptual modelling and object-oriented implementation. To do so, we propose an interface supported by a software tool and based on a generic object-oriented conceptual pivot model and a set of mapping rules.

The problem of the design and the implementation of information systems, was an impetus for the development of framework for object pivot model, based on a general metamodel and Object Oriented Conceptual Pivot Model (OOCPM). The following

[1] This project is supported by the European Commission under the contract 5311 of the second European Strategic Program for Research and Development in Information Technology (ESPRIT). The main partners include Télésystèmes (France), Société des Outils du Logiciel (France), Applied Logic (United Kingdom), Eritel (Spain) and Datamont (Italy). Université de Paris 1 involves in the development of the analysis environment and the integration of the tool in the PCTE-based software engineering environment.

section presents the object oriented conceptual pivot model. Section 3 illustrates the framework and a set of mapping rules by means of examples.

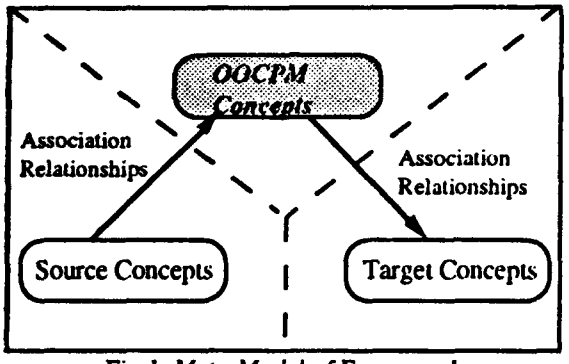


Fig 1: Meta-Model of Framework

2- The Object-Oriented Conceptual Pivot Model:

Modelisation of the real world consists of regrouping objects in classes of same type and describing relationships that can exist between them.
Developing an information system, real world classes (or entities) are modeled by corresponding objects in information system. For describing the data and behavior of an information system it should be decided which classes are relevant, and which behavior is to be expected from those classes. The behavior of information system is a reaction of one or more individual objects to external events detected by the system. The behavior of object depends on the kind of event, its internal state and possibly the internal states of other objects with which it may have relationships.
In specifying objects we distinguish static and dynamic aspects. Static aspect concerns the attributes of objects and the relationships between classes; dynamic aspect concerns, the actions or operation executed by or on classes.

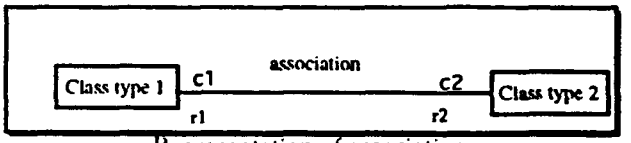
2.1- Static Model :

Our approach in systems development is to support the systems analysis and other models by means of diagram techniques in order to capture the different aspects of information system. After the modelling of a pivot conceptual model, it is possible to map the OOCPM description to different database environments and programming languages. Our aim is to bridge the gap between object-oriented conceptual modelling and object-oriented implementation. To do so, we propose an interface supported by a software tool and based on a pivot model and a set of mapping rules. Then that database schemes and code can be generated automatically. So the object-oriented approach in systems development is applied in principle without restrictions caused by implementation environment, whether it is object-oriented or not.
For representation of the aspects of an OOCPM we use diagram techniques:

- Class relationships diagram for the static structures (static representation).
- Dynamic diagram (dynamic representation).

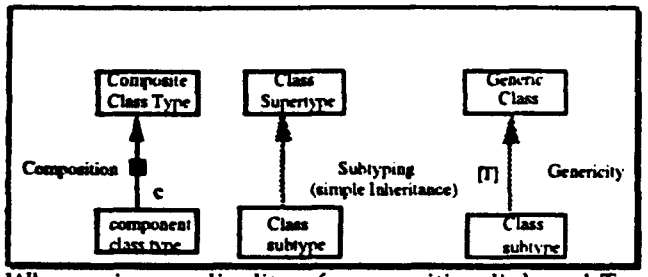
To represent the conceptual static structure, we propose an extension of E/R model [Chen76], because its diagrams are well-known, usually used, have great expressive power and used by the most object-oriented methods (OOSA [Shlaer & Mellor 91], OOA [Coad & Yourdan91], OMT [Rambaugh & al91]. These methods use E/R model to let the designers and analysts who are already familiar with this model to have the impression of not changing their habits.
The model that we present, solves some of the limitations of the existing models (e.g. the model E/R take into account the modelling of the structural and static aspects of a real-world system; the aspects related to the evolution of data and the dynamic aspects are not tackled).
The proposed model is a conceptual one, it independed of all implementation issues. It takes into account not only the static or structural aspect but also the dynamic or behavioral aspect.
It generalizes the object-oriented conceptual models such as the OOD [Booch91], OMT [Rambaugh & al. 91], OOSA [Shlaer & Mellor 91], OOA [Coad & Yourdan91], for the conceptual modelling and extend then to the modelling of the functioning of the methods. The order of the organization of constraint is not tackled.

To represent our model, we select notations for expressing class type, association, composition, roles, subtyping (specialization/generalization) and genericity. These figures below represent the static relationship types:



Representation of association

Where c_1 and c_2 are cardinality constraints and r1 and r2 represented roles.



Where c is a cardinality of composition link and T represents the type of generic class.

Association:

The associations establish relations between objects. They are expressed between, classes and they may be binary [Rambaugh & al91]. They are used to describe the conceptual relation that binds objects together. They can possess their own attributes and become classes if some operations are associated by the designer.

Cardinality ratio and participation constraint are expressed by structural constraints represented by a pair of integer numbers (n, m), where n is a min and m is a max, with each participation of a class type C in an association type A, where $0 \leq n \leq m$. This means that for each object O in C, O must participate in at least n and at most m association instances in A always. This convention is the same as the one introduced by Elmasri [Elmasri90].

The Roles represent a temporal behavior of object [Pernici90]. Classes may play several roles simultaneously.

For instance, in figure 6, the object *Order* is associated to only one *Client*.

Composition:

The link of aggregation defines "part-of" relationship between an aggregate object and one or many component objects. It can be considered as a special link of association but with stronger constraints.

Notice that, two objects having an independent life-cycle should not be linked by a link of aggregation but by an association one [Rumbaugh& al91]. The link of aggregation is a current concept of the object-oriented analysis methods [Manfredi89], [Henderson-Sellers91], [Teisseire91]. It is defined by composition link in our model.

The semantics of a composition link between a composite object and a component object is that the composite object is composed of the component object, which is strongly dependent and belongs exclusively to the composite object. A component object cannot be shared by or changed of composite object.

The composition link induces a strong coupling of the dynamic characteristics associated with the composite and the component object. It is resumed in the following rules :

The creation of a composite object implies the creation of at least one component object. The deletion of a composite object implies the deletion of all its components. Creation, modification and deletion of component objects may be achieved only through the composite object.

For example, in figure 6, *Account* is a part of *Client*.

Inheritance:

Subtyping or the inheritance relationship aims to factor out the structural properties, behavioral properties and the constraints common with several classes (sub classes) in a class in a higher hierarchy (super-classes).

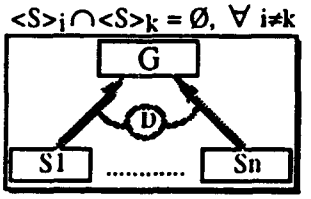
The inheritance considered is the inheritance by inclusion that has the "is-a" or "is like of" semantic between different elements:

Generalized object structure $\subset$ Specialized object structure

Generalized object behavior $\subset$ Specialized object behavior

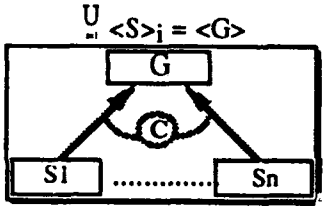
As a consequence, it is a semantic link relating two objects of the classes on which it is defined. It does not presuppose the exclusivity of the specialization. Three types of inheritance constraints may be defined with our model. They restrict the possibilities of existence of the objects of several specialized classes, for each object of the generalized class [Brunet 91] :

Disjunction constraint: A disjunction constraint between several specialized classes expresses that the intersection of their extensions is empty[2].



For instance, the classes *Car* and *Van* both inherit from the class *Vehicle*. The disjunction constraint implies that a vehicle is either a car or a van, or something else.

Covering constraint: A covering constraint between several specialized classes expresses that the union of their extensions is equal to the extension of the generalized class.

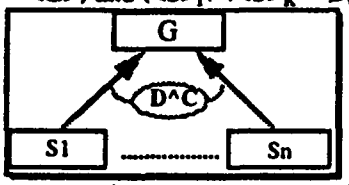


[2] Let: G be a generalized class (abstract or persistent) and S_i a specialized class i (a subclass of G). <G>: the set of G instances. $\langle S\rangle_i$: the set of S_i instances.

For instance, if the classes *Client* and *Supplier* both inherit from *Person*, this constraint stipulates that each person must be a client or a supplier (or both)

Notice also, it is possible to combine many constraints. For instance the O* partition constraint it is represented by a disjunction and a covering constraint (i.e. each instance of the generalized class is specialized in one and only one of the specialized classes. For example, a person is either a man or a woman).

$(\dot{\bigcup}_{i} <S>_i = <G>)$ and $(<S>_i \cap <S>_k = \emptyset, \forall i \neq k)$



With these constraints, we are covering all the types of constraints that exist into the most models.

Genericity:
The genericity provides a way to parameters classes. A parametrized class, or so-called generic class is a class that is being used as a model by other classes. It consists of generic parameters and cannot be instanciated directly. The definition of a parameter's class is derived from the classes parametrized, in which it provides a value to the parameters. The notion of genericity is supported by many object-oriented programming languages. For others, the inheritance can pratically support any thing allows to carry out the genericity [Meyer 90]. But the inheritance and genericity simultanously is considered as an utility in practice [Booch91]. For our model, typing would be meaningless without the possibility of defining generic classes. A generic class is one that has one or more parameters representing types. A class with one generic parameter is declared under the form

Class <class-name> [T]

And used by clients in declaration of the form :

x: <class-name> [A]

Where A is some type.

Constraints:
Some kinds of static constraints, which concern relationships between two objects(e.g. Cardinality constraints, association constraints) are specified by instanciation of the composition and association concepts. Other static constraints that are local to an object may be expressed by equations on the scheme properties and associations. They are specified in the constraint's item of the class, as illustrated in figure 7.

The Integrity constraints are specified to ensure those attribute values, states and behavior of objects in an information system accurately model corresponding real word object.

Properties:
A class is composed of properties that determine its structure.
A property is strongly depended of the owner object, and determines one of its characteristics. A property may be changed only by using the operations defined in the class. It is defined by a name and a set of values. It may take its values either in a domain. The types of data, or domains, which are used in our model are those which are used in the object-oriented models: the pre-defined types (integer, real, string, text, ...), Enumerated types defined by the analysts, and the pre-defined types to which can be associated an interval or a set of rules. The most well-know object-oriented analysis methods today [Booch91], [Coad&Yourdan90], [Rumbaugh&al91], [Shlaer &Mellor91] do not define more complex data. Notice that, our model supports the multiple domain as the most programming languages.

2.2- Dynamic Model :

The elements to be represented firstly are the actions. They are executed in the organization in terms of management rules. These actions modify the state of the elements. They are invoked during some precised situations: an arrival message coming from outside, a noteworthy change of state that happens in the organization, or a previous temporal situation.
These situations correspond to events. The concept of event is used in several methods such as IDA [DeAntonellis81], Remora [Rolland82], O* [Brunet91].
Actions and events are the two key concepts of modelling of the dynamics of own model.
The actions and the events of the Universe of Discourse are represented by the concepts of operations and events which are described as follows :

Events:
The event concept [Rolland88], [Brunet91] is introduced in order to model the dynamic aspects of the application domain. Operation expresses how objects change. Event explains why they undergo changes.
An event occurs when a noteworthy state change happens either in the environment, in one object of the information system itself, or corresponding to a predetermined time. It triggers one or several operations on one or several objects. An event stimulated by the environment is referred to as an

external event, an event due to an object state change is called internal event, and the third kind of event is called temporal event. An event has a name. Its definition includes a predicate part which specifies its occurrence condition, and a triggering part which specifies the operations to be triggered with their associated conditions and iterations.
The advantages of the event concept in the object definition are as follows :

- All the static and dynamic phenomena are specified into classes; object encapsulation is realized by operation and events.
- behavior and operational dependencies are clearly specified
- Events lead to study local situation fully delimited by state change of only one object.

Our notion of event is similar to the one described in O* model [Brunet91]. It is textually described below the keyword "event".

Operations:
The evolution of the objects is effected by executions of operations.
Following the encapsulation principle of the object-oriented paradigm, the only way of affecting (i.e. creating, modifying or deleting) an object is to execute an operation specified on the class. Figure 2 illustrate an example of graphical representation of OOCPM operations.

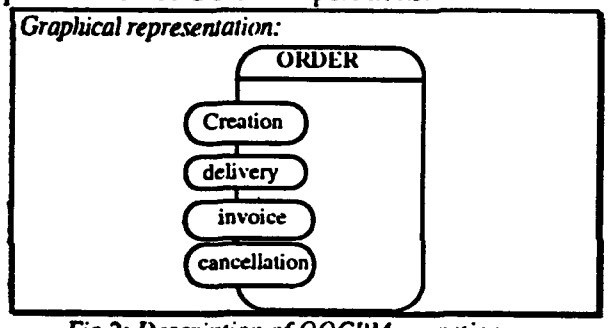


Fig 2: Description of OOCPM operation

The execution of an operation is always the result of an event occurrence. Different events may trigger the same operation.

Dynamic link:
The utilization link establishes a path allowing the message reading of an object to another object. The semantic of the utilization link is that a client object uses the services of a server or supplier object. It is called the client-server link in the OOA [Coad90] model. In our model, this link is translated by a dynamic link.

The dynamic relationships among objects are explicitly specified through events and simultaneous triggering of operations. It is important to make explicit these relationships at conceptual model because an important part of the real world complexity is due to these dynamic interactions [Brunet91].

Behavioral constraints
Behavioral constraints are concerning the dynamic aspects of a class: they restrict the possibilities of execution of some operations. As for structural constraints described before, behavioral constraints are local to a class.

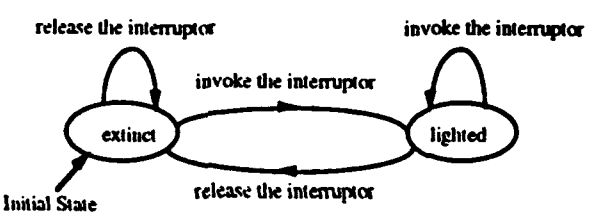
We chose to specify the dynamic constraints by the way of state transition graph, rather that by the assertions of first order [Sernadas89] or temporal logic [Jungclaus91], for the following reasons :
A state transition graph describes a local dynamic constraints at one object (principle of localization) and it provides a simple and abstract view of behavior of object. It facilitates the process of specification, completeness and validation of dynamic constraints OOD [Booch91], OMT [Rumbaugh&al.91].
States determine sets of legal actions on objects. They can be represented by attributes. State transitions are caused by actions on objects. Actions are caused by events. An action induces the transition of an object from one consistent state to another consistent state. A state transition may be represented by a function:

$$\delta : S \times \Sigma \rightarrow S$$

Where S is a set of finite states and Σ is a set of finite operations.
The function of transition indicates, when the object is in a certain state and that an operation is invoked. The nodes of the graph are the states, the arcs and the labels are fixed by the function of transition.
For instance, in order to represent the case of an electric bulb, the operations are the moves of the interruptor. There are given as follows :

S={lighted, extinct}
OP={invoke the interruptor, release the interruptor}
s() = lighted
δ (extinct, invoke the interruptor) = lighted
δ (extinct, release the interruptor) = extinct
δ (lighted,release the interruptor) = extinct
δ (lighted, invoke the interruptor) = lighted

This example may be represented by state transition diagrams as following :



2.3- Scheme representation:

A class type provides an unambiguous description of the structural and behavioral characteristics that are common to a population of objects.
Each a class type has one or more properties p_i, which form a set P ($p_i \in P$). The value of a property remains unchanged during the life cycle of the object occurrence.
The state of object occurrence is represented by an attribute state. The different state values st_i of a class type form a set St ($st_i \in St$).
Each class type has one or more constraints cn_i, which form a set Cn ($cn_i \in Cn$).
Each class type has one or more operations op_i, which form a set OP ($opi \in A$). Each action has a valuation, i.e one or more rules stating how attribute values are changed or computed by the execution of an operation.
Each class type has on or more events evi, which form a set Evt (evi ∈ Evt).
A class type can be described as a tuple <P, St, Cn, OP, Evt>.
We give here the definition of the textual specification of the base class type. An extended Backus-Naur Form is used to specify the syntax to be derived from the conceptual framework :

Class <class-name> [<Type>]
[Inherits from {< superclass-name>}+]
[Properties
{<attribute-name>: <attribute-type>}+]
[States
{<state>}+]
[Associations
{<association>}+]
[Constraints
{<static constraint>}+]
[Operations
{<operation>}+]
[Events
{<event>}+]
End --- class

With:
<Type> :: The generic parameter
<attribute-type> :: <basic_domain>| <collection_domain> | <aggregate_domain> | <enumerated_domain> | <domaine-intervale>
<basic_domain> :: integer | real | date |string | boolean ...
<collection_domain> :: SET OF (<class-name>)
<enumerated_domain> :: ENUMERATED ({value}+)
<interval_domain> :: [min..max]
<aggregate_domain> :: AGGREGATE OF (<aggregate-name-class>)
<association> :: ASSOCIATION <association-name> OF <class-1-name> [Cardmin,Cardmax], <class-2-name> [Cardmin,Cardmax]
<state> :: δ: S x Σ -> S
<static constraints> :: <attribute constraints> | <inheritance constraints> | <uniquenes constraint>
<attribute constraints> :: <expression>
<expression> :: <simple_expression> | <composed_expression>
<simple_expression> :: <term>
<comparaison_operator> <term> | <term>
<term> :: <attribute-name> | OLD.<attribute-name> | NEW.<attribute-name> <method-name> | <constant-name>
<composed_expression> :: <simple_expression> <logical_operator> <expression>
<predicate> :: <expression>
<operator> :: <logical_operator> | <set_operator>
<logical_operator> :: OR | NOT | AND
<set_operator> :: IN | UNION | INTERSECT
<comparaison_operator> :: = | > | < | ≥ | ≤ | ≠
<inheritance constraints> :: <disjunction> | <covering>
<disjunction> :: <S>$_i \cap$ <S>$_j = \emptyset$, $\forall i \neq j$
<covering> :: $\dot{\bigcup}$ <S>$_i$ = <G>
<uniqueness constraint> :: UNIQUE {<attribute>,}+
<operation> :: <operation-name> <Boby of operation> [<Type of Operation>]
<Type of Operation> :: PUBLIC | PRIVATE | PROTECT
<event> :: <event name> <event type> [<predicate> | <message>] <trigger>
<message> :: {<paramaters>}+
<trigger> :: <operation name> ON <class name> [{<Facteur>}+] [{<Condition>}+]

The following figure presents a synoptic of the use of different concepts in several representative methods of Object-oriented analysis and design. Some methods focus on the static's characteristics (OOSA, OOA, OMT, MCO), others on dynamic ones (OOD). Our model, OOCPM, takes into account the two aspects (static and dynamic).

Object-Oriented Concepts	OOA	OOAD	OMT	OOD	O*	MCO	OOCPM
Class	●	●	●	●	●	●	●
Object	●	●	●	●	●	●	●
Object Attributes	●	●	●	●	●	●	●
Aggregation	●		●	●	●	●	●
Association	●	●	●	●		●	●
Services ("client/supplier")	●	●		●		●	●
Genericity				●			●
Inheritance (by specialization)	●	●	●	●	●	●	●
Communication between objects	●	●	●	●	●	●	●
Object behavior (methods, Actions)	●	●	●	●	●	●	●
Events		●	●		●		●
States	●	●	●	●	●	●	●
Constraints on structure	●	●	●	●	●	●	●
Constraints on dynamic behavior		●	●	●	●	●	●
Control - Timing	●		●	●	●	●	●

Fig 3: A comparative Object-oriented methods

3- Mapping of OOCPM

3.1- General Overview

OOCPM aims at assuring a mapping, guided by a software tool, from any conceptual specification towards an implementation. The interface consists of an object-oriented pivot model, Rc rules for the mapping from the conceptual model to pivot model and Ri rules from the pivot model to object-oriented implementation (language and persistence).

The figure below illustrates the pivot model:

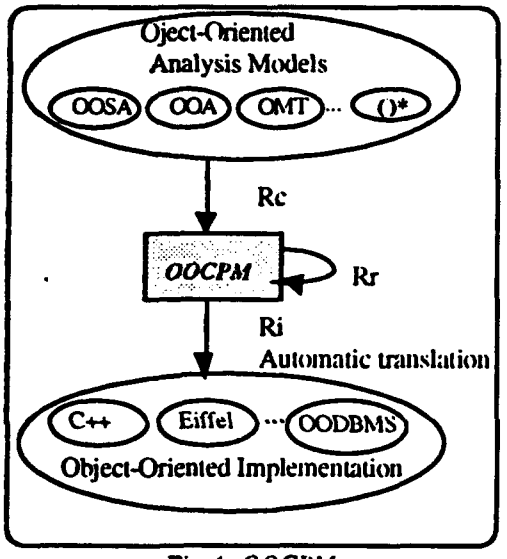


Fig 4: OOCPM

Where Rc rules are used to transform the user's conceptual schema into pivot model. Ri rules are applied to lead to an object-oriented programming environment and Rr rules are used for refinement.

3.2- The Mapping rules between the Object-oriented analysis models and OOCPM

The conversion of a given conceptual schema (a result from a certain modelling) to a standard software architecture with the help of the concepts of our pivot model is carried out in the following stages:

Firstly, the conceptual schema is transformed into an OOCPM by the application of the very precise transformation rules that assure the tracability between a given modelling and a standard generic design.
Secondly the results of this transformation, by applying a set of rules are transformed into a language or an environment (design product) that will be ready to be implemented.

3.2.1- Transformation rules of the conceptual schema

The transformation of a conceptual schema (i.e. the product of modelling) into elements of the OOCPM is presented below. This study is based on the 6 well-know conceptual models (OMT, OOA, OOD, OOAD, MCO, O*):
Note also that the conversion into the conceptual pivot model may be as follows :

- mapping simple (1-1); it is represented by a function: $m(C_S) = C_P$
- grouping (n-1); it is represented by a function: $g|(C_S)| = C_P$
- retyping (1-1); it is represented by a function: $r(C_S) = C_P$
- explosing (1-n); it is represented by a function: $e(C_S) = |C_P|$

where Cs is a source concept and Cp is a pivot concept

To transform the user's object-oriented conceptual schema, the following Rc rules must be used:

Rc_Class: Each source entity or source class is transformed into a pivot class in which the visible attributes are the properties which characterize the corresponding class.
For instance,

m(O* class) = Pivot_class
m(MCO class) = Pivot_class
m(OMT class) = Pivot_class
m(OOA class) = Pivot_class

Rc_Abst_Class: Each source abstract class or actor class is transformed into a pivot abstract class (a class without instance variable).
For instance,

m(O* actor class) = Pivot_abstract_class
m(MCO abstract class) = Pivot_abstract_class

Rc_Attributes: Each attribute of type property is translated into attribute property pivot according to their types (predefined, LIST OF, SET OF, ENUMERATED, AGGREGATE OF ...).

Rc_Att_Hist: Each variable attribute, that can be historised, requires to memorize all the state changes, is transformed into aggregate attribute in which is specified the temporal aspect (date, hour).

r(Attribute_Hist) = Aggregate_Att

For instance, To keep all the client addresses, during all their lives:

Class client
Properties
address: *AGGREGATE OF* (ad: *AGGREGATE OF* (street: string, number: integer, city: string, country: string), d: DATE)

Rc_Stat_Const: All source concepts of type static constraints are translated into a Static Constraints pivot according to their nature (uniqueness or attribute).

m(O* Uniqueness Constraint) = Pivot Uniqueness Constraint

m(O* Attribute Constraint) = Pivot Attribute Constraint

Rc_Ass_Class: The notation of an associative class type was adopted from OMT [Rambaugh&al91] OOSA [Shlaer&Mellor91], OORM[Hwang90], EROOS [Baelen92]. An associative object may have attributes, life cycle and a role. The concept of associative entity corresponds to composite entity as it was introduced by Chen [Chen85] and used by the OMT and OOSA by using the concept associative class.
This type of class will be transformed into a pivot class by specifying the links of association with the test of the classes implied.
Every source associative class (for example: OMT, OOSA) is transformed into a pivot class in which the attributes are those from the associative class.
For instance,

r(OOSA associative) = Pivot class

r(OMT associative) = Pivot class

Rc_Ass_RelationShip: Every source relationship of type (n-m), which is accompanied by properties is transformed into a pivot class. The attributes of this class are the properties which characterize the association type. The identifiers of the classes or entities implied in the association are added into the properties of the pivot class.
For instance,

m(OOSA association) = Pivot association type

m(Henderson_Sellers association) = Pivot association type

m(OMT association) = Pivot association type

Rc_Agg_Link: Every source aggregation link is transformed into a pivot composition. This link will be simple or multiple.
For instance,

r(OMT aggregation) = Pivot composition link

r(OOA aggregation) = Pivot composition link

r(BON aggregation) = Pivot composition link

Rc_Comp_Type: Every source association link in which $c_2 = (1,1)$ is transformed into a pivot composition link according to the cardinality c_1:

if $c_1 \in \{(0,1), (1,1)\}$ => Simple

if $c_1 \in \{(0,N), (1,N)\}$ => Multiple

Rc_Sem_Link: The rest of each source semantic link is transformed into a pivot association link by specifying the characteristics between the implied classes. For instance,

r(OOD Utilization link) = Pivot association link

r(OOSA Utilization link) = Pivot association link

r(BON Utilization link) = Pivot association link

Rc_Inheritance_Link: Every simple or multiple source inheritance link is transformed into a simple or multiple pivot inheritance link (i.e. inheritance by specialization) by specifying the constraints to restrict the possibilities of existence of the objects of several specialized classes, for each object of a generalized class, if necessary. For instance,

r(OMT Inheritance link) = Pivot inheritance link (with a disjunction constraint)

r(OOSA Inheritance link) = Pivot inheritance link (with a disjunction constraint)

m(O* Inheritance link (disjunction constraint)) = Pivot inheritance link (with a disjunction constraint)

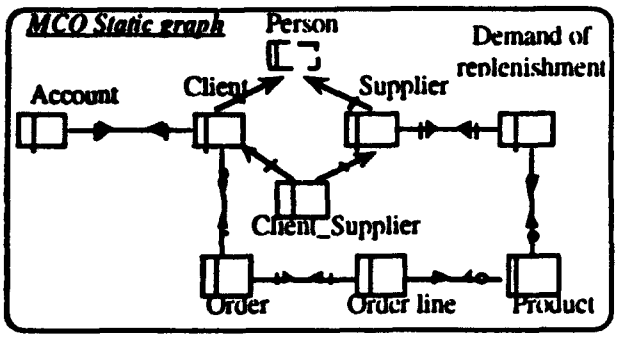
m(O* Inheritance link (covering constraint)) = Pivot inheritance link (with a covering constraint)

m(O* Inheritance link (partition constraint)) = Pivot inheritance link (with a disjunction and covering constraints)

Rc_Genericity: Every source genericity concept is transformed into a genericity pivot by specifying a type. For instance,

m(OOD genericity) = Pivot genericity link

A same example is described here after using the two source models, O* and MCO and it's translated in the OOCPM.
In this O* example, we note a covering constraint, each person must be a client or a supplier (or both). In the MCO example, this constraint is represented by an abstract class Person (non instanciable) whereas Client and Supplier are persistent classes. Lower level classes may be created to represent clients who are suppliers at the same time. Discussion about creation of such classes can be found in [Castellani93].
To represent static links between objects, in O* one way arrow is used where as in MCO double way is required.



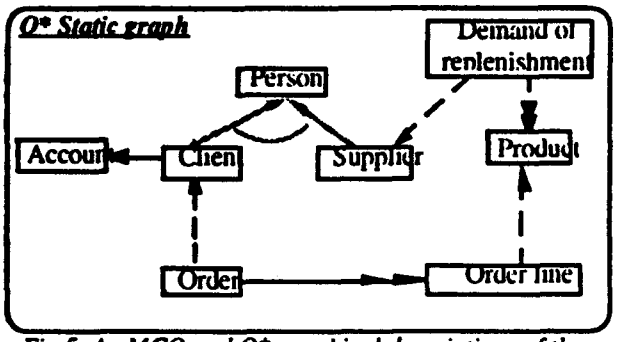


Fig 5: An MCO and O graphical descriptions of the static relationships between classes (Source Models)*

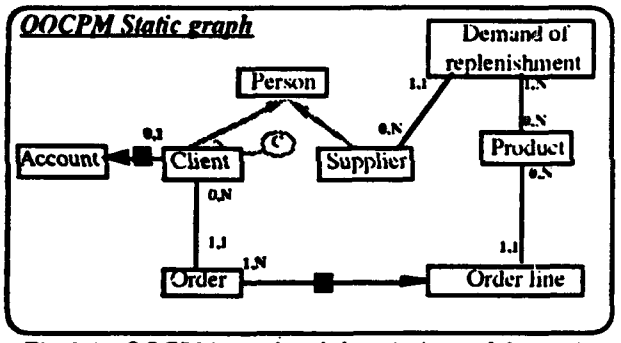


Fig 6:An OOCPM graphical descriptions of the static relationships between classes (Pivot Model):

The dynamic concepts (operation, event, state transition graph, service, actor ...) are mapped using the following rules :

Rc_Operation_Action: All source concepts of type operation, action or service are translated into an operation pivot.

Some models describe operations by a text in natural language (example O*, MODWAY [Cauvet93]. This text specifies the operation purpose and the rules according to which attributes and states are valued or changed. In this case, the designer has to give his algorithm details using, when needed, the classical instructions (IF...THEN...ELSE...ENDIF, WHILE....END,...). If the model uses a formal specification language, the translation will be automatically done into a pivot model specification language. For instance,

r(O* operation) = Pivot Operation (with a specification of the body of operation)

m(MCO Service) = Pivot Operation

For instance, an order creation operation is translated into a OOCPM as follows:

```
Class ORDER
Properties
    creation-date: DATE
    Delivery_date: DATE
    Invoiced_date: DATE
Sates
    (created, delivered, invoiced)
    δ(S0, Create_Order)=created
    δ(created, Delivery)= delivered
    δ(delivered, invoicing)= invoiced
Associations
    ASSOCIATION Ord_line OF Order [1,N],
Order_Line [0,N]
    ASSOCIATION Cli_Ord OF Order [1,1] , Client
[0,N]
Constraints
    creation_date ≤ delivery_date
    delivery_date ≤ invoice_date
Operation
  Create_Order: PRIVATE
          (Var: number, ord_line, create_date,...)
          Precondition: absent order
          Body
             [Create_Order_Line()]+
             --- create one instance of order
          EndBody
          Postcondition: state = 'created'
End
```

Fig 7: An example of operations

Rc_Event: Each source event type is transformed into an pivot event depending on its type (internal, external or temporal). For instance,
r(OMT Event) = Pivot Event (According to types)

Rc_Int_Event: Each source internal event is transformed into an internal encapsulated event in the corresponding class ascertained. For instance,
m(O* Internal Event) = Pivot Internal Event

Rc_Ext_Event: Each source external event is transformed into an external pivot event. The attributes of this source event type are constitute the corresponding. external message. For instance,
m(O* External Event) = Pivot External Event
r(MCO Event Model Object) = Pivot External Event

Rc_Temp_Event: Each source temporal event is transformed into a temporal pivot event. For instance,
m(O* Temporal Event) = Pivot Temporal Event

For example the product event (out of stock) can be translated into a OOCPM as follows :

```
Class PRODUCT
...
Operations
...
Events
 out of stock: internal
  predicate
    (OLD.qte_stock ≥ replenishment_level) and
NEW.qte_stock < replenishment_level)
  triggers
    create ON Supplier, Order
...
```

Fig8: An example of the textual description of the Out_Stock

3.3- The Mapping rules between OOCPM and target languages

The second set of rules is used for the translation from an OOCPM specification, already established before, towards a target object-oriented implementation.

Ri_Class: Each pivot class is translated into a class within the target language.

Ri_Abst_Class: Each pivot abstract class is translated into an abstract class or deferred class (a class without instance variable in OOPL).

Ri_Inherit: The *'Inherits from'* concept is translated into a classical inheritance into target languages. To resolve the multiple inheritance conflict, REDEFINE and RENAME can be used in the target language.

Ri_Basic_Dom: All object-oriented programming languages support the <basic_domain> notion.

Ri_Collection_Dom: Each <collection_domain> is translated using the generic class COLLECTION [X] or into a collection SET (X), where X is a type.

Ri_Aggegate_Dom: Each <aggregate_domain> is translated using an abstract class into target language.

Ri_Enum_Int_Dom: Each <enumerated_domain> or <interval _domain> is translated using the method or routine into target language.

To control state transitions starting from the same state and for the same event is by specifying a precondition. It is possible that an object remains in the same state as it was before. To avoid redundancy, conditions are given as static (attribute, state, occurrence, etc.) constraints if possible, if it is not possible to specify constraints for state transition as static constraints, a precondition may be specified.

Ri_State_Trans: All state concepts are translated with enumerated attribute (called STATE) and with method or routine (precondition and postcondition are mandatories).

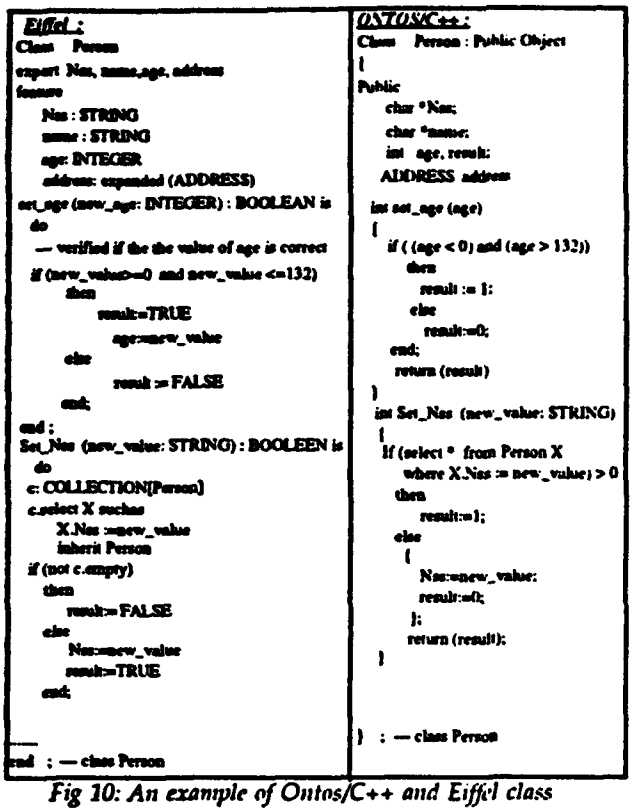
Ri_Static_Const: Each uniqueness or attribute constraint is translated into an invariant or using a specific method.

Ri_Static_Link: All concepts of type composition or association link are translated with aggregated attributes and cardinality constraints in the constraint part. In the case of a strongly dependency, cardinalities must be defined in two classes, the caller and called. However, in the case of a weak dependency, the cardinalities are expressed only in the caller objects. Each cardinality constraint is translated, into target language, within a specific method which verifies the minimal and maximal cardinalities (in the caller and the called)

For instance, the translation of the static aspect of the pivot conceptual specification, given in Figure 9 and 10 , towards Eiffel and ONTOS/C++ programming environments is as follows:

```
Class PERSON
Properties
   Nss :string(15)
   Name: string(30)
   Age: [0..99]
   Address: AGGREGATE OF(num:integer, street:
string(20), city: string(25), country: string (16))
Constraints
   UNIQUE (Nss)
   Client ∩Supplier = Person
....
END --- Person
```

Fig 9:An example of OOCPM textual specification



```
Eiffel :
Class   Person
export Nss, name,age, address
feature
    Nss : STRING
    name : STRING
    age: INTEGER
    address: expanded (ADDRESS)
set_age (new_age: INTEGER) : BOOLEAN is
  do
   -- verified if the the value of age is correct
  if (new_value>=0 and new_value <=132)
        then
            result:=TRUE
              age:=new_value
        else
              result := FALSE
        end;
end ;
Set_Nss (new_value: STRING) : BOOLEEN is
   do
  c: COLLECTION[Person]
  c.select X suchas
       X.Nss :=new_value
       inherit Person
  if (not c.empty)
     then
        result:= FALSE
     else
        Nss:=new_value
        result:=TRUE
     end;


end ; -- class Person
```

```
ONTOS/C++ :
Class   Person : Public Object
{
Public
    char *Nss;
    char *name;
    int  age, result;
    ADDRESS address
int set_age (age)
{
   if ( (age < 0) and (age > 132))
       then
          result := 1;
       else
          result:=0;
   end;
    return (result)
}
int Set_Nss (new_value: STRING)
{
 If (select * from Person X
     where X.Nss := new_value) > 0
   then
       result:=1;
   else
    {
       Nss:=new_value;
       result:=0;
    };
    return (result);
}


} ; -- class Person
```

Fig 10: An example of Ontos/C++ and Eiffel class

where Object is a predefined ONTOS class. Each persistent object must be an instance of the *Object class* or its derived classes.

The inheritance constraints are mapped in object-oriented programming using the following set of rules:

Ri_Inherit_Disj: Each inheritance constraint such that $S>_i \cap <S>_k = \emptyset$, $\forall\ i \neq k$ is translated by a classical inheritance into the target languages (all classes are persistents).

Ri_Inherit_Cov: Each inheritance constraint such that $\overset{n}{\underset{i=1}{\cup}} <S>_i = <G>$ is translated by a abstract superclass and the set of subclasses all persistents. $2^n - (n+1)$ persistent classes will be created. They represent all possible intersections between the n subclasses. To resolve the multiple inheritance conflict, in the object-oriented language, these classes will be virtual in the C++ languages.

Dynamic aspects of an object are operations and events. Operations are represented by methods in object-oriented programming. Operations may retrieve, change, or delete values of attributes, change the state of an object, set up or terminate a relationships with other objects. The OOCPM operation concept is mapped as follows :

Ri_Operation: Each instance operation is translated by a specific method into the target language. Pre and Post conditions are checked into the method body.

The meaning of the event concept is not the same in different conceptual models. This concept is particularly hard to implement in object oriented languages, because of the functional principles of the method calls [Kraïem92]. For instance, in O*, it poses some problems such as:

- the implementation of the internal event mechanism
- the management of the dynamic transition during the execution
- the saving of the event succession.

To resolve these problems, we propose a solution based on two steps:

- when event is activated, its operations are triggered and its predicates - susceptible to be chained - are tested
- then, each event having a true predicate is activated in sequence.

To implement this mechanism, we use the following rules :

Ri_Int_Event: An internal event is translated by a private method for the object.

Ri_Ext_Temp_Event: Each external or temporal event is translated by an abstract class and a method for its execution.

Ri_Event_Method: Every event method is implemented by a routine in the target language.

Ri_Event_Pred: For every private event method, a specific method (TEST_PRED) is implemented in order to test predicate. A Boolean parameter is used when calling the method. When the call has a factor, we must keep the predicate value for each affected object.

```
Class Product
 feature
 Pred_Out_of_stock: BOOLEAN;
 sp: Supplier;

  Out_Of_Stock (Pred_Out_of_stock): BOOLEAN is
  do
   if (OLD.qte_stock >=replenishment_level) and
     (NEW.qte_stock < replenishment_level)
   then
     Demand_Of_Replenishment(Pred_Out_of_stock);
   end;
  end;
.....
end - - - class Product

Class Supplier
 export Demand_Of_Replenishment ...;
 feature
 ------
 Demand_Of_Replenishment(Pred_Out_of_stock): BOOLEAN is
 do
  -- operation performing text
 end;
end; - - - class Supplier
```

Fig 11: Eiffel implementation of an pivot event.

A detailed algorithm is presented in the [Kraïem92,93] for the mapping of events into programming languages.

4- Conclusion

OOCPM solves some of the limitations of the existing models. It is a generic conceptual model combining together the static and dynamic aspects. Coupling this models with an object-oriented environments allows to extenuate some of their limitations.

We introduced rules that allow the formulation, at an in high of abstraction of program architecture according to principles of object-oriented analysis, design and programming.

For our future works, we address the problem of reusing generic pieces of the Requirements Engineering Process which are called Process Chunks . This notion will be introduced in our framework.

A process chunck is a piece of generic knowledge reusable for requirements engineering issues of the same kind. It is in the form <situation, decision, argument, action>.

Acknowledgments

We thank Professor Colette ROLLAND, Professor Faouzi BOUFARES for their helpful advise and remarks and their amiability.

References

[Baelen92], S.V.Baelen, J.Lewi, E.Steegmans, H.Van Riel, "EROOS: an Entity-Realtionship base Object-Oriented Specification Method", TOOLS'7, 1992.

[Bailin89].Bailin, S.C., "An object-oriented requirements specification method", Communications of the ACM, Vol. 32, N°. 5, 1989.

[BC92] BUSINESS CLASS project, "Analyst Workbench Tutorial" Release 2, Deliverable BC.R.TS.T34, Business class ESPRIT II P 5311, 1992.

[Booch86] G.Booch, "Object-Oriented Development", IEEE Trans. on S.E., Vol. SE-12, N°2, Feb. 1986.

[Booch87] G.Booch, *Software Engineering with Ada*, 2nd Edition, Benjamin/Cummings Publishing Co., Menlo Park, 1987.

[Booch91], G.Booch, "Object Oriented Design With Applications", Benjamin Cumming Ed., 1991.

[Brunet90] J. Brunet, C.Cauvet, L. Lassoudris, "Why Using Event in a Hignt Level Specification", in Proc. of the Entity / Relationship Conference, Lausanne, 1990.

[Brunet91] J. Brunet, "Modeling the world with Semantic Objects", Working conference on the object oriented approach in information systems, 1991.

[Castellani93] X. Castellani, "MCO Méthodologie Générale d'analyse et de Conception des Systèmes d'Objets, Tome 1: l'Ingénieurie des besions" Edition Masson, Paris 1993.

[Cauvet93], C. Cauvet, F. Semak, D. Meddahi, J. Brunet, "MODWAY: Object-Oriented analysis method", Proc of CAiSE'93 Conf., Paris 1993.

[Chen76], P.P. Chen, "The entity relationshipmodel: towards a unified view of data", ACM TODS, vol. 1, n°1, 19976.

[Coad & Yourdan91] E. Yourdon, P. Coad "Object oriented analysis", second Edition, Yourdon press, 1991.

[Coad90], Coad, P. and Yourdon, E. "Object-Oriented Analysis", Prentice-Hall, Englewood Cliffs, NJ, 1990.

[Cointe87] P Cointe, "Metaclasses are First Class: the ObjVlisp Model",LITP, Université Paris VI ACM, OOPSLA, 1987.

[DeAntonellis81], J. DeAntonellis, B. Zonta, "Modelling Events in Data Base Applications Design", Proc. of the 7th Int. Conf. on VLDB, Cannes, 1981.

[Elmasri90], R. Elmasri, I. El_Assal, V. Kouramajian, "Semantics of temporal data in an extended ER model", 9th Int. Conf. on the Entity-Relationship Approach, Lausanne, Switzerland, Oct. 1990.

[Heitz89] M. Heitz, "HOOD, une méthode de conception hiérarchisée orientée objet pour le développement des gros logiciels techniques et temps réel", Déc. 1989, Journées Ada-France.

[Henders-Sellers 90] Henders-Sellers, B. and Edwards, J.M., "The object-oriented systems life cycle" Communications of the ACM, Vol.33,N°9, 1990.

[Henders-Sellers 91], Henders-Sellers, B. "Analysis and Design, Methodologies and Notation", Tutorial, TOOLS, Paris, 1991.

[Hwang90], S. Hwang, S. Lee, "An Object-Oriented Approach to Modelling Realtionships and Constraints based on Abstraction Concept", Int. Conference of Database and Expert Systems Applications, Vienna, Autsria, Aug. 1990.

[Iivari91], Iivari, J, "Object-Oriented Design of Information Systems: The design process", Proceedings of the Object-Oriented Appoch in Information Systems Conference, Elsvier Science Publishers B.V (North-Holland), 1991 IFIP.

[Jungclaus91], R. Jungclaus, G. Saake, T. Hartman, C. Sernadas, "Object-Oriented Specification of Information Systems: The TROLL Language", Technische Universität Braunschweig, Dec. 1991.

[Kraïem92] N.Kraïem, J.Brunet " Mapping of Conceptual Specifications into Object-oriented Programs", SEKE'92, Proceedings of the Fourth International Conference on Software Engineering and Knowledge Engineering, IEEE, Capri, June 1992.

[Kraïem93], N. Kraïem, F. Gargouri, F. Boufares, "From Object-Oriented Design Towards Object-Oriented Programming", Proc of CaiSE'93, Paris, 1993.

[Manfredi89], F. Manfredi, G. Orlando, P. Tortorici, "An Object-Oriented Approach to the System Analysis", 2nd European Software Engineering Conference, Springer-Verlag, Sept. 1989.

[Meyer90] B. Meyer , 'Conception et programmtion orientée objets", InterEdition, 1990.

[Monarchi92] D.E. Monarchi, G.I. Puhr " A research typology for object-oriented analysis and design", Communications of the ACM, September 1992, Vol 35, N°9

[Pernici90], B. Pernici, "Objects with Roles", ACM/IEEE Conference on Office Information Systems, Boston, MA, April 1990.

[Rambauhg& al91], Rambauhg, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W.,"Object-Oriented Modeling and Design" Prentice-Hall, Englewood Cliffs, NJ, 1991.

[Rolland82], C. Rolland, C. Richard, "The Remora methodology for Information Systems Design and Management", IFIP TC8 Int. Conference on Comparative Review of Information Systems DEsign Methodologies.n North Holland, 1982.

[Rolland88] C. Rolland, O. Foucaut, G. Benci, *Conception de Systèmes d'Information, la méthode Remora*, Ed. Eyrolles, Paris 1988.

[Seidewitz89], Seidewitz, E., "General object-oriented software development: Background and experience", The Journal of Systems and Software, Vol.9, 1989.

[Sernadas89], A. Sernadas, J. Fiadero, C. Sernadas, H.D. Ehrich, "The Basic Building Block Of Information Systems, Information Systems Concept", Noth Holland, Namur, 1989.

[Shlaer&Mellor91] S. Shlaer, S.J. Mellor,"Object Lifecycles: Modelling the World in States", Prentice Hall, 1991.

[Shlaer89], Shlaer, S. and Mellor, S.J., "An Object-Oriented Systems Analysis, Modeling the World in Data" Yourdon Press, Prentice-Hall, Englewood Cliffd, New Jersey, 1988.

[Smith89], Smith, M.K, Hoza, B.J and Tockey, S.R, " An introduction to object-oriented analysis" in Proceedings of the Fifth Structured Techniques Association Conference, Chicago, 1989.

[Teisseire91], M. Teisseire, P. Poncelet, A. Cavarero, S. Miranda, "A-HOOK, The object-oriented analysis of the HOOK system", report of External European Research Project, 1991.

[Wegner89] Peter Wegner. "Concepts and Paradigms of Object-Oriented Programming,"OOPSLA-89 Keynote Talk, 1989.

[Wirfs-Brock90], Wirfs-Brock, R., Wilkerson, B. and Wiener, L., "Designing Object-Oriented Software" Prentice-Hall, Englewood Cliffs, NJ, 1990.

The System Engineering Technology Interface Specification (SETIS): An Update

1993 Complex Systems Engineering Synthesis and Assessment Technology Workshop (CSESAW '93)

July 20–22, 1993
Washington D.C.

Baba Prasad, Moon Lee, Rajesh Puroshothaman, Evan Lock
Computer Command and Control Company
2300 Chestnut, Suite 230
Philadelphia, PA 19103
Tel.: (215) 854–0555, Fax (215) 854–0665

The System Engineering Technology Interface Specification (SETIS): An Update

I. Introduction

The Design Structuring and Allocation Optimization (DeStinAtiOn) project is a research effort sponsored by the Naval Surface Warfare Center (NSWC) that attempts to provide systems engineers with various types of tools and techniques to perform design optimization and tradeoff analysis [HoNH], [NgHo]. A typical scenario that the DeStinAtiOn project hopes to address is that of a systems engineer who wants to change nonfunctional requirements (referred to as System Design Factors [SDF]) such as modifying a constraint in his design and observing how his changes affect reliability. In other words, he wants to perform a tradeoff analysis between the system design factors of performance and reliability [NgHo]. The DeStinAtiOn prototype attempts to arrive at a new methodology in the area of design optimization and tradeoff analysis.

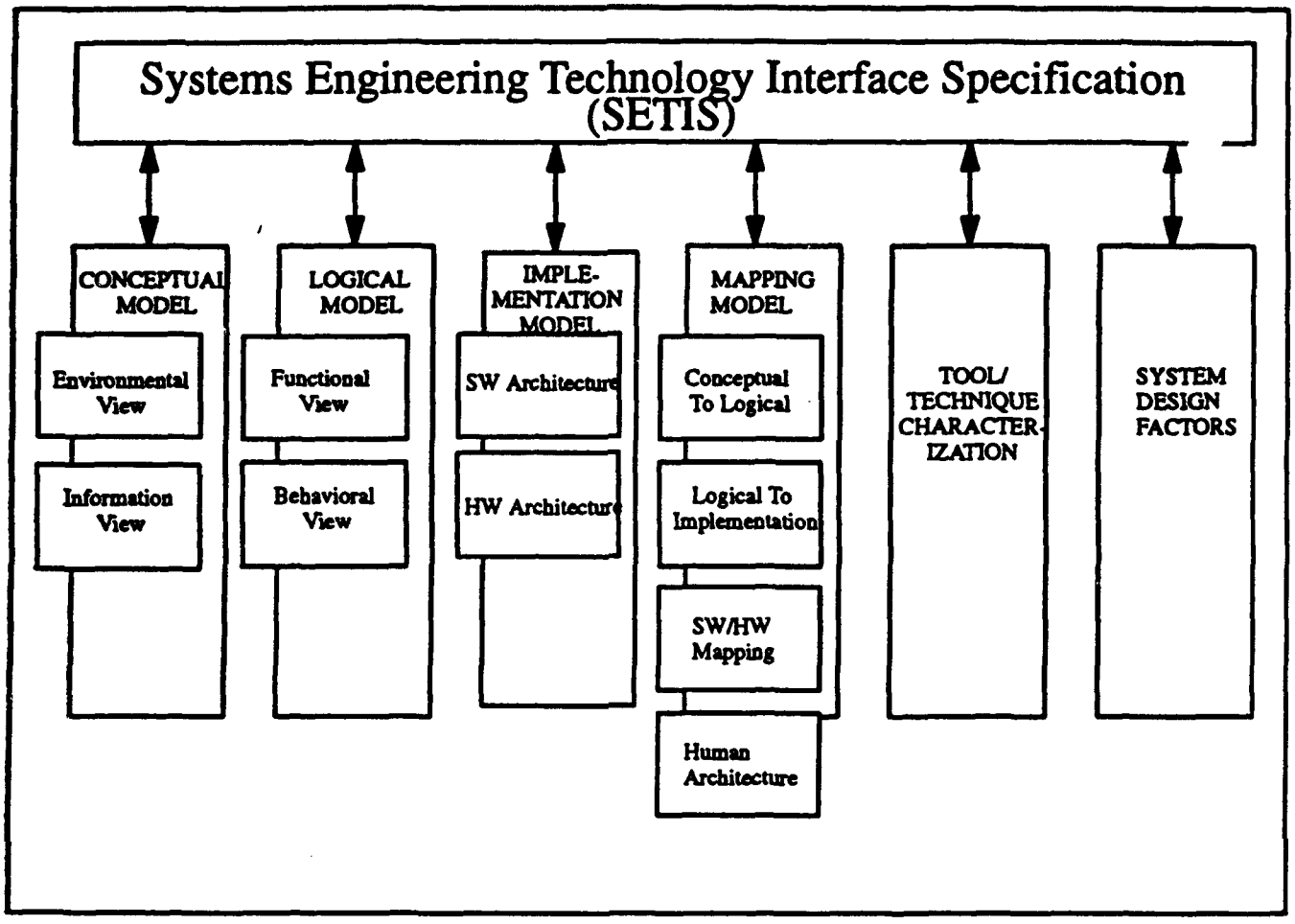
The project attempts to integrate a number of tools and techniques that will facilitate both the design and tradeoff analysis aspects of systems engineering. This immediately necessitates the exchange of information between the different tools being employed. The Systems Engineering Technology Interface Specification (SETIS) is an approach to facilitating information exchange across various tools being employed in DeStinAtiOn in particular and systems engineering in general. SETIS attempts to incorporate similar information exchange standards that are evolving in the CASE industry. The CASE Document Interchange Format (CDIF) is currently the leading standard for the import and export of information between different CASE tools. CDIF is presently oriented primarily toward software engineering. However, SETIS extends the technique to include enhancements for systems engineering information. The intent is to maintain compliance with CDIF. This document will provide an update on the current status of research on SETIS.

II. SETIS: An Overview

The primary goal of SETIS is to provide a standard for information exchange between various tools employed in systems design capture, analysis, and simulation with design optimization tools and techniques. The following components of systems design are examples of the tools and techniques that interface through SETIS:

1. Front–end CASE tools for capturing analysis and design.
2. System behavior modeling tools for analysis and simulation.
3. Optimization algorithms such as scheduling, resource allocation and design structuring .

The various information categories that SETIS covers is illustrated in the following figure:



We will briefly describe the information categories and their implications. (For further information refer to an earlier paper, [LLPL]). The conceptual model captures parameters that arise both from the operational environment and from information models [Kare, Hoan]. Through the conceptual model, the systems engineer and the customer can arrive at an understanding of the system being designed.

The logical model visualizes the system from the perspective of functional and behavioral models without paying particular attention to the implementation methodology. This model thus emphasizes what should be done by the system and not how it should be done. The implementation model, on the other hand, specifically addresses the "how" part of the system. It envisages the various hardware and software components that are required to provide the desired functionality of the system. The mapping model contains pairs that designate how objects are allocated within and across models. Two mappings that are of particular interest to the research are the logical implementation model mapping and the mapping of software onto hardware. Moreover, it also accounts for various human factors that may affect the operation of the system. Objects within any of these models may be attributed by System Design Factors.

As mentioned earlier, SETIS provides guidance and captures information on different types of tools and technqiues. The Tool/Technique Characterization describes the software packages and approaches available (both commerically and within the public domain) that may be applied for capture, analysis, simulation, optimization and assessment, as well as techniques that may be used in the system life cycle, and also justifies their applicability based on the system requirements or specifications.

C++ has been used to describe class hierarchies within these models along with data structures and operations. As an example, operations include procedures for object creation and destruction and for reading and writing to an ASCII file on disk.

III. Use of SETIS with FE–CASE Systems

Several Front–End Computer Aided Software Engineering (FE–CASE) tools are currently popular in the real–time systems community. Cadre's Teamwork, IDE's Software through Pictures (StP), and Mark V Systems' ObjectMaker are ready examples. The systems design engineer may use these FE–CASE tools to specify portions of the system. In particular, the systems engineer follows a methodology and uses the icons that the FE–CASE graphics provide to indicate the various components of the system and their interconnections.

To perform design optimization with DeStinAtiOn the information captured within the FE–CASE tool must be imported into DeStinAtiOn. To accomplish this in a standard way for a diverse set of tools and information, the transfer approach put forth by the Case Document Interchange Format (CDEF) Committee has been used. A picture of the transformations necessary to import the FE–CASE information into SETIS is shown in Figure 2.

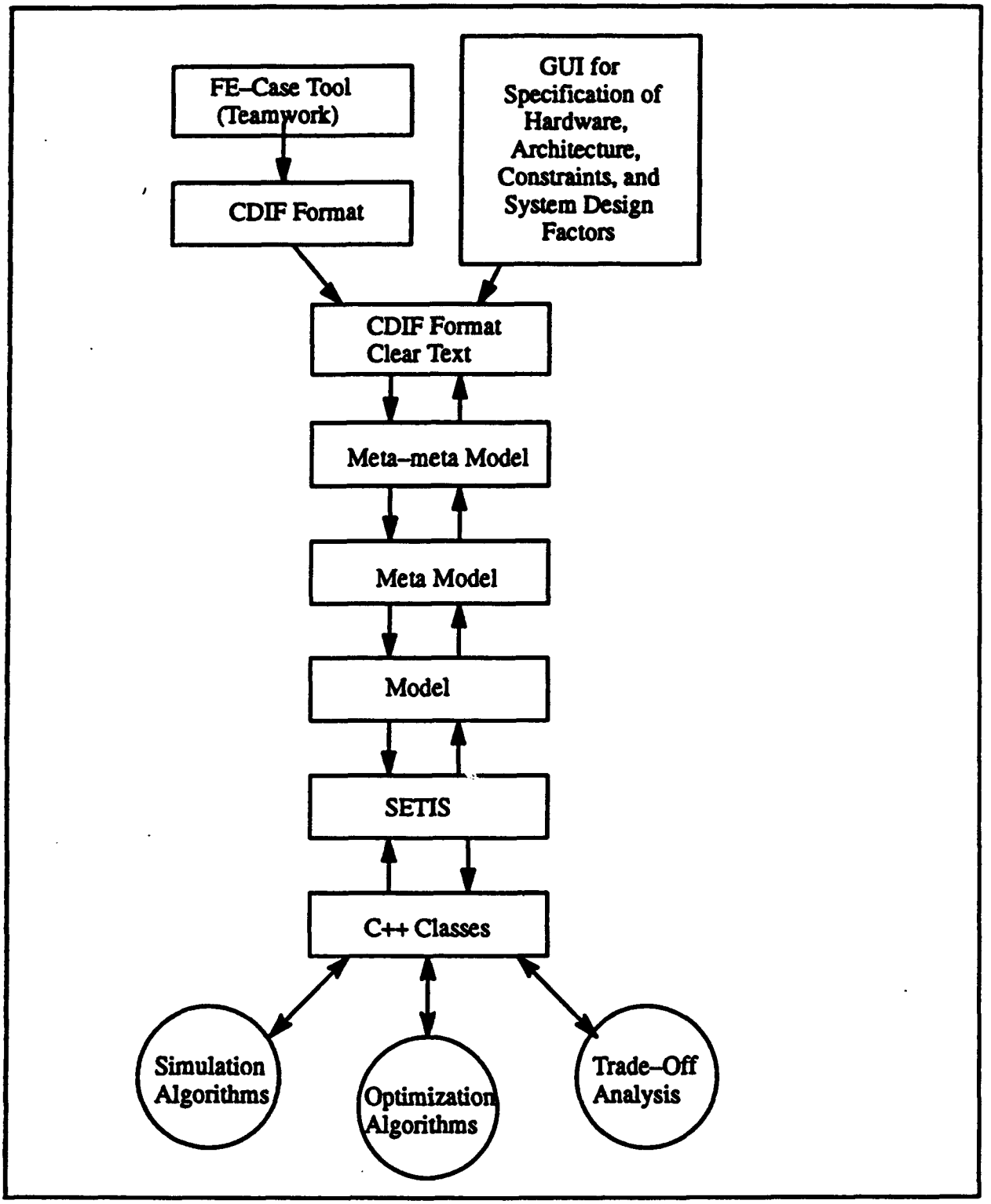


Figure 2. Design Capture Interface.

There is additional information required by optimization techniques beyond what is captured within the FE–CASE tools. Namely, information regarding hardware architecture, constraints (e.g., timing and placement) and Systems Design Factors (SDF). Remember that the SDFs capture the

nonfunctional requirements of a system and are the subject of tradeoff analysis. For demonstration purposes we have constructed separate windows that are independent of the CASE system for entering this additional information. This is shown in the upper right of Figure 2.

SETIS must now integrate the FE–CASE dependent information and the GUI–based information into a tool independent data set that the simulation and optimization algorithms can use. The FE–CASE dependent information can now be put out as a common FE–CASE independent. SETIS will integrate the FE–CASE dependent information and the corresponding FE–CASE independent system information into a single data set (currently envisaged as one file) that complies with CDIF standards.

CDIF envisages the design as moving through several levels of abstraction. The Meta–Meta Model specifies the system at the highest abstraction: there are Meta–Objects with Meta–Attributes and Meta–Entities with Meta–Relationships. At the Meta–Model level, the Meta–Objects and their attributes become more specific although the objects and relationships are still abstract. The system components and their relationships become most clear at the Model level. This hierarchial abstraction is exemplified in the two figures (Figures 4 and 5) shown for the Software Structure. The Software Structure is a component of the Implementation Model within SETIS which was shown in Figure 1 discussed earlier.

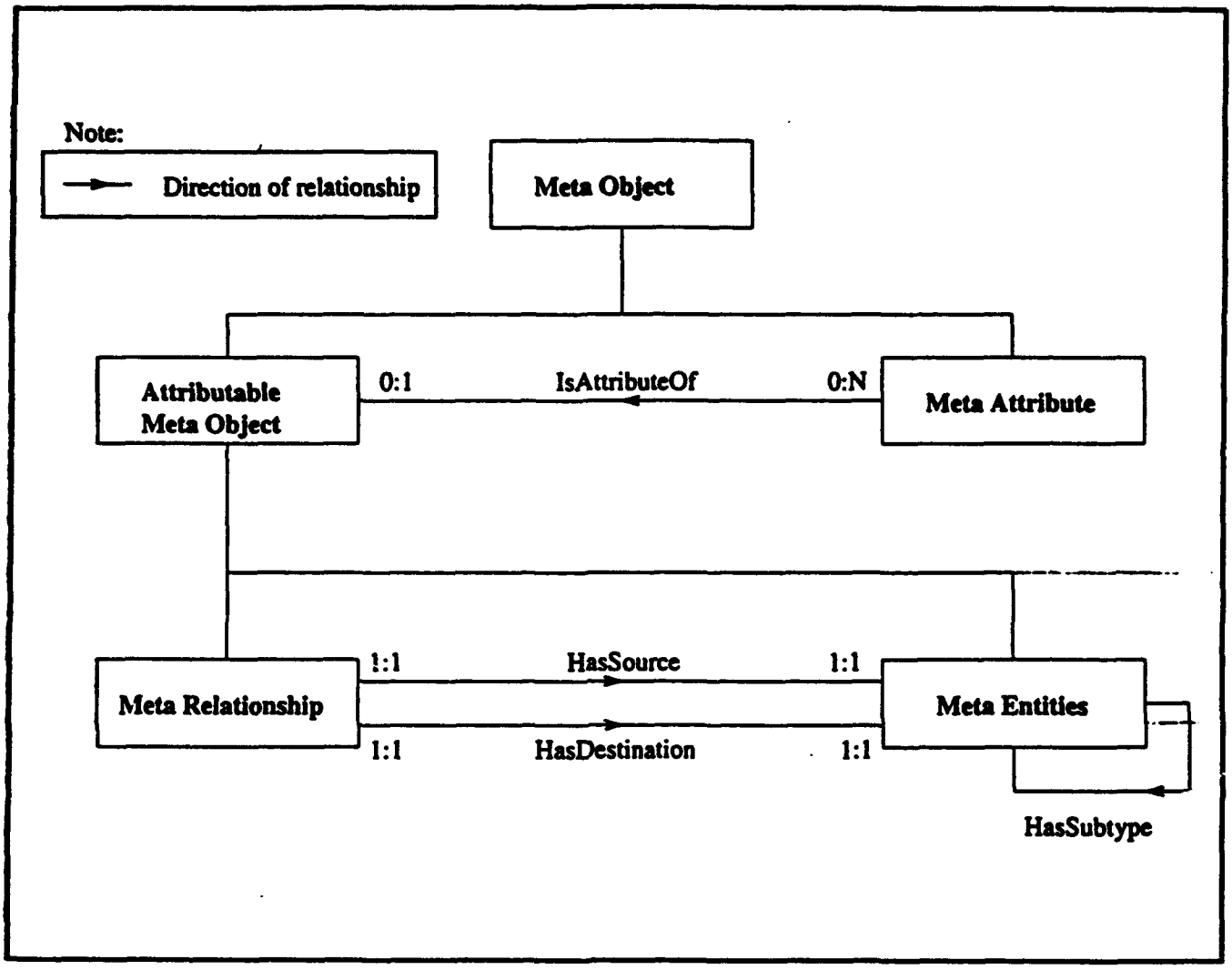


Figure 3. Meta Meta Model

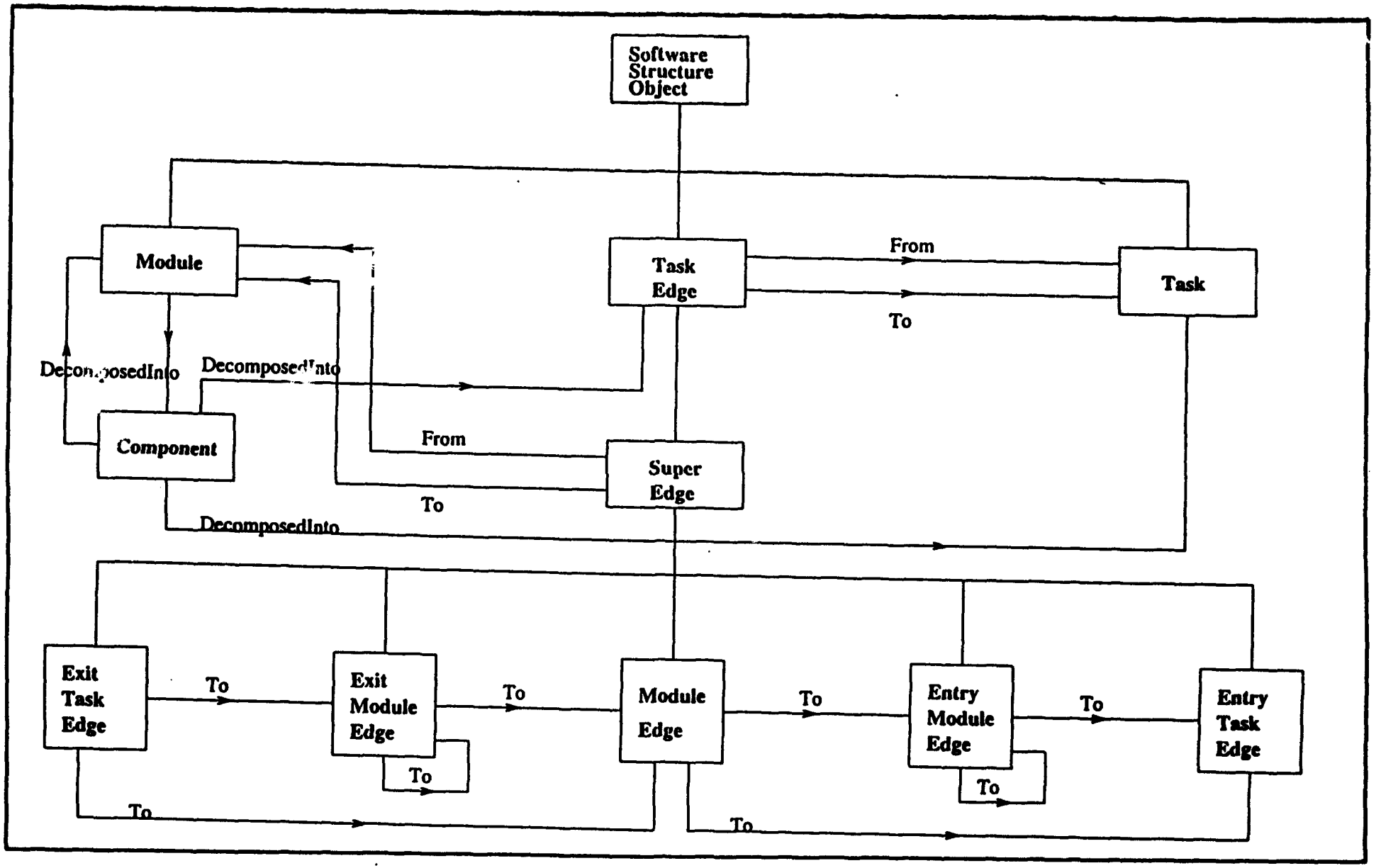


Figure 4. Meta Model for Software Structure.

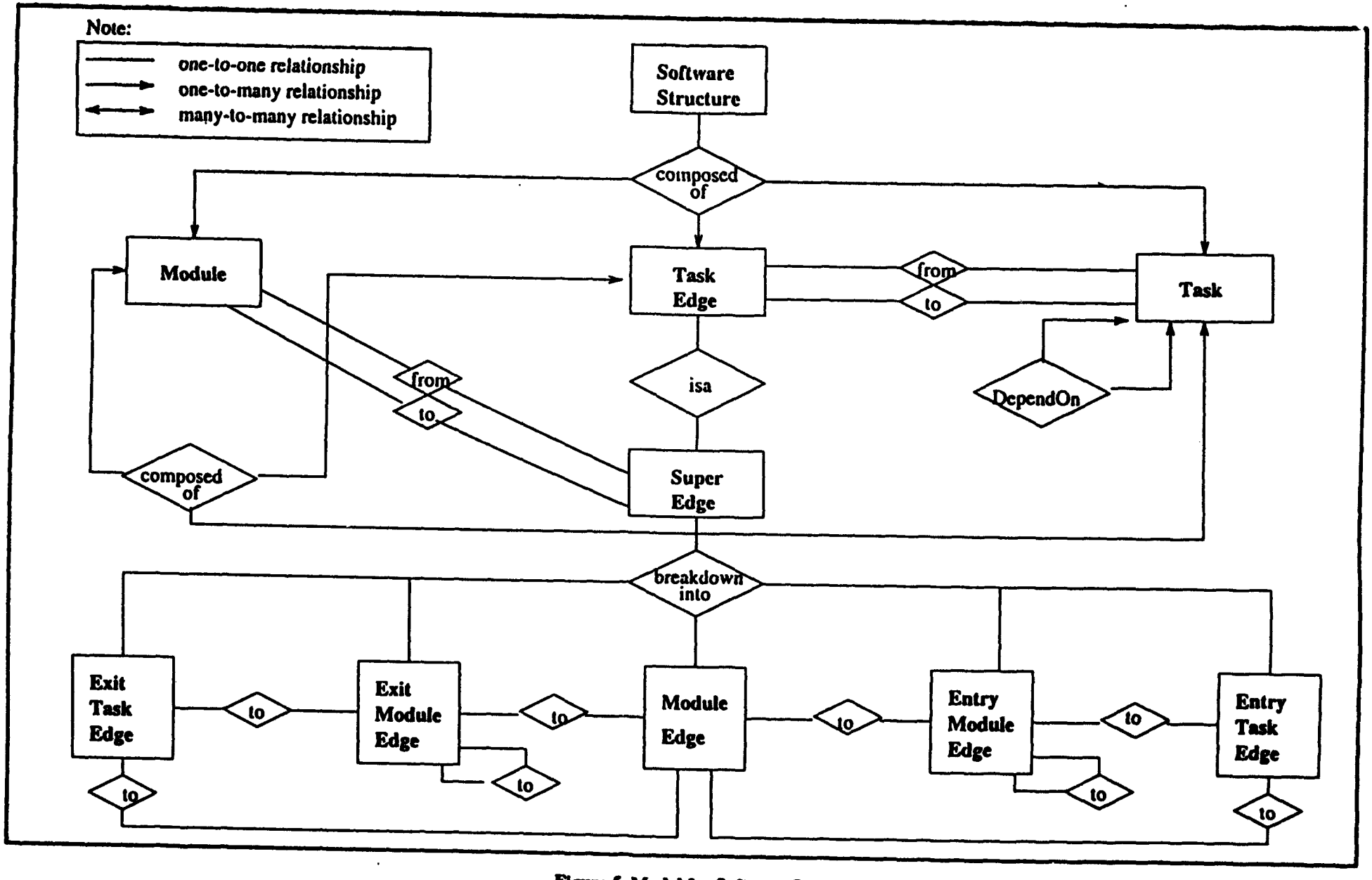


Figure 5. Model for Software Structure.

The design is written out to a file in what is called a "Clear Text" format. This file, which is in ASCII, can be read by any other tool that understands CDIF Clear Text. The model can thus be reconstructed for the tool to operate on. Our current effort has been to construct the various CDIF models for the Implementation Model that was shown in Figure 1.

We have been working on a C++ class library complete with data structures and operations for the model that is discussed in the next section. Users can add their own functions beyond the ones available in the class library to perform optimization or simulation of the design.

III. C++ Class Design for the SETIS Implementation Model

The Implementation Model will be used as an example to illustrate the C++ class library design. In this model, the primary components are the Software Architecture, the Hardware Architecture and the Mapping that relates these two structures (see Figure 1). Figures 6, 7 and 8 show graphics of the current C++ class libraries for each of these components.

Software is visualized as a set of interacting software modules. Each module comprises a number of tasks or submodules. We structure the entire software architecture as a graph where each node is either a module or a task, and each edge represents an interaction (data or control transfer) between the nodes. The edges are further classified depending on whether they connect tasks in different modules or tasks in the same module, and also depending upon the direction of the interaction (entry and exit). Each node and edge has a unique (name, id) pair that identifies the software structure. The "primitive" subclasses have been added to facilitate future additions.

The hardware structure is almost analogous to the software structure; here too, the hardware configuration is visualized as a graph with each host being a node and physical communication links between hosts being edges. The communication aspects of the link, however, neccesitate a slightly different model. There are various communic ıtion subsystems that incorporate intelligent software into their basic hardware to achieve some required functionality. Examples of these are intelligent packet switchers, which are primarily communication oriented hardware devices, but also include packet routing software. Such communication links have unique properties and require a special class called intelligent links. All links can also be classified further depending upon whether they are dedicated point–to–point communication paths between two hosts, or whether they can be used by multiple hosts.

Finally, the mapping structure incorporates the various placement and time constraints and also SDF constraints (e.g. reliability requirements, specified performance requirements, etc.) specified by the system designer, and arrives at a mapping view that determines which software should run on which hardware and at what time.

These are only some of the factors that affected the design of the C++ class hierarchy for the Implementation Model. Figures 6 and 7 provide the class structures in greater detail.

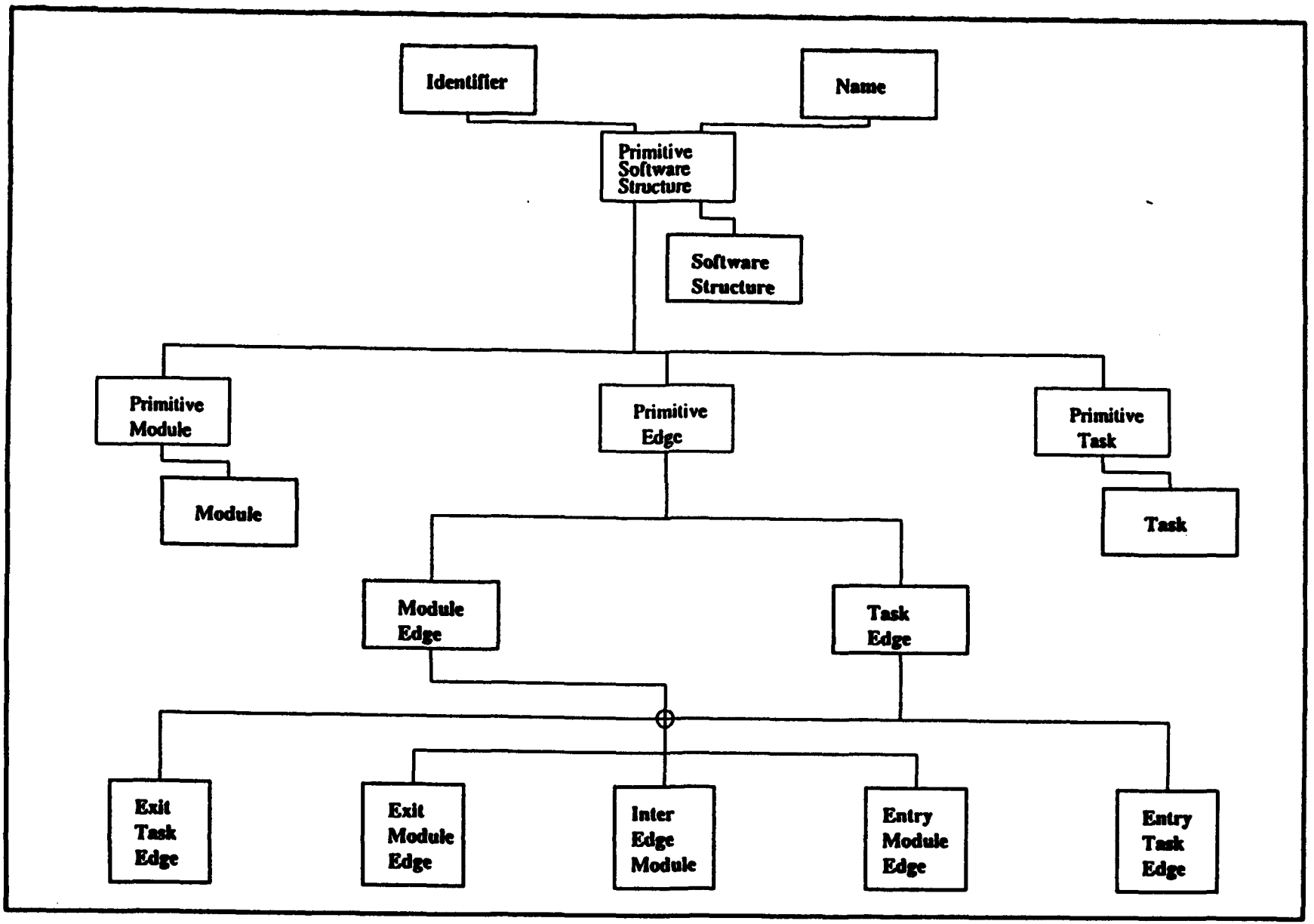


Figure 6. Class Hierarchy for Software Structure.

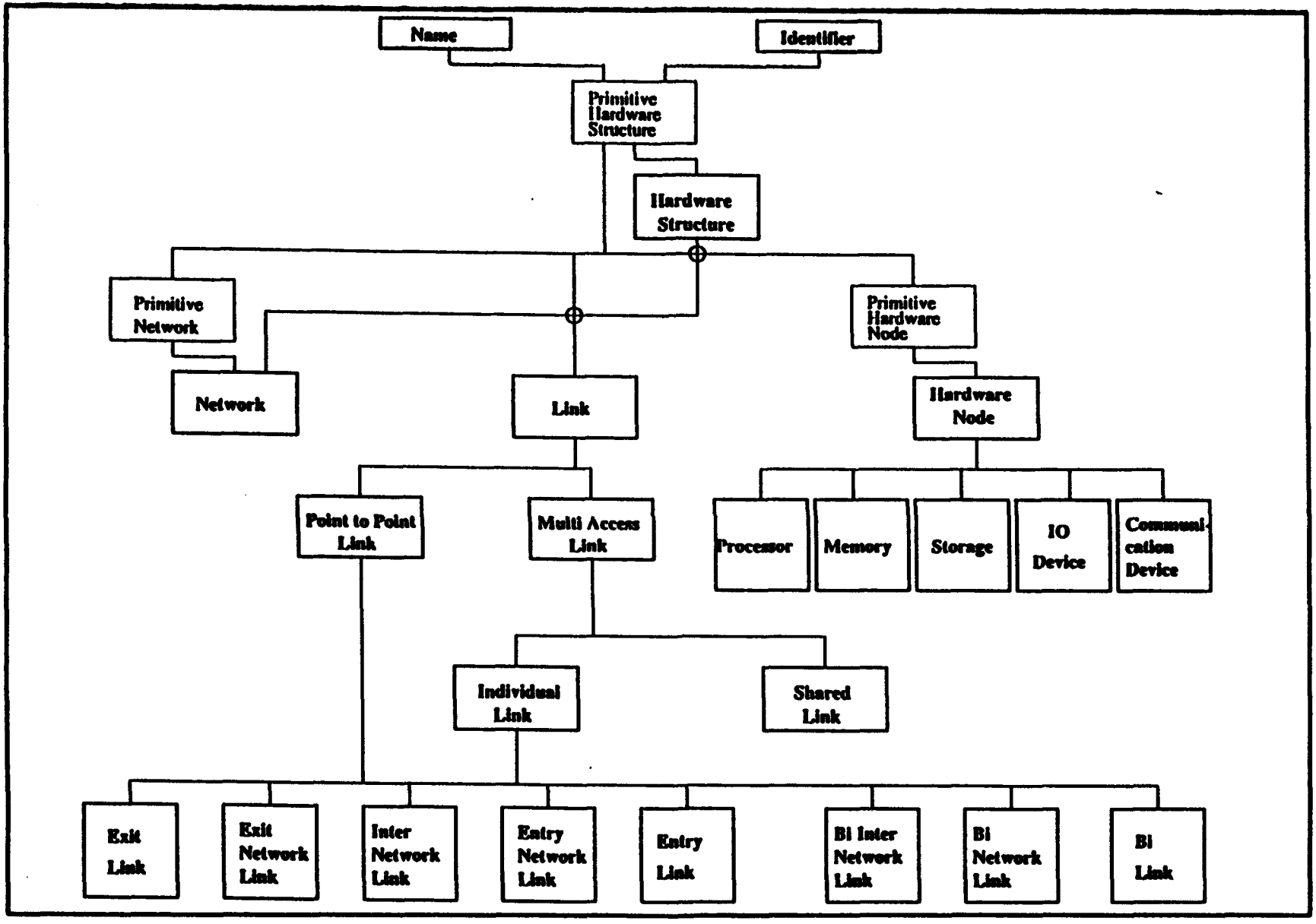


Figure 7. Class Hierarchy for Hardware Structure.

REPRESENTATION AND MEASUREMENT

The Representation of Resources for Large-Sized and Complex Systems

Nicholas Karangelen
John Intintolo
Trident Systems Incorporated

Ngocdung Hoang
Steve Howell
Naval Surface Warfare Center Dahlgren Division

1. INTRODUCTION

The complexity and sheer magnitude of modern intensive systems (including advanced combat, sensors, and weapons systems) require a disciplined structured approach for development. The principal objective of the system development process is to establish a design which satisfies the system requirements and constraints while optimizing the key tradeoffs and issues associated with system functionality, behavior, and implementation. A multi-domain design capture and analysis methodology[1] has been developed under the Office of Naval Research's Engineering of Complex Systems (ECS) Technology Block which partitions the system design into five Design Capture Views addressing the following principal design perspectives: (1) Environmental, (2) Informational, (3) Functional, (4) Behavioral, and (5) Implementation. Of these five capture views, the Implementation Capture View, which addresses the physical systems' architectures (including hardware, software and humanware), is very crucial for the evaluation of the design and yet, is not efficiently addressed by existing system design methods and tools.

An important aspect of the Implementation Capture View is the method for capturing system resources. The ability to capture and analyze system resources early in the design process supports the understanding of how resources are utilized, resulting in a major cost reduction in system integration. The resource capture method needs a robust and flexible mechanism for characterizing system resources including hardware, software, and human operators. The mechanism must also allow the manipulation of system resources such that alternate designs can be traded-off in achieving system requirements. The design of large-sized and complex system requires the analysis of the logical (functional) as well as the physical (resource) aspect of the design. Trade-offs should be performed at all necessary stages of the development process. Hence, the ability to capture system resources at various levels of abstraction and in a systematic and consistency manner is very important for system engineers. The capture of system resources is not only used to search for an optimal design, but is also used to document the rationale of design selections which will be very important when requirements are changed, added, or when new technologies become available.

Although existing analysis tools (i.e., simulation, optimization) provide different mechanisms to specify resources, the representation is very specific to the type of analysis that it supports. The intention of this resource capture approach is to provide a baseline information of the system resources that can be used by various types of analysis. This will minimize the number of assumptions that were generated in the construction of analysis models and guarantee the consistency of the analysis results. Objectives and requirements for an advanced complex system resource modeling methodology and an object-oriented approach for implementing an advanced tool for resource capture will be addressed in this paper.

2. RESOURCE CAPTURE METHODOLOGY OBJECTIVES AND REQUIREMENTS

This resource capture methodology is intended to support the characterization of complex computer-based system resources across the broad spectrum of resource types and at various levels of design detail. The spectrum of resource descriptions needed to capture the

implementation of a complex system design at various points in the design process may vary from very specific to generic, from abstract to detailed, and from simple to complex.

In a top-down design process, the design implementation progresses over time, from high level abstractions of system resources to increasing levels of detailed hardware, software and human operator representations. Analysis and simulation of a particular design option may be required at various points in the design process to address trade-offs or demonstrate compliance with key system requirements. The resource capture methodology must therefore provide a robust flexible method for representing resources of many different types and levels of abstraction. Figure 1 depicts four levels of abstraction, from very highly aggregated resources down to the resources at the sub-component level. (Four levels are picked for illustration purposes only; more or less may be required to support the design capture and analysis needs of a given project.) Early in a top-down design process, the system resources may be represented as large, highly aggregated resources (e.g., subsystems) and the system under design is represented as a combination of interconnected complex subsystems. These subsystem resources are decomposed at the next level of design detail and are represented as aggregations of hardware and software components. At the next level, components are individually represented, including software configuration items with their associated mapping to tasks in real-time system execution and hardware units with their associated lowest replaceable units (LRU's). Typically, the systems engineering activities do not extend below this level of detail; however, this resource capture methodology should support linkage with hardware and software engineering methods and tools to provide support for the entire spectrum of the system development environment.

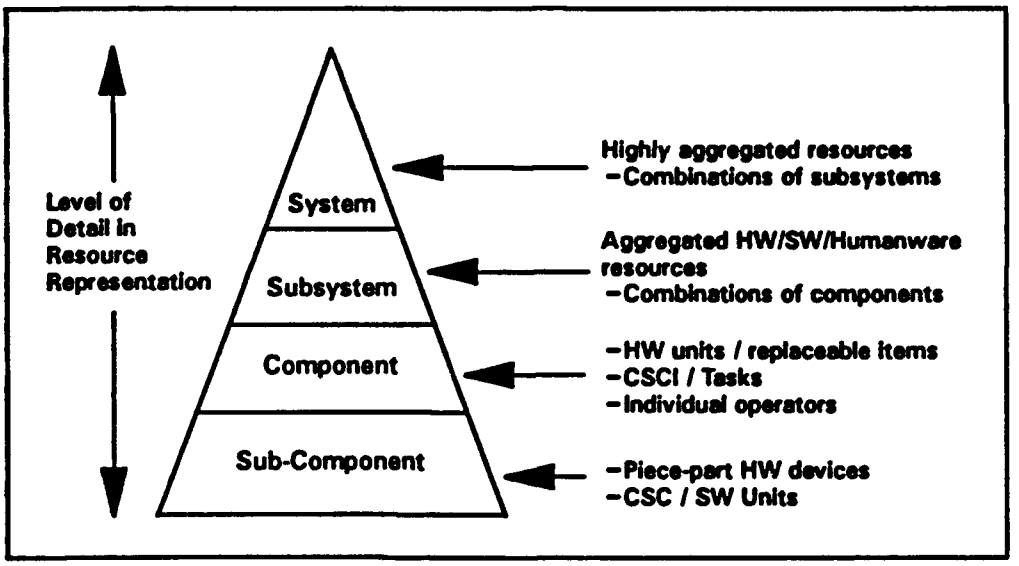


Figure 1. Spectrum of Resource Description

The resource capture methodology must also provide a mechanism for managing complexity in the characterization of resources at various levels of design detail. One or more sets of rules will be established for linking resource representations at a higher level of abstraction to resource representations at the next lower level of design detail. This capability will support decomposing resource descriptions in a top down design scenario or recomposing them in a re-engineering or bottom-up design scenario. When the transition from one to another level of abstraction is not based on decomposition, the capability will provide consistency supports for the transformation. Techniques must also be provided to manage multiple concurrent or alternate resource and design options.

A potentially large number of possible resource types (including hardware, software, and human operator categories) must be captured. A mechanism for efficiently representing a large number of resource types must be provided as well as a flexible, robust approach to portray a diverse spectrum of resource characteristics and attributes. This capability must be extensible to include new resource types as well as new characteristics of existing resource types. A mechanism must also be provided for organizing, accessing and extracting resource descriptions. This capability will establish a library for the captured resource descriptions and will provide an interface to analysis, simulation, optimization, and other tools.

Additional requirements for the resource capture methodology include: (1) defining a formal linkage/mapping with the other design capture views, (2) establishing common formats for resource characterization information exchange, and (3) providing a mechanism for system requirements traceability. The requirements described in this section establish a top level set of objectives for an advanced resource capture methodology.

3. RESOURCE MODEL CONCEPT AND FEATURES

Successful employment of a resource capture and analysis methodology in supporting a complex system design is largely a function of the degree of mechanization which can be achieved. The size and complexity of large scale systems render manual application of any detailed resource capture method unusable. Considerable potential benefits can be gained from a highly automated design capture environment supporting a disciplined structured capture of resource descriptions which can be employed for design analysis, simulation, and optimization activities.

Recent advances in object oriented software engineering techniques provide a powerful mechanism for implementing a tool for resource capture which embodies the requirements of the previous section. Figure 2 illustrates the resource model implementation concept through an Information Model.

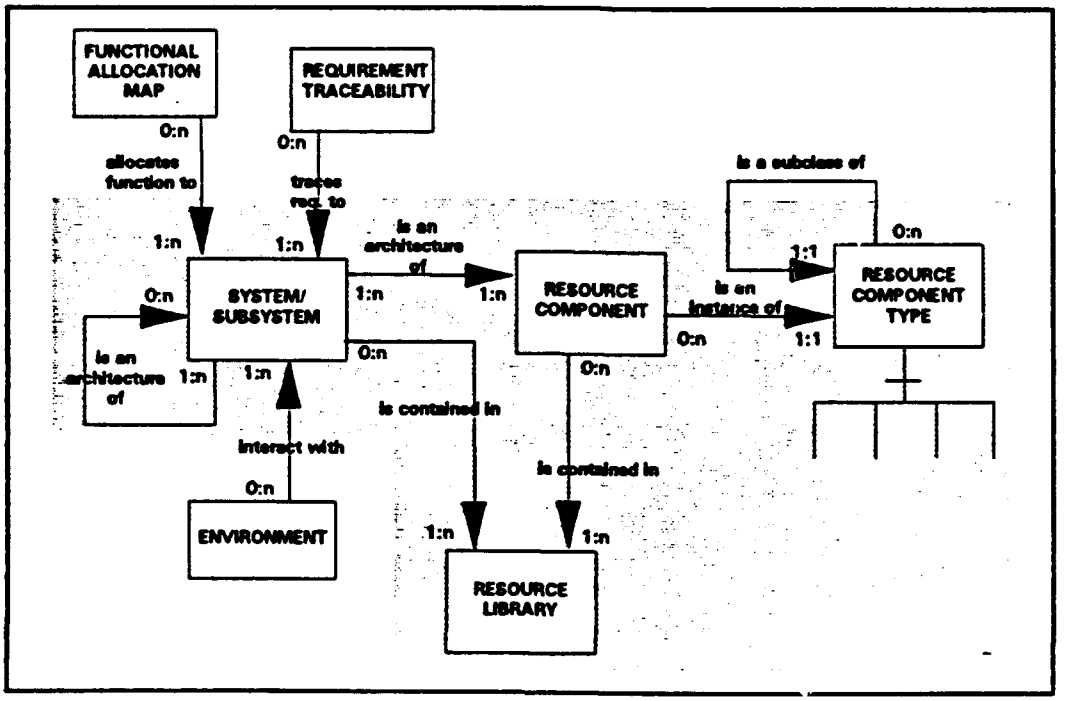


Figure 2. Information Model of Resource Capture Concept

The shaded area is the essential information that will be captured in the resource capture tool and will be addressed in the following sections. The captured system resource architectures are used to evaluate the design in reference to different criteria which are dictated by or derived from the requirements. In order for such analyses to be performed, additional information must be represented and linked to the resource model. Other objects in this figure are examples of the required information that will be used in conjunction with the resource model for the evaluation of the design.

3.1 Resource Component Type

A resource component type is a class structure of the kind of resource that can be used in the system design process. Each resource type can be instantiated to a number of specific resources that share some common properties. The resource component type is characterized by attributes and methods and is organized in a supertype/subtype[2] manner which leverage inheritance and polymorphism to efficiently represent a large number of resources.

Although this classification concept is straightforward, selecting the most efficient classification is very challenging. The selection of resources can be motivated by its functionality (i.e., operating system, beamforming, detection), by its character (i.e., hardware, software), or by its domain specific (i.e., acoustic, electromagnetic environment). Whether a particular classification tree of resource type can support all possible selections is questionable. The multiple inheritance concept can be used to support the cross-reference representation of such structures; however, it will complicate the tracing capability. Figure 3 shows an example of a hardware component type that was selected for the design of a personal computer.

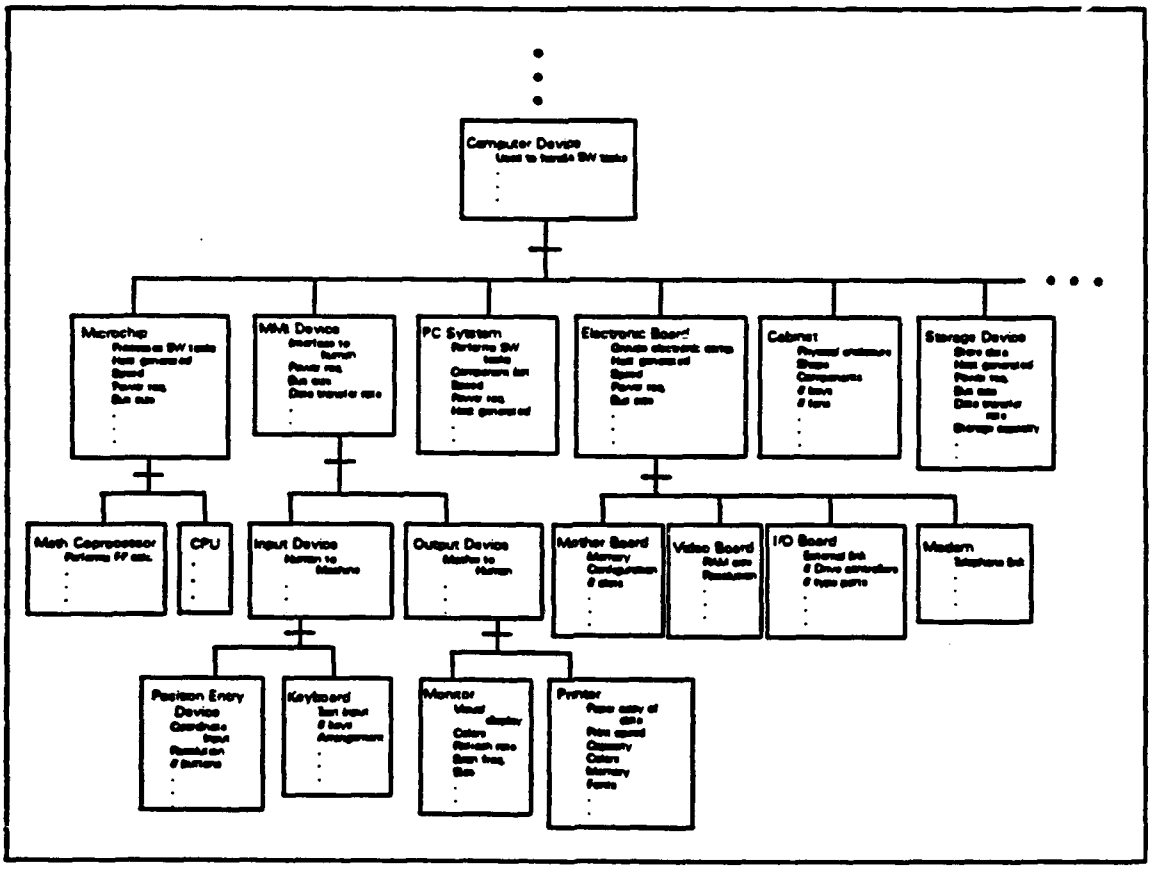


Figure 3. Example of Resource Component Type

3.2 Resource Component

A resource component is an instantiated representation of a resource component type, with specific values for attributes, methods, and interfaces. Hence, a collection of compatible resource components can be derived from an individual resource component type. The representation of resource component can be very generic or very specific depending on how much of its attributes, methods, and interfaces are indicated. The resource component structure can be complex or simple depending on the available information. Hence, component level does not imply that the resource is at its lowest level of decomposition.

3.3 System/Subsystem

A collection of resource components can be connected in a specific manner to form a system/subsystem. Based on different connecting topologies, alternate designs of the same system/subsystem can be derived from a single collection of resource components. The connection and reconnection of system/subsystem resource structures must be generated rapidly and effectively such that multiple evaluation can be performed to achieve an optimal design.

Once the resource components are aggregated into a system/subsystem, its characteristic can be defined through a set of attributes, methods, and interfaces. Since system/subsystem characteristics are not only defined by the combination of its components characteristic, simulation or actual testing can be used to derive appropriated system/subsystem property.

3.4 Resource Library

The resource library is a repository for descriptions of instantiated resource components, subsystems, and systems. It also is a place holder for the system resource information that will be accessed by analysis tools to evaluate the design. Each element in the library includes an excessive amount of information and has multiple related versions. Efficient tracing and version controlling techniques must be incorporated in the management feature of the resource library.

4.0 CONCLUSION

The resource capture tool concept, which includes a graphic interface supporting creation and editing of resource types, instantiation of components, interconnection of components and subsystems, and organization of components and subsystems into libraries, is very essential in the development of large-sized and complex systems. A robust and flexible mechanism is required for characterizing system resources including hardware, software, and human operators. This resource characterization approach is used to support the optimization of the system design in achieving competing system requirements such as performance, reliability, cost, and security. It also provides a vehicle for the selection of hardware, software, and human operation as well as the trade-off between alternate hardware architectures and software partitioning early in the design process.

REFERENCES

1. Mission Critical System Development: Design Views and their Integration, N. Hoang, N. Karangelen, and S. Howell, Technical Report, Naval Surface Warfare Center Dahlgren Division, NSWCDD/TR91-586, Dahlgren, VA, Oct. 1991.

2. Shlaer, Sally and Mellor, Stephen, Object-Oriented Systems Analysis - Modeling the World in Data, Yourdon Press, Englewood Cliffs, NJ, 1988.

A Software Metrics Integration Framework

Dr. William M. Evanco

The MITRE Corporation
Mail Stop Z385
7525 Colshire Drive
McLean, Virginia 22101-3481

e-mail: evanco@mitre.org

Keywords: Software metrics, software quality, multivariate analysis, prediction models, reliability modeling, maintenance modeling

Biographical Sketch: William M. Evanco received the B.S. degree in physics from Carnegie-Mellon University, Pittsburgh, PA, and the Ph.D. degree in theoretical physics from Cornell University, Ithaca, NY. He has been with MITRE Corporation since 1987 and is currently associated with the Space Systems Division in the Center for Civil Systems. His primary research interests are software metrics and software performance modeling.

1.0 Introduction

The production of high quality software systems (e.g., reliable, maintainable) is regarded as an important goal for software development organizations. Much effort has been devoted to assessing these systems based on quantitative measures of quality. Various metrics have been proposed to represent quality as well as the characteristics of both the software product and the system development process contributing to quality. It may not be possible to directly measure quality during the early phases of development. However, the product and process characteristics which emerge during these phases can be used as early indicators of quality.

It has been recognized in recent years that no single attribute of product or process can adequately explain quality. Many features of product and process play a role in determining quality. The challenge is to account for their impacts on quality outcomes in some integrated fashion.

We describe a metrics integration framework which is capable of merging both product and process oriented measures to arrive at a characterization of software quality. The framework is extensible in terms of being able to incorporate new metrics as they become available. In addition, it can accommodate a variety of development contexts (e.g., incremental builds, commercial off the shelf integration, reengineering) and programming languages. Finally, the framework is adaptable in the sense of being able to incorporate data collected by different software analyzers.

2.0 Framework Overview

The framework serves as the basis both for software metrics research and for the application of the research results to assess software development projects.

The major elements of the metrics integration framework shown in Figure 1 involve data collection and analysis, model building and validation, and application to new projects to predict quality outcomes and prescribe system improvements. A database of projects is maintained and continually updated to provide a foundation for ongoing analyses and assessments.

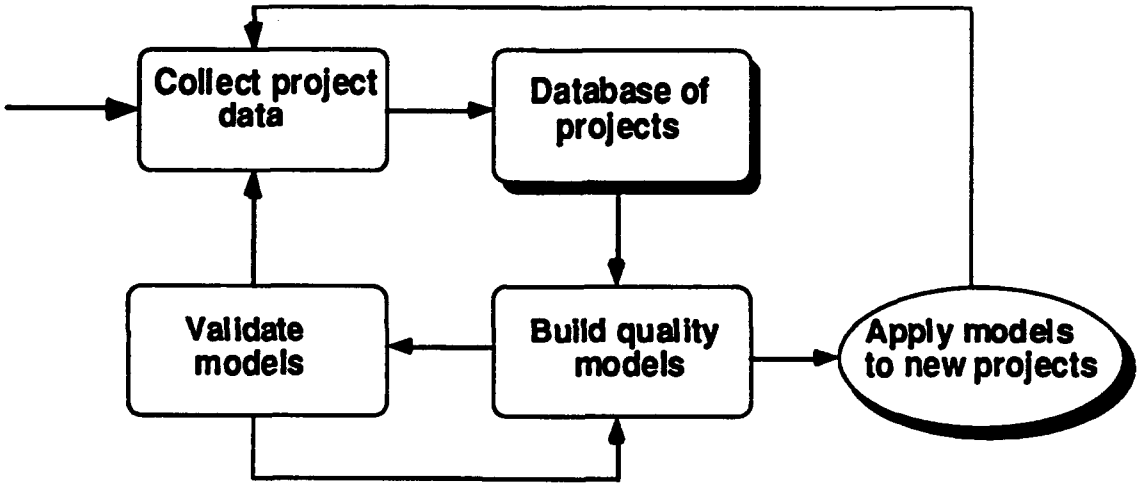


Figure 1: Overview of Integration Framework

Data collection and analysis serves two purposes. First, "historical" data is collected from completed projects to populate a project database. This data is obtained from measurements of software code through the use of software analyzers and from project records such as problem tracking reports. Additional data is collected on the software development environment. The building of the models and their validation depends on this historical data.

Second, to make predictions for projects under development, data is collected during the early development stages from software artifacts (such as designs). This data serves as input to the models predicting software quality, for example, and for the prescription of software improvements. As actual outcome data becomes available, it becomes part of the "historical" project database. This data may be used for model validation and to identify additional enhancements or improvements to the models.

The models thus far built predict the quality factors of reliability, maintainability, and flexibility, as well as lines of code estimation based on early design features [1,2,3,6,7]. The analytical techniques thus far employed to calibrate the models include multivariate regression analysis and ordered response approaches. Also, proportional hazards models have been identified for the analysis of time between failures data. In addition to predicting development outcomes, the models can also be used prescriptively to examine tradeoffs among various software artifact characteristics.

3.0 Data Collection and Analysis

The lack of reliable and appropriate data has been a major impediment in the development of software metrics and their use in system development assessment. A systematic approach is required to collect consistent and accurate data across development projects.

Details of a mechanism for data collection and analysis are shown in Figure 2. The development of a system is conducted within the framework of a development organization. Three types of data can be collected: software artifact data, software project data, and development environment data.

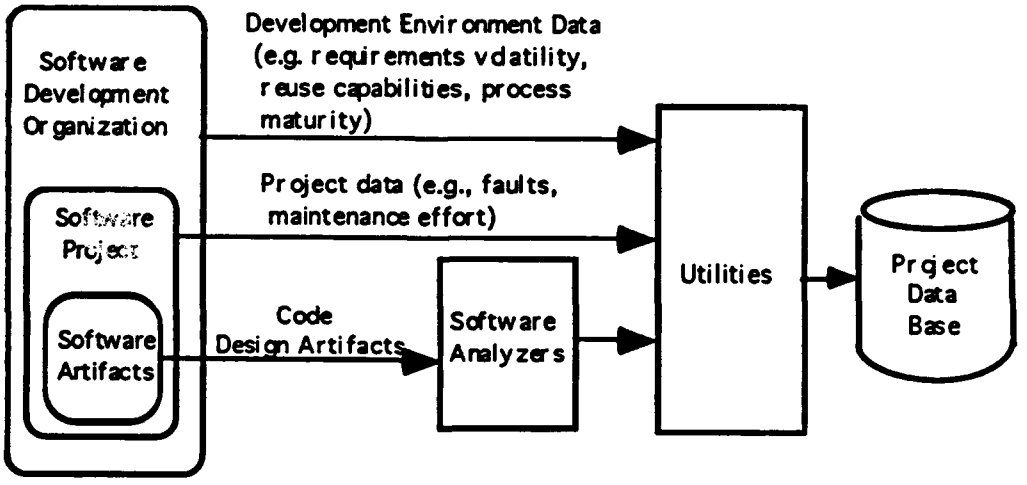


Figure 2: Data Collection and Analysis

3.1 Software Artifact Data

Software artifact data is available at different stages of the development process. The artifact may be software code, for example. Or the artifact may be a software design specified either by means of a design language, or through some Computer Aided Software Engineering (CASE) tool which provides a representational capability for design architecture. As will be discussed in further detail below, this raw data is fed into one or more software analyzers which extract artifact features such as module size, declaration counts, and control flow information. Many of these features are related to the complexity characteristics of the software artifacts, and are used in the derivation of complexity metrics.

3.2 Software Project Data

The second type of data is that which generally cannot be collected from the analysis of software artifacts. During the process of software development, data may be collected on problems uncovered during testing, the effort to isolate or fix software faults, and the effort required to enhance the software. Such data is increasingly available in electronic form, and its extraction and analysis is amenable to automation.

Project data of this sort provides information on software development outcomes and is used to derive quality metrics associated with the software. For example, problem tracking reports collected during testing can be used to identify fault-prone modules or subsystems. Similarly, effort data associated with maintenance activities may help to identify excessively complex portions of the system which are difficult to maintain.

3.3 Development Environment Data

The last type of data characterizes the software development organization in which the effort takes place. This data provides an indication of the organization's contribution to the complexity of the development effort.

For example, an environment in which many requirements changes and additions are made adds to the complexity of the development effort. These changes lead to unplanned adjustments and modifications to the design or implementation. The details of these changes and their impacts may not be well communicated within the organization, resulting in fault propagation throughout the design or implementation.

As another example, a reuse policy in a development organization which encourages the use of previously developed software artifacts in new projects and provides component libraries for its accomplishment will reduce the complexity of its development efforts.

The quality of the software development staff as measured by experience or educational levels and the turnover rates of project personnel are additional characteristics which can influence the complexity of the development effort.

The level of integration of technology (such as CASE tools) and the use of rigorous development methodologies (such as the clean room approach) may also influence the development effort's complexity.

A mature development environment as measured by an organization's ability to treat the software development process as a measurable and controllable activity may also affect the

complexity of the development effort. The Software Engineering Institute (SEI) has developed a process maturity framework for the ranking of software development organizations according to process maturity.

3.4 Data Analysis

Each of the three classes of raw data discussed above must be processed further to provide appropriate metrics for the modeling effort.

Software analyzers reduce the detail of software artifacts to a manageable set of measures. Either source code or program design language code is parsed and then input to a metrics algorithm processor for further analysis. The types of analyses include:

- Dependency analyses to determine various kinds of intermodule dependencies
- Complexity analyses to examine control flow and functional decomposition complexity
- Call graph analyses to determine how parts of a system use or are used by other parts
- Interface analyses to examine the passing of variables between different parts of the program
- Cross-reference analyses to report on the usage of symbolic names
- Standards checking to examine if specific project standards have been followed

Software analyzer outputs are not always in a form appropriate for building models and using them for prediction purposes. The data must be expressed at appropriate levels of software aggregation (e.g., modules, subsystems, configuration items) to facilitate the analytical and evaluation objectives. One can imagine having to address questions about software and its characteristics at both detailed and aggregated levels. For example, how much testing effort should be allocated to a configuration item or subsystem? Or, given a subsystem, which of its modules are expected to be more defect-prone?

To be used as input for analytical models, the software characteristics must be represented by summary statistics. For example, a source lines of code count may represent the size or magnitude of some level of software aggregation such as a subsystem. Similarly, the number of calls by a subprogram to other subprograms could represent a measure of its coupling complexity.

As with software artifact data, the project and development environment data must also be expressed at appropriate levels of aggregation. For example, software fault data collected from problem tracking reports may be rolled up to a subsystem or configuration item level. Similarly, the number of software changes not related to fault correction may be aggregated to obtain a development environment volatility measure at the project level.

Hybrid measures can be derived from the three categories of data discussed above. For example, a fault density measure, defined as the number of faults divided by the number of source lines of code, may be useful as a reliability related measure. Or the staff effort per software change may be taken as a measure of maintainability.

This additional data analysis and reduction is accomplished by means of a set of utilities as shown in Figure 2. The software artifact data generated by analyzers along with project and development environment data are input to these utilities. The outputs are then stored in the project data base.

If new metrics are developed or new data becomes available, these utilities, as a part of the metrics integration framework, may be modified or augmented.

4.0 Model Building and Validation

System development can be viewed in economic terms as a production process. A result of this process is a product, namely, a system of certain capabilities and characteristics created by the application of resources. The characteristics of the system include, for example, the size and complexity of the associated software, and quality aspects such as reliability and maintainability. The resources involve labor to develop the system and capital such as hardware resources to facilitate the development. These resources are brought together by technologies consisting of software engineering methodologies and tools applied during the development effort.

The project database, consisting of historical data on system and development environment characteristics and development outcomes, is the foundation for model building and validation. This data is used to calibrate multivariate statistical models. These models, once calibrated, are validated on the basis of additional project data not used in their calibration. This validation process provides a test of the assumption that the models are applicable to a range of projects and environments different from the ones used for calibration.

The kinds of models which are built are limited by the availability of appropriate data. Typically, an outcome variable characterizes product or process attributes that might be desirable to predict or control. There may be a need for the early identification of an outcome important for the management of a system development project. For example, a manager might want to determine the total effort required to develop a system early in the development life cycle. Therefore, the outcome variable might be the staff years of effort required to complete the project. To aid in estimating the effort, a model may be built to relate effort to the estimated source lines of code to be built as well as other variables such as the complexity of the proposed system. Effort models of this sort have been well documented in the literature [4].

Our focus is on the prediction of software development outcomes related to quality--namely, reliability, maintainability, and flexibility--and on the prediction of lines of code based on software artifacts as they develop through the life cycle.

4.1 Types of Models

Several major categories of analytical models have been identified. The models allow us to conduct multivariate analyses relating system and development environment characteristics to development outcomes. They differ with regard to the level of data aggregation and in terms of the development outcome under consideration.

4.1.1 Multivariate Linear Regression Models

Models of this kind are appropriate when the development outcome variable is either continuous in nature or can be approximated by a continuum. For example, defect density (defects divided by lines of code) is continuous. On the other hand, subsystem defects is

discrete. However, at the subsystem level, the number of defects tends to be large and may be approximated as a continuous variable. At the module level, such an approximation would not be appropriate and the ordered response models described in the next section would be used.

Multivariate linear regression models are expressed as linear combinations of parameters which must be calibrated. Let Y be the outcome variable and $X_1, X_2,...,X_m$ be the system and development environment characteristic variables. The variable Y is a random variable whose expected value is estimated by some function of the explanatory variables $X_1, X_2,...,X_m$. We have used two forms of regression models: those linear in the variables and those logarithmic in the variables. In either case, the equations are linear in the parameters. They are expressed, respectively, as:

$$Y = a_0 + a_1*X_1 + a_2*X_2 + ... + a_m*X_m + \varepsilon \quad (1)$$

$$\ln(Y) = a_0 + a_1*\ln(X_1) + a_2*\ln(X_2) + ... + a_m*\ln(X_m) + \varepsilon \quad (2)$$

where $a_0, a_1,...,a_m$ are the parameters to be estimated, and ε is a residual term accounting for the discrepancy between the actual value of Y and its value as estimated by the remaining terms on the right hand side.

To estimate the parameters, assumptions must be made about the probability distribution of Y and, hence, of ε. The probability distribution of ε is usually assumed to be normal, centered about the origin with standard deviation σ. A normal ε in equation (2) implies that Y is log-normally distributed, assuring that neither Y nor ε is ever negative. For either equation (1) or (2), a least squares approach can be used to estimate the parameters.

Logarithmic forms as shown in equation (2) are useful in representing non-linear relationships between the dependent and independent variables. The logarithmic form also enforces positivity for the dependent variable if all the terms on the right-hand side are real.

Examples of multivariate linear regression analyses are given in [6] for subsystem level defects and in [3] for subsystem level defect densities.

4.1.2 Ordered Response Models

The discreteness of defect measures becomes apparent when analyses are conducted at the module level. The number of defects tends to be small and many modules may have no defects at all. The discreteness and the skewness of the defect distribution toward zero invalidates the normality assumptions associated with the least squares approaches described above.

The number of defects can be viewed as categorical data. A module is classified as having 0, 1, 2, ... , n, or >n defects. Thus, the dependent variable is a discrete categorical variable with n+2 categories. In this case, ordered response models, as discussed by Gurland, et al. [8], can be used.

We hypothesize that the number of defects associated with a module is related to a single measure characterizing its complexity. This complexity measure is a function of both the system and development environment characteristics which are assumed to be of logarithmic form:

$$\ln(C^*) = a_0 + a_1*\ln(X_1) + a_2*\ln(X_2) + \ldots + a_m*\ln(X_m) + \varepsilon \qquad (3)$$

where ε is a normally distributed residual term. The logarithmic form assures that the complexity is a positive quantity. The residual term in (3) incorporates all those characteristics not taken into account in $X_1, X_2, \ldots, X_m$.

The composite complexity measure, C^*, is not observed directly. However, the number of defects is related to complexity, and it is these defects that are observed. The number of defects changes as this measure crosses different thresholds. Expressed mathematically,

$$\begin{aligned} b_{i-1} < \ln(C^*) \leq b_i \quad &\Rightarrow (i-1) \text{ defects if } i=1,2,\ldots,n+1 \qquad (4) \\ &\Rightarrow >n \text{ defects if } i=n+2 \end{aligned}$$

where the b_i represent the thresholds with $b_0 = -\infty$ and $b_{n+2} = +\infty$ and $b_i < b_{i+1}$ for i=0, ..., n+1.

Substituting equation (3) into equation (4) and rearranging provides constraints on the residual, ε:

$$\begin{aligned} b_{i-1} + A*\ln(X) < \varepsilon \leq b_i + A*\ln(X) \quad &\Rightarrow (i-1) \text{ defects if } i=1,2,\ldots,n+1 \qquad (5) \\ &\Rightarrow >n \text{ defects if } i=n+2 \end{aligned}$$

where the parameters and the variables are expressed as vectors, $A = (a_0\ a_1\ a_2\ ..\ a_m)$ and $X^T = (X_1\ X_2 \ldots X_m)$ where X^T is the transpose of a column vector, X

Given a probability density function (PDF) for the residual, the probabilities of 0,1,...,n, >n defects in a library unit can be calculated. The PDF is arbitrary relative to scale and translation transformations, and may be chosen with unit standard deviation centered at the origin It is denoted by the function PDF(u) with corresponding cumulative distribution function, CDF(u).

For a residual, ε, with a normally distributed probability distribution function (PDF), the cumulative distribution frequency (CDF) is the error function denoted by erf(u). From equation (5), the probability of zero defects is the integral of the PDF from $-\infty$ to $b_1+A*\ln(X)$. Similarly, the probability of one defect is the integral from $b_1+A*\ln(X)$ and $b_2+A*\ln(X)$, and so forth.

Thus, the defect probabilities can be written as:

$$\text{Prob(i defects)} = \text{erf}(b_{i+1}+A*\ln(X)) - \text{erf}(b_i+A*\ln(X)) \qquad (6)$$

for i=0, 1, 2, ... , n+1 (where i=n+1 refers to >n defects).

Note that the probabilities given in (6) sum to unity. The expected number of defects is given by:

$$E(\text{defects}) = \sum_{i=0}^{n} i * Pr(i) + E(>n \text{ defects}) * Pr(n+1) \quad (7)$$

where E(>n defects) is the mean number of defects for those library units with more than n+1 defects.

An example of the application of this approach is given in [6] for defect estimation models and in [2] for maintainability models.

4.1.3 Proportional Hazard Models

The models discussed in the previous two subsections are appropriate for the estimation of software defects or faults. In this section, we focus on reliability as measured by the time between failures. Defects or faults are developer oriented measures, while failures are customer oriented. The developer wants to know how many defects must be corrected. The customer wants to know how long he might be able to use the system with failure-free operation. A failure is the result of a fault encountered during system use.

The purpose of testing is to identify failures leading to the isolation and fixing of faults. Also, during field operation, system execution may lead to more failures and result in the repair of additional faults. As faults continue to be repaired, and the system has fewer remaining faults, we expect higher reliability as measured by failure-free operation over longer and longer periods of time.

The *hazard function* (also called the failure rate or force of mortality) for a system of characteristics described by vector X and cumulative execution time T_e for the last failure occurrence is defined by

$$h(t \mid X, T_e) = \frac{f(t \mid X, T_e)}{1 - F(t \mid X, T_e)} \quad (8)$$

where $f(t \mid X, T_e)$ is the probability distribution of t, the time to the next failure, and $F(t \mid X, T_e)$ is the corresponding cumulative distribution function. The characteristics, X, may vary with execution time T_e.

The hazard function represents the conditional probability rate of failure given that the system has survived up to T_e+t. Thus $h(t \mid X, T_e)*\Delta t$ is interpreted as the conditional probability of failure in $[T_e+t, T_e+t+\Delta t]$ given that the system has not failed in $[T_e, T_e+t]$.

Reliability $R(t \mid X, T_e)$ is defined as the probability of failure-free operation in the open interval $[T_e, T_e+t)$. It can be expressed in terms of the cumulative distribution function as

$$R(t \mid X, T_e) = 1 - F(t \mid X, T_e) \quad (9)$$

Recognizing that the probability distribution function is the derivative of the cumulative distribution function, substituting (9) into (8) and integrating, we can express the reliability in terms of the hazard function as

$$R(t \mid X, T_e) = \exp\{-\int_{T_e}^{T_e+t} h(s \mid X, T_e) ds\} \quad (10)$$

Another measure of reliability, the *mean time between failures* (MTBF), can be expressed as

$$MTBF(X,T_e) = \int_{T_e}^{\infty} R(t\,|X,T_e)dt \qquad (11)$$

In order to simplify statistical estimation, Cox [5] proposed a separable form for the hazard function (8) given by

$$h(t\,|X,T_e) = h_0(t)\exp(X\beta + \alpha g(T_e)) \qquad (12)$$

where vector β and scalar α are parameters to be estimated, $g(T_e)$ is some function of the execution time, and $h_0(t)$ is the baseline hazard function.

Prentice, et al. [9] suggest a variation of equation (12) whereby the hazard function depends on $T=T_e+t$ and can be written as

$$h(T\,|X) = h_0(T)\exp(X\beta) \qquad (13)$$

In order to estimate β and $h_0(t)$ from empirical data on failure rates, we could attempt to maximize the likelihood function for the observed data simultaneously for α, β and $h_0(t)$ in equation (12). But based on the separable form for the hazard function, Cox proposed an approach for survival data [5] which was extended by Prentice, et al. [9] to repairable systems. A partial likelihood technique was identified whereby the likelihood function for the estimation for α and β in (12) does not depend on $h_0(t)$. Once β has been estimated, $h_0(t)$ can be estimated in nonparametric form through another likelihood function. Prentice, et al. [9] proposed a similar solution for hazard functions of the form (13).

The approach outlined above is powerful in that it makes no assumptions about the probability distribution of the time between failures in (12) or the cumulative execution time to failure in (13). It is a semi-nonparametric approach in that the reliability estimates depend parametrically on the characteristics but nonparametrically on the time between failures in (12) or the cumulative execution time to failure in (13).

5.0 Development Outcome Prediction

Models of the types described above can be used for predictive purposes on new projects once they have been calibrated on the basis of previous project data. The new projects are at a developmental stage where outcome data is not available, but software and developmental characteristic data, X, are. For example, the complexity characteristics of software begin to emerge during the design stage as the system architecture is developed. A model that relates these complexity characteristics to quality outcomes such as defects can be used to predict the cumulative defects at the end of the testing stage. However, care must be taken when applying the model for predictive purposes. Not all of the variables that could determine outcomes are incorporated in the model. For example, the quality of the development staff may have been excluded as an explanatory variable. If the staff quality for the projects used to calibrate the model were substantially higher than that for the project to be predicted, we might expect the defect predictions to be underestimated.

The lesson to be learned from the above is that models for prediction should not be applied in a casual fashion. The user must have a strong understanding of:

- the basis on which a model was derived
- the range of variation of values of the explanatory variables
- the potential explanatory variables excluded from the model
- the potential impact of these excluded variables.

A large dose of judgment may sometimes be required to use a predictive model intelligently. One may use a model to predict values for some outcome variable of interest. Or the model may be used to rank order a collection of modules, for example, on the basis of their defects.

The latter approach could be less stringent regarding the assumptions on which the model is based. Suppose a model of the kind expressed by equation (2), for example, were used to predict defect density. Calibrating the model on the basis of a set of projects developed with relatively high quality staff would result in specific values of the parameters a_0, a_1, ..., a_m. If a second set of projects involving low quality staff were pooled with the first set and the model were recalibrated, we might have a model of the sort

$$\ln(Y) = a_0+b_0*D_s + a_1*\ln(X_1) + a_2*\ln(X_2) +... + a_m*\ln(X_m) + \varepsilon \qquad (14)$$

where D_S is a dummy variable equal to zero for the projects with high staff quality and unity for the projects with low staff quality. Thus, the impact of staff quality is in the difference of the constant terms for the two types of projects. High staff quality projects have a constant term a_0 while the constant term for projects with low staff quality is a_0+b_0. A positive b_0 would yield a higher defect density for the projects with lower staff quality. A model of the form (14) is, in principle, empirically testable, as is the hypothesized sign for b_0.

Suppose staff quality enters as a dummy variable as in (14) or as an additional additive term $a_{m+1}*\ln(X_{m+1})$. If these terms were ignored and a model of the form (2) were calibrated only on the basis of projects with high quality staff, then it would be possible only to rank-order subsystems of new projects, for example, according to their relative defect densities. To provide information about the actual defect densities would require knowledge about b_0 or a_{m+1}.

While the initially calibrated model may not provide information about either b_0 or a_{m+1}, there are contexts in which this information can be obtained through analytical means. For example, if a project is developed as a series of incremental builds, information about defects, for example, associated with build N, can be used to provide an estimate of the parameter, b_0, and the resulting recalibrated model can be used to predict defects for subsequent builds N+r where r>0.

6.0 Software Improvement Prescription

The previous section discussed the use of models to predict development outcomes. In this section, we focus on the use of statistical models to prescribe changes for improving system quality.

In Ada systems, context coupling of library unit aggregations (LUAs) is a contributor to software complexity and, hence, to defect densities [1]. A LUA consists of a specification,

a body, and any related subunits. Context coupling of LUAs is achieved by the use of a "with" clause and allows for the exportation and importation of visible declarations among LUAs. A LUA may use any of the imported declarations as resources in its implementation.

The relationship between context coupling and defect density was found to be [3]

$$\ln(\text{Defect Density}) = .53 + .45*\ln(\text{Context Coupling}) \qquad (15)$$

with a coefficient of determination, R^2, equal to .63.

Now consider a design with a context coupling profile as shown by the solid line in Figure 3. This figure represents the percent of LUAs having context coupling greater than a certain value. For example, about twenty-seven percent of the LUAs have a context coupling greater than fifteen and about seven percent of the LUAs have a context coupling of forty-five or more. The mean context coupling of this design is 10.2 "withs" per library unit aggregation. Using equation (15), the system has a predicted defect density of 4.81 defects per thousand lines of source code. A one million source line of code system would contain 4810 defects.

A redesign of the system might consider those LUAs in the tail end of the profile. These LUAs tend to be large and usually require extensive resources imported from other LUAs through context coupling. A programmer responsible for implementing such LUAs is confronted with a large number of declarations, including those defined within the LUA as well as those which are imported through context coupling. The resulting complexity may lead to larger defect densities in these LUAs.

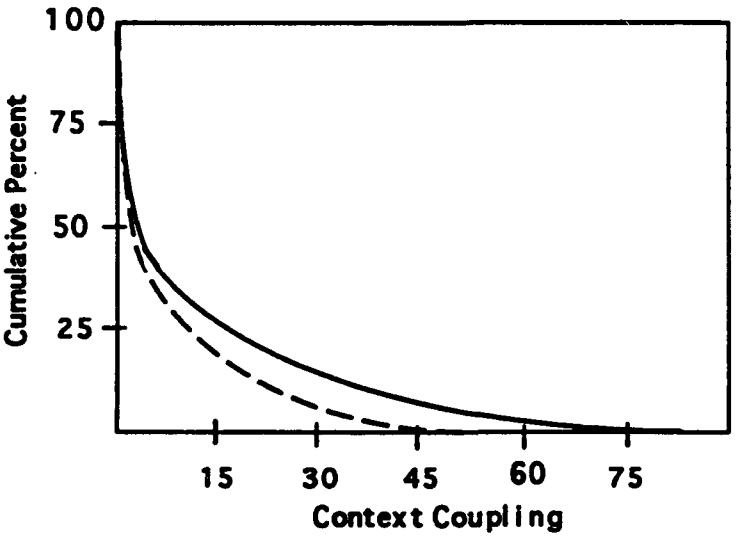


Figure 3: Context Coupling Profile

Dividing the large context coupling LUAs (>45 "withs") in the tail of Figure 3 into smaller units reduces the context coupling profile to that shown by the dashed line. The redesigned system has an average coupling of 7.1 "withs" per LUA. The predicted defect density is then reduced to 4.08 defects per thousand lines of source code. Assuming that the redesigned system has approximately one million source lines of code, the predicted

number of defects is 4080 for a net saving of 730 defects. The number of defects is thus reduced by about fifteen percent.

This analysis can be extended to estimate the cost saving of the redesign. The cost saving depends on the testing cost to uncover each of the 730 defects, the cost of isolating the defect to a specific portion of code, the cost of fixing the defect, and the cost of additional regression testing.

Another way of looking at the analysis leading to equation (15), which does not depend on the constant term, is to take the differential of (15) yielding

$$\frac{\Delta DD}{DD} = .45 * \frac{\Delta CC}{CC} \qquad (16)$$

where DD refers to defect density and CC to context coupling.

This equation indicates that a ten percent change in the context coupling results in a 4.5 percent change in the defect density. Reducing context coupling from 10.2 to 7.1 represents about a thirty percent change. This leads to about a fifteen percent change in defect density consistent with the analysis above.

The utility of equation (16) is that knowledge of the value of the constant term in equation (15) is not required. Therefore, it may have greater applicability across projects substantially different from the ones used to estimate equation (15).

7.0 Conclusions

We have presented a framework for the development of software metric models and their applications to the assessment of development projects. The framework is robust in that it can incorporate a variety of software metrics and can accommodate different programming languages and development contexts.

We have discussed several statistical analysis techniques which span different levels of system aggregation and focus on different outcome variables. These techniques provide the basis for building models to predict development outcomes and to prescribe system changes that may improve development outcomes.

This framework is the foundation for continuing work to enhance the technology for quantitative system evaluation and improvement.

REFERENCES

1. Agresti, W., W. Evanco, M. Smith (1990), "Early Experiences Building a Software Quality Prediction Model," Proceedings of the Fifteenth Annual Software Engineering Workshop, NASA/GSFC.

2. Agresti, W. W., W. M. Evanco, D. D. Murphy, W. M. Thomas, B. T. Ulery (1991), "An Approach to Software Quality Prediction from Ada Designs," Proceedings of the Third Annual Software Quality Workshop, Rochester, NY.

3. Agresti, W. W., and W. M. Evanco (1992), "Projecting Software Defects from Analyzing Ada Designs," IEEE Transactions on Software Engineering, Volume 18, Number 11, pp. 988-997.

4 Boehm, B. W. (1981), Software Engineering Economics, Englewood Cliffs, NJ: Prentice-Hall.

5. Cox, D. R. (1975), "Partial Likelihood," Biometrika, Volume 62, pp. 269-276.

6. Evanco, W. M. and W.W. Agresti (1992), "Statistical Representation and Analyses of Software," in Proceedings of the Seventeenth Symposium on the Interface of Computer Science and Statistics, College Station, TX.

7. Evanco, W. M. and W.W. Agresti (1993), "Software Defect Prediction," manuscript submitted for publication.

8. Gurland, J., T. Lee, and P. Dahm (1960), "Polychotomous Quantal Response in Biological Assay," Biometrics, Volume 16, pp. 382-398.

9. Prentice, R. L., B. J. Williams, and A. V. Peterson (1981), "On the Regression Analysis of Multivariate Failure Time Data," Biometrika, Volume 68, pp. 373-379.

MEASUREMENT AND EVALUATION OF COMPLEX NAVY SYSTEM DESIGNS

Osman Balci
David DeVaux
Richard E. Nance

Department of Computer Science
and
Systems Research Center
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061-0106

ABSTRACT

The purpose of this paper is to present a multifaceted approach to the measurement and evaluation of complex Navy system designs with embedded real-time mission critical characteristics. The approach advocates the use of a visual simulation model, constructed in the Visual Simulation Support Environment (VSSE), representing the design so as to achieve dynamic measurement of the design. A knowledge-based approach is proposed for the independent evaluation of hundreds of design indicators. The Objectives / Principles / Attributes (OPA) framework is used as the underlying structure for the measurement and evaluation of all three major aspects of a system design: project, process, and product.

1. INTRODUCTION

A complex Navy system is composed of three major components: software, hardware, and "humanware". These components are intertwined with real-time mission critical characteristics and pose significant technical challenges for system designers, developers, and maintainers independent of warfare area.

A critically important phase in the engineering of complex systems methodology is system design measurement and evaluation. Before making the commitment to spend millions of dollars for building a system, its design must be carefully evaluated.

The purpose of this paper is to present an approach for measurement and evaluation of complex Navy system designs. Section 2 describes some basic concepts of measurement and the terminology. The proposed approach based on the use of simulation modeling, indicators (metrics), and the OPA framework is described in Section 3. After concluding remarks in Section 4, a bibliography is given.

2. THE MEASUREMENT SCHEME

Measurement is an important activity of interest in many disciplines. However, a standard terminology does not exist. Different terms are used in different disciplines to convey the same notion. The term "metric" is used in Software Engineering in measurement of software quality characteristics and software project, process, and products. The terms "Measure" and "Index" are used in Computer Performance Evaluation in measurement of different aspects of a computer system. The terms "Scale" and "Factor" are used in statistical measurement. The term "Indicator" is used in Economics for measurement of the economy (e.g., leading economic indicators) and in Psychometric Theory in measurement of psychological problems.

The common goal in all these disciplines is to try to accurately measure a concept which can be either quantitative or qualitative. Measurement of quantitative concepts (e.g., response time, throughput, utilization) can be done directly. Whereas, qualitative concepts (e.g., design utility, maintainability, complexity) must be measured indirectly by using a hierarchy of indicators as shown in Figure 1.

In this paper, we use the term "indicator" and define it as an indirect measure of a qualitative concept. We define the term "metric" as an indicator the value of

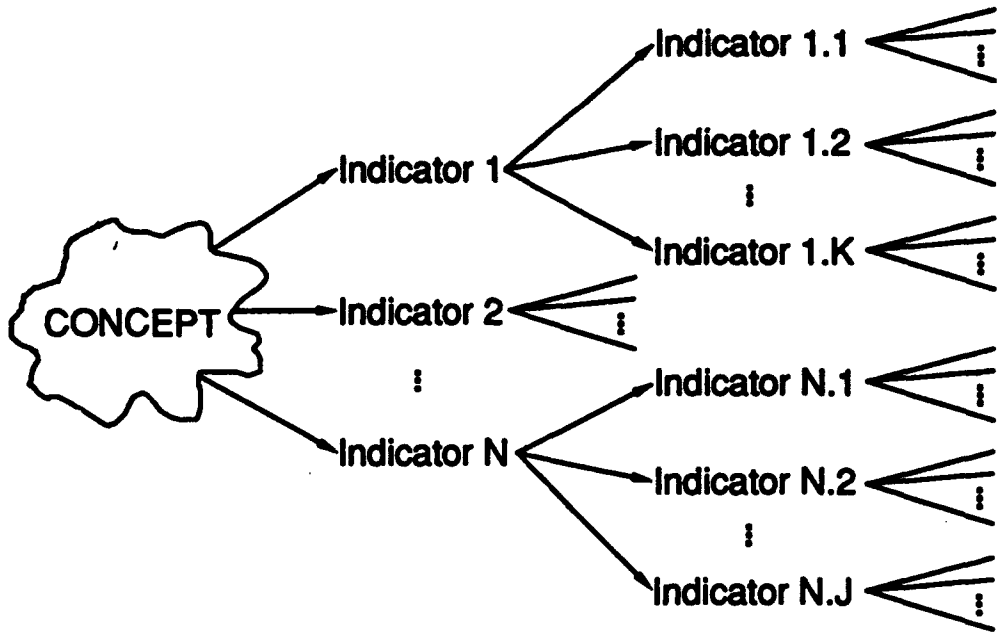


Figure 1. Measurement of Qualitative Concepts

which can be computed by using a formula. A literature survey reveals three categories of indicators: (1) process indicators (e.g., standards, design methodology), (2) product indicators (e.g., execution efficiency, reliability, usability), and (3) external indicators (e.g., security, physical, financial).

A qualitative concept (e.g., system design utility) is measured by N indicators at the first level. Those indicators which cannot be directly measured are further decomposed into other indicators at the second level. This decomposition continues until the indicators at the base level (i.e., the ones that are not decomposed further) are directly measurable. Figure 2 presents a hierarchy of indicators for measurement of system design utility. Note that the decomposition of indicators in Figure 2 is not carried completely, and many indicators shown at the base level are not directly measureable.

Measurement of a concept using indicators carries some error:

CONCEPT = INDICATORS + ERROR

Validation and verification of indicators deal with determining the amount of error in the measurement process and require collection of data from the application of the indicators to a large number of cases. This data collection is extremely time consuming and very expensive in many cases.

3. THE PROPOSED APPROACH

Measurement and evaluation of a complex Navy system design represented in a static manner (on paper) cannot be made convincingly. The system being represented is typically embedded within a larger system. Performance parameters are carefully defined, dependent on an equally careful assessment of mission requirements. Introducing even greater challenge is the real-time or time-critical dimension of system behavior. Therefore, the system design must be represented in a dynamic fashion so that its measurement and evaluation can capture all of its dynamic characteristics. We propose to represent a complex Navy system design in the form of a visual simulation model and experiment with it for the purpose of measurement and evaluation of the system design as illustrated in Figure 3. However, we realize that the development of a visual simulation model of a complex system design is very complex and difficult itself. Therefore, we propose to use the Visual Simulation Support Environment (which can also be characterized as computer-aided visual simulation software development environment) a fully functional prototype of which has been constructed at Virginia Tech. The simulation environment is briefly described in Section 3.1.

The measurement of a complex Navy system

- System Design Utility
  - Performance
    - Response Time
    - Efficiency
      - Device efficiency
      - Accessibility
      - Conciseness
    - Predictability
    - Throughput
    - Time Complexity
  - Reliability
    - MTTF - Mean Time to Failure.
    - MTBF - Mean Time Between Failures
    - Accuracy
      - Informal Testing
        - Desk Checking
        - Structured Walkthroughs
        - Inspections
        - Reviews
        - Audit
      - Static Testing
        - Structural Analysis
        - Consistency Checking
      - Dynamic Testing
        - Top-Down Testing
        - Bottom-Up Testing
        - Black-Box Testing
        - White-Box Testing
        - Stress Testing
        - Execution Monitoring
        - Execution Profiling
        - Regression Testing
      - Symbolic Testing
        - Path Analysis
        - Cause-Effect Graphing
      - Constraint Testing
        - Assertion Testing
        - Boundary Analysis
        - Inductive Assertions
    - Fault Tolerance
      - Graceful Degradation
      - Redundancy
      - Crash Recoverability
    - MTTR - Mean Time to Repair
    - Availability
    - Computation Heavy Process (Stress) Effects
    - Complexity
      - Size
        - Halstead's program length
          - n1
          - n2
        - Jensen's program length
          - n1
          - n2
        - Function Points
      - McCabe's cyclomatic number
      - Henry/Kafura information flow metric
      - IF4 information flow metric
      - Belady's bandwidth
      - System complexity
        - Data complexity (D)
        - Structural complexity (S)
      - Procedural complexity
      - Functional complexity
      - Whitworth & Szulewski control flow complexity
      - Whitworth & Szulewski data flow complexity
      - McClure's Module Invocation Complexity
      - Woodfield's Review Metric
      - Woodward's K
      - Chen's MIN
      - Benyon-Tinker's Cx
    - Intensity of program use
    - Program age
  - Maintainability
    - Correctibility
    - Extensibility
    - Complexity†
    - Adaptability
    - Information hiding
      - Coupling
      - Cohesion
      - Well-Defined Interface
    - Modularity
    - Connectivity
      - Functional connectivity
      - Data connectivity
    - Stability
      - Yau & Collofello stablity metric
      - Requirements Stability
  - Cost
    - Development Cost
      - COCOMO
        - Size†
      - Complexity†
      - SLIM
      - ESTIMACS
    - Testing Cost
    - Maintenance Costs
    - Operation Costs
    - Productivity
      - Source lines of code per work month
      - Dollars expended per line of source code
    - Purchase Costs
    - Personnel Costs
    - Cost of implementing security barriers
      - Initial cost
        - Analysis of security requirements
        - Implelementation
        - Validation and testing
      - Operational cost
        - Extra CPU time required for logging
        - Encipherment time coefficient
        - Maintenance of authorization matrix
        - Restriction of previously offered services
  - Security
    - Expected time to break password(s)
    - Size of the authorization matrix
    - Depth of the authority levels
    - Quality of the introduced cryptography system
  - Usability
    - User intensity
    - Required number of operators
    - Number of simultaneous users
    - Ease of use
    - User response time
    - Completeness
    - Communicativeness
    - Training

† See the decomposition of this indicator given earlier in the hierarchy.

Figure 2. A Hierarchy of Some Indicators for System Design Measurement

- User interface
  - # Mnemonics/commands
  - Length of mnemonics (commands)
  - # of mnemonics beginning with same character
  - Response time
  - Average # choices in menu
  - Avg. # of menus in one procedure sequence
  - Good or bad graph representations
  - Good or bad error messages
  - Number of error messages
- Installability
  - # of Downs/day
- Reuseability
  - Information Hiding†
  - Hierarchial Decomposition
  - Functional Decomposition
  - Modularity
  - Portability
  - Complexity†
- Testability
  - Hierarchial Decomposition
  - Functional Decomposition
  - Information Hiding†
  - Exhaustivity of test data
  - Modularity
  - Adverse testing effects
  - Complexity†
- Understandability
  - Number of functions
  - Amount of data processed
  - Information hiding†
  - Modularity
- Correctness
  - Requirements Traceability
  - Requirements Definition
  - Completeness
  - Consistency

† See the decomposition of this indicator given earlier in the hierarchy.

Figure 2. A Hierarchy of Some Indicators for System Design Measurement (Continued)

design involves indicators that can be assessed only by experts with intimate knowledge of the mission areas for which the system is intended. Therefore, we propose to use a knowledge-based approach as described in Section 3.2.

We envision the use of hundreds of indicators for the measurement and evaluation of a complex Navy system design. Some of these indicators can be measured by system engineers, some (especially those indicators for dynamic system characteristics) can be measured by using the visual simulation model representing the system design, and some can be measured by only the warfare domain experts.

All these indicators need to be applied under a framework in order for the measurement to be effective. The needed framework is the Objectives/Principles/Attributes (OPA) framework developed at Virginia Tech and is briefly described in Section 3.3.

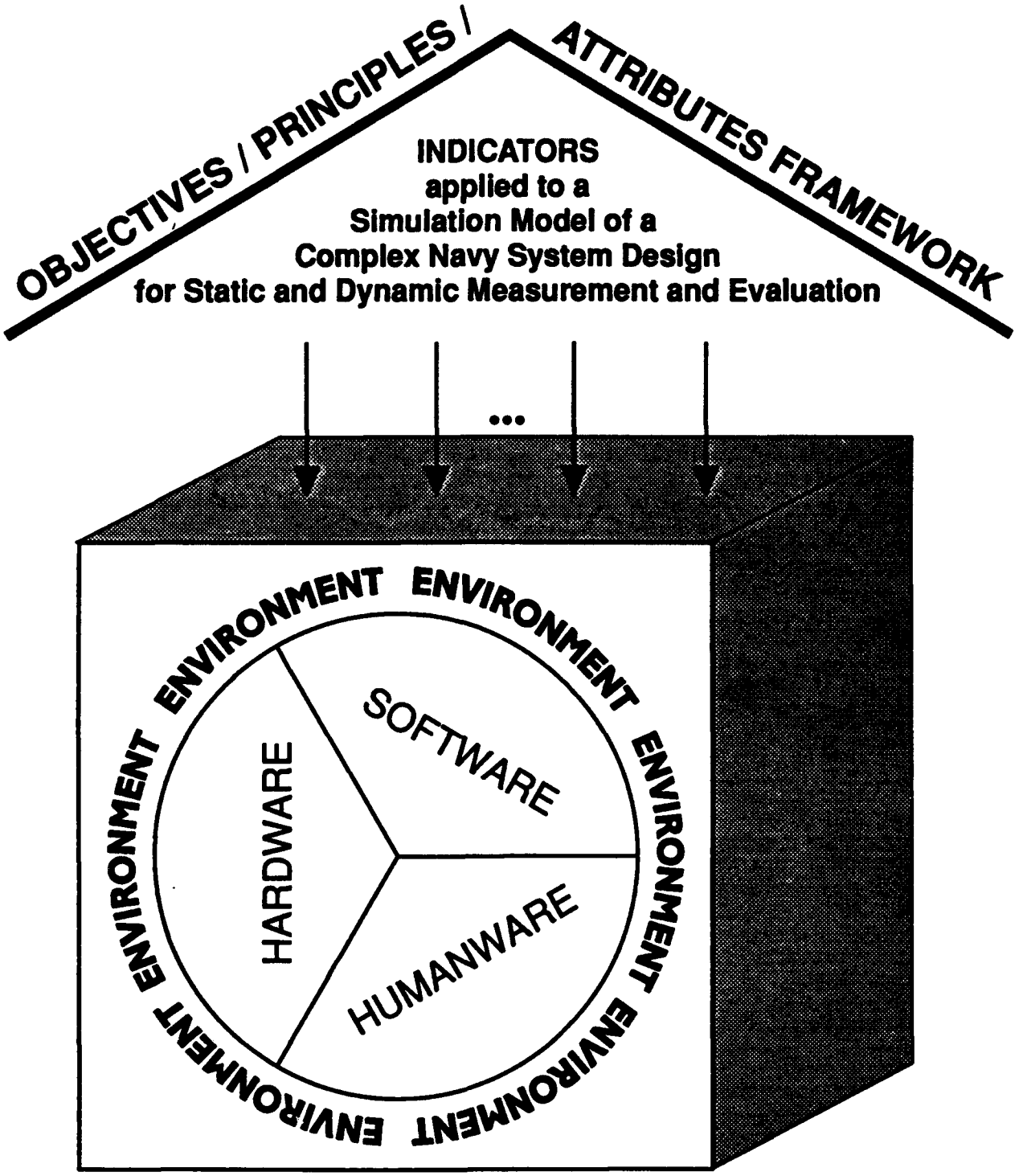
3.1 The Visual Simulation Support Environment

The ever-increasing complexity of visual simulation model development is undeniable. A simulation programming language supports only the programming process—one of 10 processes in the life cycle of a simulation study [Balci 1990]. Automated support throughout the entire visual simulation model development life cycle is crucially needed. This support can be provided in the form of an environment composed of integrated software tools providing computer-aided assistance in the development and execution of a visual simulation model.

A collection of computer-based tools makes up a development environment if, and only if, the tools are highly integrated and work under a unifying Conceptual Framework (CF). The Simulation Model Development Environment (SMDE) research project [Balci 1986; Balci and Nance 1987a, 1992] at Virginia Tech has recently achieved the automation-based software paradigm [Balci and Nance 1987b] and developed: (1) the multifaceteD cOnceptual fraMework for vIsual simulatioN mOdeling (DOMINO) [Derrick 1992; Derrick and Balci 1992a], (2) the Visual Simulation Support Environment (VSSE) [Derrick 1992; Derrick and Balci 1992b], and (3) the Visual Simulation Model Specification Language (VSMSL) [Derrick 1992; Derrick and Balci 1992c].

The rapid prototyping technique has been used in the VSSE's evolutionary joint development with the DOMINO. Many VSSE tool prototypes have been developed, implemented, experimented with, and documented. Some prototypes have been discarded; however, the experience and knowledge gained through experimentation with those prototypes have been kept.

Figure 4 depicts the VSSE architecture in four layers: (0) Hardware and Operating System, (1) Kernel



A VISUAL SIMULATION MODEL OF A COMPLEX NAVY SYSTEM DESIGN

Visual Simulation Support Environment (VSSE)

Figure 3. Simulation Model-Based Evaluation of Complex Navy System Designs Using Indicators

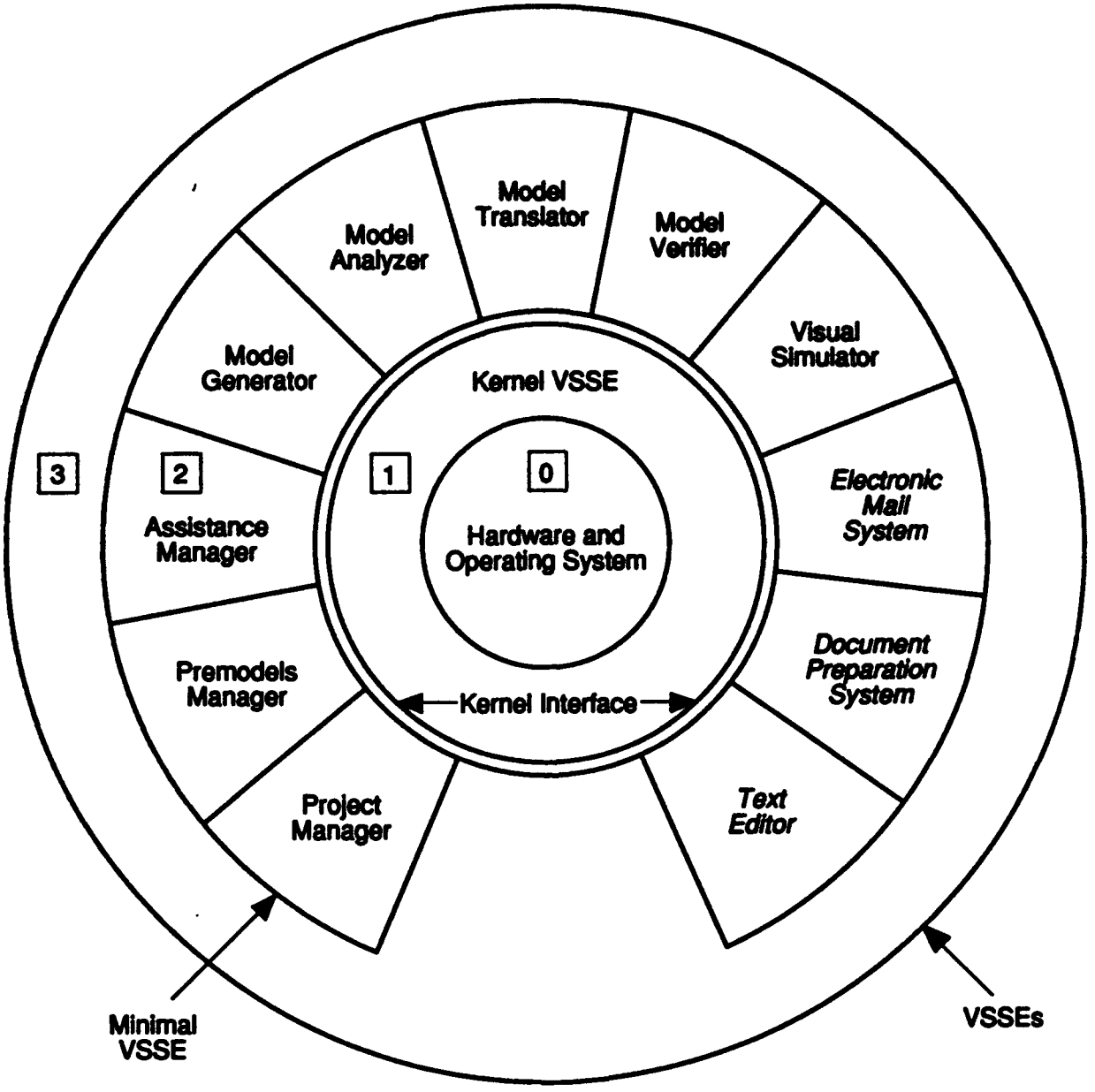


Figure 4. Visual Simulation Support Environment Architecture

VSSE, (2) Minimal VSSE, and (3) VSSEs.

3.1.1 Layer 0: Hardware and Operating System

A Sun computer workstation constitutes the hardware of the VSSE. The UNIX SunOS operating system and utilities, SunView graphical user interface, and INGRES relational database management system constitute the software environment upon which the VSSE is built.

3.1.2 Layer 1: Kernel Visual Simulation Support Environment

Primarily, this layer integrates all VSSE tools into the software environment described above. It provides INGRES databases, communication and run-time support functions, and a kernel interface. Three INGRES databases occupy this layer, labeled project, premodels, and assistance, each administered by a corresponding manager in layer 2. All VSSE tools are required to communicate through the kernel interface. Direct com-

munication between two tools is prevented to make the VSSE easy to maintain and expand. The kernel interface provides a standard communication protocol and a uniform set of interface definitions. Protection is imposed by the kernel interface to prevent any unauthorized use of tools or data.

3.1.3 Layer 2: Minimal Visual Simulation Support Environment

This layer provides a "comprehensive" set of tools which are "minimal" for the development and execution of a visual simulation model. "Comprehensive" implies that the toolset is supportive of all model development phases, processes, and credibility assessment stages. "Minimal" implies that the toolset is basic and general. It is basic in the sense that this set of tools enables modelers to work within the bounds of the minimal VSSE without significant inconvenience. Generality is claimed in the sense that the toolset is generically applicable to various simulation modeling tasks.

Minimal VSSE tools are classified into two categories. The first category contains tools specific to simulation modeling: Project Manager, Premodels Manager, Assistance Manager, Model Generator, Model Analyzer, Model Translator, Model Verifier, and Visual Simulator. The second category tools (also called assumed tools or library tools) are expected to be provided by the software environment of Layer 0: Electronic Mail System, Document Preparation System, and Text Editor.

3.1.4 Layer 3: Visual Simulation Support Environments

This is the highest layer of the environment, expanding on a defined minimal VSSE. In addition to the toolset of the minimal VSSE, it incorporates tools that support specific applications and are needed either within a particular project or by an individual modeler. If no other tools are added to a minimal toolset, a minimal VSSE would be a VSSE.

The VSSE tools at layer 3 are also classified into two categories. The first category tools include those specific to a particular area of application. These tools might require further customizing for a specific project, or additional tools may be needed to meet special requirements. The second category tools (also called assumed tools or library tools) are those anticipated as available due to use in several other areas of application: statistical analysis of simulation output data, designing simulation experiments, documentation and credibility assessment, and input data modeling. Some examples of such tools comprise layer 3.

A VSSE tool at layer 3 is integrated with other VSSE tools and with the software environment of layer 0 through the kernel interface. The provision for this integration is indicated in Figure 4 by the opening between Project Manager and Text Editor. A new tool can easily be added to the toolset by making the tool conform to the communication protocol of the kernel interface.

The VSSE was developed by using the C programming language, SunView graphical user interface, Sun programming environment, and INGRES relational database management system Embedded QUEry Language/C (EQUEL/C). It encompasses more than 50,000 lines of documented code and runs on a Sun color workstation.

Currently, we are building the production version of the VSSE under the NeXTstep object-oriented display postscript-based Operating System.

3.2 The Knowledge-Based Evaluation

The overall evaluation of a complex Navy system design must be conducted independently. The organization which creates the system design is not qualified to also perform its final overall evaluation because of the "developer's bias". We envision the scenario illustrated in Figure 5 in which an independent organization is charged with the task of measurement and evaluation of the system design. This organization can be independent to the sponsoring and developing organizations or it can be a branch within the sponsoring organization.

Based on the warfare domain the system design is intended for, hundreds of indicators should be identified for measurement and evaluation. Some of these indicators can be measured by the use of a simulation model representing the system design, some can be computed by using a formula, and some need to be

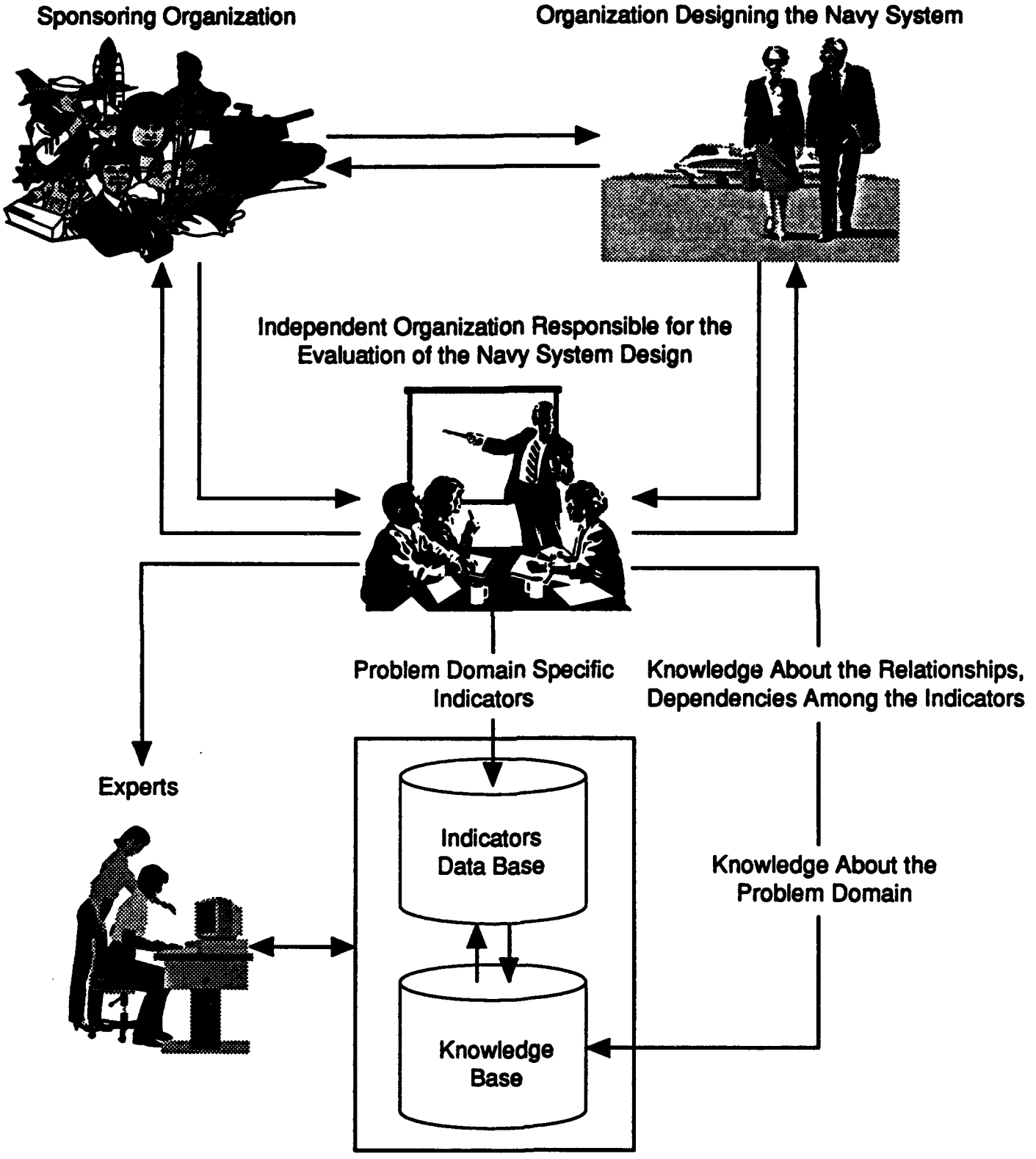


Figure 5. Knowledge-Based Evaluation of Complex Navy System Designs Using Indicators

assessed by expert people who have intimate knowledge of the warfare domain.

The assessment of indicators requires knowledge about the problem domain and knowledge about the relationships and dependencies among the indicators. These types of knowledge are essential for scoring on the indicators.

Thus, the evaluation process using hundreds of indicators and the knowledge base becomes a very complex process requiring computer-aided assistance. Such assistance is being provided in a software tool called SENATE which is under development.

Figure 6 shows the SENATE tool browser containing the hierarchy of indicators presented in Figure 2. The SENATE allows the user to create, modify, delete, weight, and score on the indicators. An expert system shell under development will enable the system administrators to store rule-based knowledge into the inference engine of the SENATE. The knowledge is used in the background during weighting and scoring on the indicators.

3.3 The OPA Framework

The Objectives/Principles/Attributes (OPA) framework [Arthur and Nance 1987, 1990; Arthur et al. 1993; Nance and Arthur 1988] characterizes the *raision d'etre* for software engineering and establishes definitive linkages among project level objectives, software engineering principles, and desirable product attributes as illustrated in Figure 7.

The OPA framework is applied in the organization and application of indicators throughout the system engineering life cycle. It is essential that we not only measure and evaluate the system design, but also the process by which the design is created under project level objectives. The OPA framework provides a comprehensive view covering project, process, and product measurement and evaluation.

4. CONCLUDING REMARKS

The Navy systems are indeed very complex and contain three diverse components: software, hardware, and "humanware". These components are intertwined with real-time mission critical characteristics and pose significant technical challenges for system designers, developers, and maintainers independent of warfare area. A credible approach to the assessment of such a complex system must include at least the following elements: (1) a simulation model built to represent the system design so that dynamic measurement can be done; (2) identification of hundreds of qualitative and quantitative indicators to measure software, hardware, and "humanware" components of the system design; (3) a knowledge-based system that provides computer-aided assistance in the evaluation process; (4) conducting the evaluation in an independent fashion, preferably by a third party; and (5) identifying experts to evaluate some of the indicators based on their expert knowledge.

ACKNOWLEDGEMENTS

This research was sponsored in part by the U.S. Navy under contract N60921-89D-A239 through the Systems Research Center at VPI&SU. The authors acknowledge stimulating discussions with James D. Arthur which contributed to the research described herein.

BIBLIOGRAPHY

Ackerman, A.F., L.S. Buchwald and F.H. Lewski (1989), "Software Inspections: An Effective Verification Process," IEEE Software 6, 3 (May), 31-36.

Agresti, W.W. and W.M. Evanco (1992), "Projecting Software Defects from Analyzing Ada Designs," *IEEE Transactions on Software Engineering 18,* 11 (Nov.), 988-997.

Andersen, O. (1990), "Use of Software Engineering Data in Support of Project Management," *Software Engineering Journal 5,* 6 (Nov.), 350-356.

Anderson, G.E. (1984), "The Coordinated use of Five Performance Evaluation Methodologies," *Communications of the ACM 27,* 2 (Feb.), 119-133.

Arthur, J.D. and R.E. Nance (1987), "Developing an Automated Procedure for Evaluating Software Development Methodologies and Associated Products," Technical Report SRC-87-007, Systems Research Center, Virginia Tech, Blacksburg, VA.

Arthur, J.D. and R.E. Nance (1990), "A Framework for Assessing the Adequacy and Effectiveness of Software Development Methodologies," In *Proceedings of the Fifteenth Annual Software Engineering Workshop*, Greenbelt, MD.

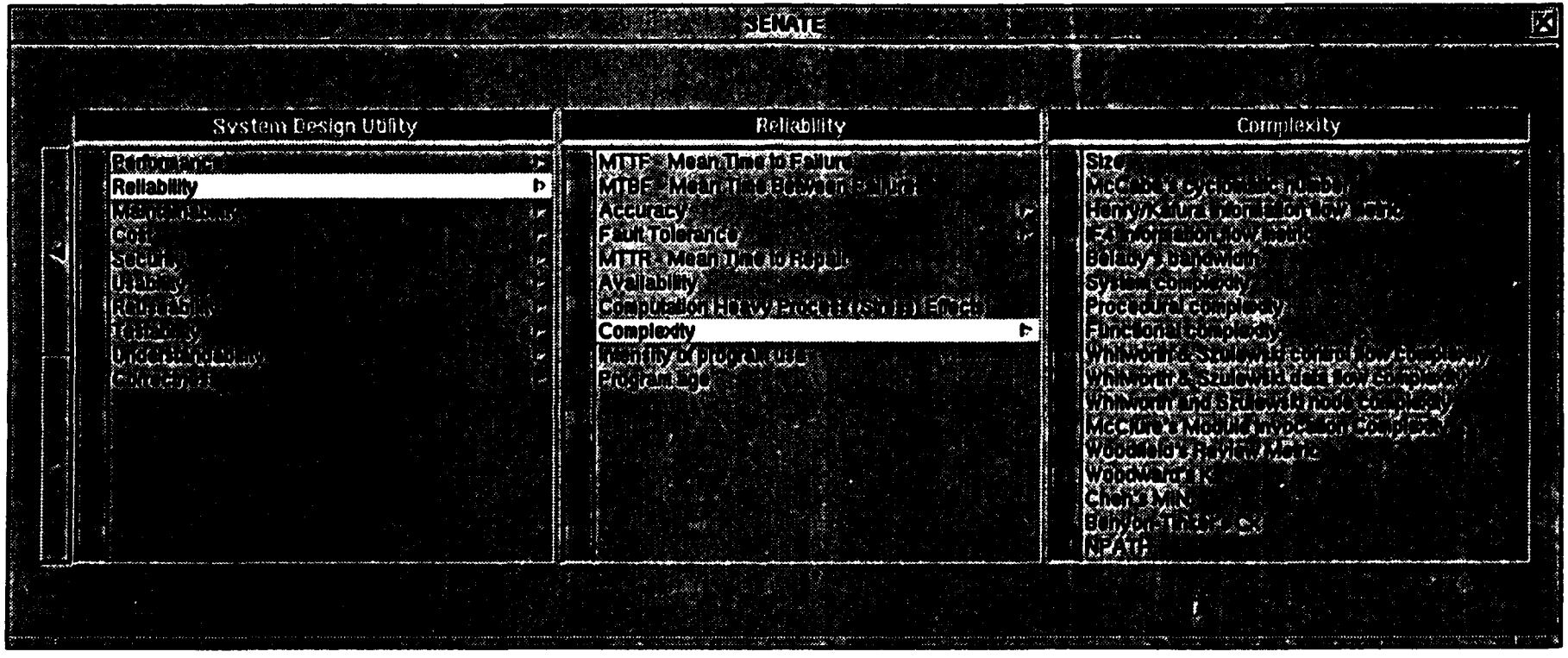


Figure 6. SENATE Browser

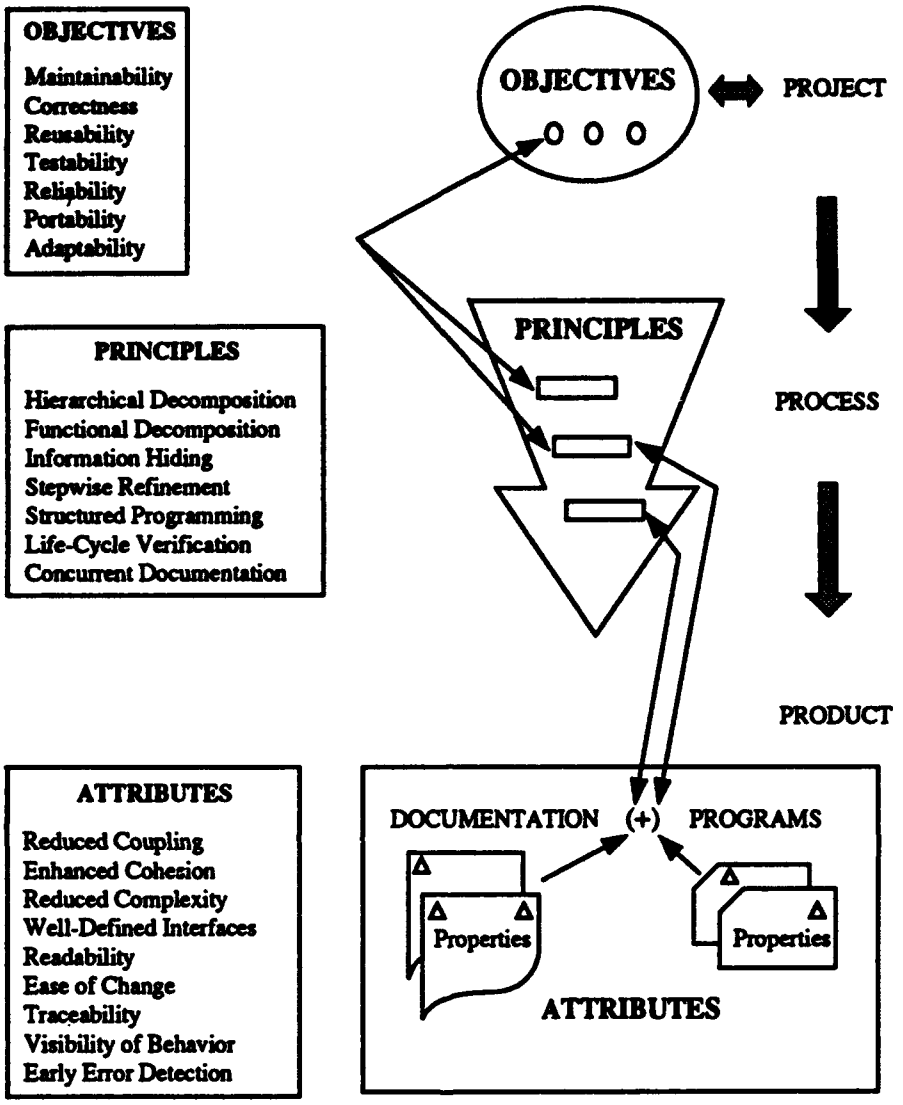


Figure 7. Illustration of the Relationship Among Objectives, Principles, Attributes in the Software Development Process

Arthur, J.D. and R.E. Nance, and O. Balci (1993), "Establishing Software Development Process Control: Technical Objectives, Operational Requirements, and the Foundational Framework," *The Journal of Systems and Software*, to appear.

Bache, R. and R. Tinker (1988), "A Rigorous Approach to Metrification: A Field Trial Using Kindra," In *Software Engineering 88 Second IEE/BCS Conference* (Liverpool, England, July 11-15), IEE, London, England, pp. 28-32.

Balci, O. (1986), "Requirements for Model Development Environments," *Computers & Operations Research 13*, 1 (Jan.-Feb.), 53-67.

Balci, O. (1990), "Guidelines for Successful Simulation Studies," In *Proceedings of the 1990 Winter Simulation Conference*, O. Balci, R.P. Sadowski, and R.E. Nance, Eds. IEEE, Piscataway, NJ, pp. 25-32.

Balci, O. and R.E. Nance (1987a), "Simulation Model Development Environments: A Research Prototype," *Journal of the Operational Research Society 38*, 8 (Aug.), 753-763.

Balci, O. and R.E. Nance (1987b), "Simulation Support: Prototyping the Automation-Based Paradigm," In *Proceedings of the 1987 Winter Simulation Conference*, A. Thesen, H. Grant, and W.D. Kelton, Eds. IEEE, Piscataway, NJ, pp. 495-502.

Balci, O. and R.E. Nance (1992), "The Simulation Model Development Environment: An Overview," In *Proceedings of the 1992 Winter Simulation Conference* (Arlington, VA, Dec. 13-16). IEEE, Piscataway, NJ, pp. 726-736.

Barshefsky, A. and J.L. Carter (1984), "Application of Software Metrics to Autoplex Cellular Development," In *Conference Record of IEEE Global Telecommunications Conference, GLOBECOM '84: Communications in the Information Age* (Atlanta, Georgia, Nov. 26-29), IEEE, Piscataway, New Jersey, pp. 1295-1298.

Basili, V.R. and E.E. Katz (1983), "Metrics of Interest in an ADA Development," In *IEEE Computer Society Workshop on Software Engineering Technology Transfer* (Miami Beach, Florida, April 25-27), IEEE, Piscataway. NJ, pp. 22-29.

Basili, V.R. and R.W. Selby (1985), "Calculation and Use if an Environment's Characteristic Software Metric Set," In *Proceedings of Eighth International Conference on Software Engineering* (London, England, August 28-30), IEEE, Piscataway, New Jersey, pp. 386-391.

Beane, J., N. Giddings and J. Silverman (1984), "Quantifying Software Designs," In *Proceedings - Seventh International Conference on Software Engineering* (Orlando, Fla, March 26-29), IEEE, Piscataway, NJ, pp. 314-322.

Boehm, B.W. and P.N. Papaccio (1988), "Understanding and Controlling Software Costs," *IEEE Transactions on Software Engineering 14*, 10 (Oct.), 1462-1477.

Booth, T.L. and B. Quin (1987), "Use of Performance to Guide Software Designs," In *Second International Conference on Computers and Applications* (Beijing, China, June 23-27), IEEE, Piscataway, NJ, pp. 305-311.

Bryan, W.L. and S.G. Siegel (1984), "Product Assurance: Insurance Against a Software Disaster," *Computer 17*, 4 (Apr.), 75-83.

Buckley, F.J. (1989), "Standard Set of Useful Software Metrics Is Urgently Needed," *Computer 22*, 7 (July), 88-89.

Caldiera, G. and V.R. Basili (1991), "Identifying and Qualifying Reusable Software Components," *Computer 24*, 2 (Feb.), 61-70.

Cardenas-Garcia, S. and M.V. Zelkowitz (1991), "A Management Tool for Evaluation of Software Designs," *IEEE Transactions on Software Engineering 17*, 9 (Sept.), 961-972.

Carver, D.L. (1988), "Comparison of the Effect on Development Paradigms on Increases in Complexity," *Software Engineering Journal 3*, 6 (Nov.), 223-228.

Chu, W.W., C. Sit and K.K. Leung (1991), "Task Response Time for Real-Time Distributed Systems With Resource Contentions," *IEEE Transactions on Software Engineering 17*, 10 (Oct.), 1076-1092.

Clapp, J. (1993), "Getting Started on Software Metrics," *IEEE Software 10*, 1 (Jan.), 108-110.

Conte, S.D., H.E. Dunsmore, and V.Y. Shen (1986), *Software Engineering Metrics and Models*, Benjamin Cummings Publishing, Menlo Park, CA.

Derrick, E.J. (1992), "A Visual Simulation Support Environment Based on a Multifaceted Conceptual Framework," Ph.D. Dissertation, Department of Computer Science, VPI&SU, Blacksburg, VA, Apr.

Derrick, E.J. and O. Balci (1992a), "DOMINO: A Multifaceted Conceptual Framework for Visual Simulation Modeling," Technical Report TR-92-43, Department of Computer Science, VPI&SU, Blacksburg, VA, Aug.

Derrick, E.J. and O. Balci (1992b), "A Visual Simulation Support Environment Based on the DOMINO Conceptual Framework," Technical Report TR-92-44, Department of Computer Science, VPI&SU, Blacksburg, VA, Aug.

Derrick, E.J. and O. Balci (1992c), "A Visual Simulation Model Specification Language," (in preparation).

Deutsch, M.S. (1988), "Focusing Real-Time Systems Analysis on User Operations," *IEEE Software 5*, 5 (Sept.), 39-50.

Emerson, T.J. (1984), "A Discriminant Metric for Module Cohesion," In *Proceedings - Seventh Intenational Conference on Software Engineering* (Orlando, Fla, March 26-29), IEEE, Piscataway, NJ, pp. 294-303.

Emery, K.D. and B.K. Mitchell (1989), "Multi-level Software Testing Based on Cyclomatic Complexity," In *Proceedings of the IEEE 1989 National Aerospace and Electronics Conference - NAECON 1989* (Dayton, Ohio, May 22-26), , Piscataway, NJ, pp. 500-507.

Far, W.H. and A. Ashton (1992), "Developing a Metrics Assessment Program for the SLBM Software Development Division," In *Proceedings of the 1992 Complex Systems Engineering Synthesis and Assessment Technology Workshop*, NSWC, Silver Spring, MD, pp. 139-146.

Fenick, S. (1990), "Implementing Management Metrics: An Army Program," *IEEE Software 7*, 2 (Mar.), 65-72.

Fenton, N.E. and A.A. Kaposi (1987), "Metrics and Software Structure," *Information and Software Technology 29*, 6 (Jul.-Aug.), 301-320.

Freedman, R.S. (1991), "Testability of Software Components," *IEEE Transactions on Software Engineering 17*, 6 (June), 553-564.

Geist, R. and K. Trivedi (1990), "Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques," *Computer 23*, 7 (July), 52-62.

Gibson, V.R. and J.A. Senn (1989), "System Structure and Software Maintenance Performance," *Communications of the ACM 32*, 3 (Mar.), 347-358.

Gilb, T. (1977), *Software Metrics*, Winthrop Publishers, Cambridge, MA.

Gilb, T. (1985), "Software Specification and Design Must 'Engineer' Quality and Cost Iteratively", In *Third International Workshop on Software Specification and Design* (London, England, August 26-27), IEEE, London, England, pp. 75-76.

Gill, G.K. and C.F. Kemerer (1991), "Cyclomatic Complexity Density and Software Maintenance Productivity," *IEEE Transactions on Software Engineering 17*, 12 (Dec.), 1284-1288.

Gould, J.D., S.J. Boies, and C. Lewis (1991), "Making Usable, Useful, Productivity. Enhancing Computer Applications," *Communications of the ACM 34*, 1 (Jan.), 75-85.

Grady, R.B. (1987), "Measuring and Managing Software Maintenance," *IEEE Software 4*, 5 (Sept.), 35-45.

Grady, R.B. (1990), "Work-Product Analysis: The Philosopher's Stone of Software?," *IEEE Software 7*, 2 (Mar.), 26-34.

Grady, R.B. and D.L. Caswell (1987), *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, Englewood Cliffs, NJ.

Gremillion, L.L. (1984), "Determinants of Program Repair Maintenance Requirements," *Communications of the ACM 27*, 8 (Aug.), 826-832.

Hall, D.L., J.J. Gibbons and D.A. Woodle (1985), "Avoid Disaster: The Use of an Integrated Tool for Managing Throughput and Response Time Requirements in Embedded Real-Time Systems," In *Conference on Software Tools* (New York, NY, April 15-17), IEEE, Piscataway, NJ, pp. 106-111.

Han, W., Y. Choe and Y. Park (1987), "Software Metrics Using Operand Types," In *Proceedings - TENCON 87: 1987 IEEE Region 10 Conference, 'Computers and Communications Technology Towards 2000'* (Piscataway, New Jersey, August 25-28), IEEE, Seoul, South Korea, pp. 1212-1215.

Heitkoeten, U. (1990), "Design Metrics and Their Aid to Automatic Collection," *Information and Software Technology 32*, 1 (Jan.-Feb.), 79-87.

Henry, S. and C. Selig (1990), "Predicting Source-Code Complexity at the Design Stage," *IEEE Software 7*, 2 (Mar.), 36-43.

Henry, S. and D. Kafura (1984), "The Evaluation of Systems Structure Using Quantitative Software Metrics," *Software Practice and Experience 14*, 6 (June), 561-573.

Henry, S. and R. Goff (1989), "Complexity Measurement of a Graphical Programming Language," *Software- Practice and Experience 9*, 4 (Nov.), 1065-1088.

Henry, S. and R. Goff (1991), "Comparison of a Graphical and a Textual Design Language Using Software Quality Metrics," *Journal of Systems and Software 14*, 3 (Mar.), 133-144.

Herndon, M.A. and J.A. McCall (1983), "The Requirements Management Methodology: A Measurement Framework for Total Systems Reliability," In *Total Systems Reliability Symposium* (Gaithersburg, MD, Dec. 12-14), IEEE, Piscataway, NJ, pp. 119-122.

Hirayama, M. (1990), "Practice of Quality Modeling and Measurement on Software Life-Cycle," In *12th International Conference on Software Engineering* (Nice, France, Mar. 26-30), IEEE, Piscataway, NJ, pp. 98-107.

Hoffman, G.D. (1989), "Early Introduction of Software Metrics," In *Proceedings of the IEEE 1989 National Aerospace and Electronics Conference - NAECON 1989* (Dayton, Ohio, May 22-26), IEEE, Piscataway, NJ, pp. 559-563.

Ince, D.C. and S. Hekmatpour (1988), "An Approach to Automated Software Design Based on Product Metrics," *Software Engineering Journal 3*, 2 (Mar.), 53-56.

Ince, D.O. and M.J. Sheppard (1988), "System Design Metrics: A review and perspective," In *Software Engineering 88 Second IEE/BCS Conference* (Liverpool, England, July 11-15), IEE, London, England, pp. 23-27.

Joshi, S.M. and K.B. Misra (1991), "Quantitative Analysis of Software Quality During the 'Design and Implementation' Phase," *Microelectronics and Reliability 31*, 5, 879-884.

Karolak, D.K. (1985), "Identifying Software Quality Metrics for a Large Software Development," In *GLOBECOM '85: IEEE Global Telecommunications Conference Record* (New Orleans, LA, Dec. 2-5), IEEE, Piscataway, NJ, pp. 61-64.

Karunanithi, N., D. Whitley, and Y.K. Malaiya (1992), "Predictability of Software Reliability Using Connectionist Models," *IEEE Transactions on Software Engineering 18*, 7 (July), 563-574.

Kavinde, T.M. (1989), "Performance Analysis of Software: CDOT case study," In *TENCON '89: Fourth IEEE Region 10 International Conference* (Bombay, India, Nov. 22-24), IEEE, Piscataway, New Jersey, pp. 718-721.

Kearney, J.K., R.L. Sedlmeyer, W.B. Thompson, M.A. Gray and M.A. Adler (1986), "Software Complexity Measurement," *Communications of the ACM 29*, 11 (Nov.), 1044-1050.

Kemerer, C.F. (1987), "An Empirical Validation of Software Cost Estimation Models," *Communications of the ACM 30*, 5 (May), 416-429.

Kemerer, C.F. (1993), "Reliability of Function Points Measurement," *Communications of the ACM 36*, 2 (Feb.), 85-97.

Kemerer, C.F. and B.S. Porter (1992), "Improving the Reliability of Function Point Measurement: An Empirical Study," *IEEE Transactions on Software Engineering 18*, 11 (Nov.), 1011-1024.

Khoshgoftaar, T.M., J.C. Munson, B.B. Bhattacharaya and G.D. Richardson (1992), "Predictive Modeling Techniques of Software Quality from Software Measures," *IEEE Transactions on Software Engineering 18*, 11 (Nov.), 979-987.

Kitchenham, B.A. (1988), "An Evaluation of Software Structure Metrics," In *Proceedings of the Twelfth Annual International Computer Software and Applications Conference (COMPSAC 88)* (Chicago, Illinois, Oct. 5-7), IEEE, Piscataway, NJ, pp. 369-376.

Kitchenham, B.A. and J.A. McDermid (1986), "Software Metrics and Integrated Support Environments," *Software Engineering 1*, 1 (Jan.), 58-64.

Kitchenham, B.A. and S.J. Linkman (1990), "Design Metrics in Practice," *Information Software and Technology 32*, 4 (May), 304-310.

Kitchenham, B.A., L.M. Pickard, and S.J. Linkman (1990), "Evaluation of some design metrics," *Software Engineering Journal 9*, 1 (Jan.), 50-58.

Lakshmanan, K.B., S. Jayaprakash and P.K. Sinha (1991), "Properties of Control-Flow Complexity Measures," *IEEE Transactions on Software Engineering 17*, 12 (Dec.), 1289-1295.

Laranjeira, L.A. (1990), "Software Size Estimation of Object-Oriented Systems," *IEEE Transactions on Software Engineering 16*, 5 (May), 510-522.

Lew, K.S., T.S. Dillon and K.E. Forward (1988), "Software Complexity and Its Impact on Software Reliability," *IEEE Transactions on Software Engineering 14*, 11 (Nov.), 1645-1655.

Litke, J. (1992), "A Method for the Assessment of System Designs," In *Proceedings of the 1992 Complex Systems Engineering Synthesis and Assessment Technology Workshop*, NSWC, Silver Spring, MD, pp. 155-169.

Low, G.C. and D.R. Jeffrey (1990), "Function Points in the Estimation and Evaluation of the Software Process," *IEEE Transactions on Software Engineering 16*, 1 (Jan.), 64-71.

MacKnight, C.B. and S. Balagopalan (1989), "An Evaluation Tool for Measuring Authoring System Performance," *Communications of the ACM 32*, 10 (Oct.), 1231-1236.

McCabe, T.J. and C.W. Butler (1989), "Design Complexity Measurement and Testing," *Communications of the ACM 32*, 12 (Dec.), 1415-1425.

McCabe, T.J., L.F. Young, K.W. Claybaugh and J. McManus (1983), "Design Basis Paths: A Complexity Driven Design Inspection Methodology," In *Total Systems Reliability Symposium* (Gaithersburg, MD, Dec. 12-14), IEEE, Piscataway, NJ, pp. 67-72.

Mohanty, S.N. (1981), "Entropy Metrics for Design Evaluation," *Journal of Systems and Software 2*, 1 (Feb.), 39-46.

Mukhopadhyay, T. and S. Kekre (1992), "Software Effort Models for Early Estimation of Process Control Applications," *IEEE Transactions on Software Engineering 18*, 10 (Oct.), 915-924.

Munson, J. and T.M. Khoshgoftaar (1992), "The Detection of Fault-Prone Programs," *IEEE Transactions on Software Engineering 18*, 5 (May), 423-433.

Munson, J.C. and T.M. Khoshgoftaar (1992), "Measuring Dynamic Program Complexity," *IEEE Software 9*, 6 (Nov.), 48-55.

Musa, J.D. and A.F. Ackerman (1989), "Quantifying Software Validation: When to Stop Testing?," *IEEE Software 6*, 3 (May), 19-27.

Nance, R.E. and J.D. Arthur (1988), "The Methodology Roles in the Realization of a Model Development Environment," In *Proceedings of the 1988 Winter Simulation Conference*, pp. 220-225.

Navlakha, J.K. (1987), "A Survey of System Complexity Metrics," *Computer Journal 30*, 3 (June), 233-238.

Nejmeh, B.A. (1988), "NPATH: A Measure of Execution Path Complexity and its Applications," *Communications of the ACM 31*, 2 (Feb.), 188-200.

Nguyen, C.M. and S.L. Howell (1992), "System Design Factors," In *Proceedings of the 1992 Complex Systems Engineering Synthesis and Assessment Technology Workshop*, NSWC, Silver Spring, MD, pp. 147-154.

Parnas, D.L., J.v. Schouwen, and S.P. Kwan (1990), "Evaluation of Safety-Critical Software," *Communications of the ACM 33*, 6 (June), 636-648.

Paulish, D.J. (1990), "Methods and Metrics for Developing High Quality Patient Monitoring System Software," In *Proceedings of the Third Annual IEEE Symposium on Computer-Based Medical Systems* (Chapel Hill, NC, Jun 3-6), IEEE, Piscataway, NJ, pp. 145-152.

Pollock, G.M. and S. Sheppard (1987), "A Design Methodology for the Utilization of Metrics Within Various Phases of Software Life-cycle Models," In *Proceedings 11 - COMPSAC 87: The Eleventh Annual International Computer Software and Applications Conference* (Tokyo, Japan, Oct. 7-9), IEEE, Piscataway, NJ, pp. 221-230.

Porter, A.A. and R.W. Selby (1990), "Empirically Guided Software Development Using Metric-Based Classification Trees," *IEEE Software 7*, 2 (Mar.), 46-54.

Ramamoorthy, C.V., A. Bhide and V. Garg (1986), "Software Quality and Requirements Specification," In *Proceedings - IEEE Computer Society 1986 International Conference on Computer Languages* (Miami, Florida, Oct. 27-30), IEEE, Piscataway, NJ, pp. 75-83.

Ramamoorthy, C.V., W. Tsai and Y. Usuda (1984), "Software Engineering: Problems and Perspectives," *Computer 17*, 10 (Oct.), 191-207.

Ramamoorthy, C.V., W. Tsai, T. Yamaura, and A. Bhide (1985), "Metrics Guided Methodology," In *Proceedings - COMPSAC 85: The IEEE Computer Society's Ninth International Computer Software and Applications Conference* (Chicago, Illinois, Oct. 9-11), IEEE, Piscataway, New Jersey, pp. 111-120.

Ramamurthy, N. and A. Melton (1988), "A Synthesis of Software Science Measures and the Cyclomatic Number," *IEEE Transactions on Software Engineering 14*, 8 (Aug.), 1116-1121.

Reibman, A.L. and M. Veeraraghavan (1991), "Reliability Modeling: An Overview for Systems Designers," *Computer 24*, 4 (Apr.), 49-56.

Reynolds, R.G. (1987), "Metric-Based Readoning About Psuedocode Design in the Partial Metrics System," *Information and Software Technology 29*, 9 (Nov.), 497-502.

Reynolds, R.G. (1987), "The Partial Metrics System: Modeling the Stepwise Refinement Process Using Partial Metrics," *Communications of the ACM 30*, 11 (Nov.), 956-963.

Reynolds, R.G. (1990), "Partial Metrics System: A Tool to Support the Metrics-Driven Design of Psuedocode Programs," *Journal of Systems and Software 9*, 4 (Jan.), 287-295.

Rombach, H.D. (1990), "Design Measurement: Some Lessons Learned," *IEEE Software 7*, 2 (Mar.), 17-25.

Schneidewind, N.F. (1979), "Software Metrics for Aiding Program Development and Debugging," In *AFIPS Conference Proceedings Vol. 48* (New York, NY, June 4-7), AFIPS Press, Montvale, NJ, pp. 989-994.

Schneidewind, N.F. (1992), "Methodology for Validating Software Metrics," In *Proceedings of the 1992 Complex Systems Engineering Synthesis and Assessment Technology Workshop*, NSWC, Silver Spring, MD, pp. 171-198.

Selby, R.W. (1990), "Extensible Integration Frameworks for Measurement," *IEEE Software 7*, 6 (Nov.), 83-84.

Shapperd, M. (1990), "Design Metrics: An Empirical Analysis," *Software Engineering Journal 5*, 1 (Jan.), 3-10.

Sheppard, M. (1990), "Early Life-cycle Metrics and Software quality models," *Information and Software Technology 32*, 4 (May), 311-316.

Shepperd, M. (1988), "An Evaluation of Software Product metrics," *Information and Software Technology 30*, 3 (Apr.), 177-288.

Shepperd, M. and D. Ince (1989), "Metrics, Outlier Analysis and the Software Design Process," *Information and Software Technology 31*, 2 (Mar.), 91-98.

Shepperd, M. and D. Ince (1990), "The Use of Metrics for the Early Detection of Design Errors," In *SE90: Proceedings of Software Engineering 90* (Brighton, UK, July 24-27), Cambridge University Press, Cambridge, UK, pp. 67-88.

Silverman, J., N. Giddings, and J. Beane (1983), "An Approach to Design-for-Maintenance," In *Record-Software Maintenance Workshop* (Monterey, OA, Dec. 6-8), IEEE, Piscataway, NJ, pp. 106-110.

Smith, C. and J.C. Browne (1980), "Aspects of Software Design Analysis: Concurrency and Blocking," *Performance Evaluation Review 9*, 2 (Summer), 245-253.

Symons, C.R. (1988), "Function Point Analysis: Difficulties and Improvements," *IEEE Transactions on Software Engineering 14*, 1 (Jan.), 2-12.

Troy, D.A. and S.H. Zweben (1981), "Measuring the Quality of Structured Designs," *Journal of Systems and Software 2*, 2 (June), 113-120.

Velez, C.E. and P.A. Scheffer (1978), "On the Problem of Software Design and Measuring Quality," In *IEEE Proceedings of the National Aerospace and Electronics Conference NAECON '78* (Dayton, Ohio, May 16-18), IEEE, Piscataway, New Jersey, pp. 223-229.

Verner, J. and G. Tate (1992), "A Software Size Model," *IEEE Transactions on Software Engineering 18*, 4 (Apr.), 265-278.

Vessey, I. and R. Weber (1983), "Some Factors Affecting Program Repair Maintenance," *Communications of the ACM 26*, 2 (Feb.), 128-136.

Vienneau, R.L. (1992), "The Consolidated Experience Factory: An Approach for Instrumenting Systems Engineering," In *Proceedings of the 1992 Complex Systems Engineering Synthesis and Assessment Technology Workshop*, NSWC, Silver Spring, MD, pp. 201-206.

Walters, G.F. and J.A. McCall (1979), "Software Quality Metrics for Life-Cycle Cost-Reduction," *IEEE Transactions on Reliability R-28*, 3 (Aug.), 212-220.

Weyuker, E.J. (1988), "Evaluating Software Complexity Measures," *IEEE Transactions on Software Engineering 14*, 9 (Sept.), 1357-1365.

Whitworth, M.H. and P.A. Szulewski (1980), "The Measurement of Control and Data Flow Complexity in Software Designs," In *IEEE Computer Society International Computer Software Applications Conference 4th COMPSAC 80* (Chicago, Illinois, Oct. 27-31), IEEE, Piscataway, New Jersey, pp. 735-743.

Wohlin, C. and D. Rapp (1989), "Performance Analysis in the Early Design of Software," In *Seventh International Conference on Software Engineering for Telecommunications Switching Systems* (Bournemouth, England, July 3-6), IEE, London, England, pp. 114-121.

OPTIMAL SELECTION OF FAILURE DATA FOR PREDICTING FAILURE COUNTS

Norman F. Schneidewind

Code AS/Ss
Naval Postgraduate School
Monterey, CA 93943
(408) 656-2719/2471
FAX: (408) 656-3407
Internet: 0442p.@vm1.cc.nps.navy.mil

Abstract

In the use of software reliability models it is not necessarily the case that all the failure data should be used to estimate model parameters and to predict failures. The reason for this is that old data may not be as representative of the current and future failure process as recent data. Therefore it may be possible to obtain more accurate predictions of future failures by excluding or giving lower weight to the earlier failure counts. Although techniques such as moving average and exponential smoothing are frequently used in other fields, such as inventory control, we did not find use of this idea in the various models we surveyed. One model that includes the concept of selecting a subset of the failure data, where appropriate, is the Schneidewind Non-Homogeneous Poisson Process (NHPP) software reliability model. In order to use the concept of "data aging", there must be a criterion for determining the optimal value of the starting failure count interval. In previous research we identified the mean square error as the best criterion for selecting the starting interval of the failure data. In this paper we apply the criterion to select the optimal starting interval. We show that significantly improved reliability predictions can be obtained by using a subset of the failure data, based on applying the criterion, and using the Space Shuttle On-Board software as an example.

Keywords: NHPP software reliability model, optimal selection of failure data, Space Shuttle.

INTRODUCTION

In the use of software reliability models it is not necessarily the case that all the failure data should be used to estimate model parameters and to predict failures. The reason for this is that old data may not be as representative of the current and future failure process as recent data. If the failure process remains the same over a long series of observations, we should use a great deal (or all) of the failure data; if there is a significant change in the process, we should use only the most recent observations [BRO 63]. Therefore it may be possible to obtain more accurate predictions of future failures by excluding or giving lower weight to the earlier failure counts. Although techniques such as moving average and exponential smoothing are frequently used in other fields, such as inventory control, we did not find mention of this idea in the many models we examined in various papers and reports that contain surveys of models [AIA 91, ABD 86, FAR 91, FAR 83, GOE 85, LIT 80]. One model that includes the concept of

selecting a subset of the failure data is the Schneidewind Non-Homogeneous Poisson Process (NHPP) software reliability model [SCH 75, XIE 92]. In order to use the concept of data aging (i.e., giving more weight to recent failure counts), there must be a criterion for determining the optimal value of s, an index in the range $1 \leq s \leq t$, which is the starting value of equal length failure count intervals. In this model one may choose to use all the failure counts in the execution intervals from 1 to t (Method 1), exclude counts from 1 to s-1 (Method 2), or use an aggregate count from 1 to s-1 and individual counts from s to t (Method 3).

Importance of Research

The importance of this research is that significant improvements were obtained in the accuracy of predicting failure count and time to next failure (due to space limitations, only the failure count analysis is presented in this paper) by *not using all the observed failure data*, where appropriate, as we will illustrate in the examples. Also of significance is the identification of a criterion for determining "where appropriate"; that is, the method for determining the optimal value of s, s^*, where "optimal" is defined as the value of s that produces the most accurate predictions. This research was conducted on the Schneidewind model and the criterion was applied to the Space Shuttle On-Board flight software. Since this model is used to assist IBM-Houston in making software reliability predictions for the Space Shuttle software, we were motivated to find a generic method for optimal failure data selection and to apply this method to obtain the most accurate predictions possible for the Space Shuttle [SCH 92, AIA 92]. The concepts developed here have general applicability to other models but in order to realize the advantages of optimal data selection, it would be necessary to modify the parameter estimation methods used in those models to explicitly allow for subsets of the failure data to be used.

The purpose of our research is to demonstrate the effectiveness of the Mean Square Error (MSE) criterion, which we identified as the best of four criteria which were developed and analyzed in previous research [SCH 93], for selecting s^*. We demonstrate that, when conditions warrant, $s^*>1$ can produce more accurate failure predictions than s=1 for the Space Shuttle software.

Before discussing the criterion for selecting s^*, we provide an overview of the Schneidewind model parameter estimation in order to establish the rationale for data aging. As a by-product of this analysis we show that, for certain modules, dramatic improvements can be made in prediction accuracy by *not using all the failure counts*. We close with conclusions about the utility of the data aging approach and the criterion to use for data aging; we also indicate our future research efforts.

OVERVIEW OF SCHNEIDEWIND MODEL PARAMETER ESTIMATION

The method of maximum likelihood is used to estimate the model parameters α and β, for a given s, where α is the failure rate at t=0 and β is the failure rate time constant (i.e., a measure of how fast the

failure rate decays -- the smaller the value of β, the faster the failure rate decreases).

a. Parameter estimation: Method 1

Use all of the failure counts from interval 1 through t ($1 \le s \le t$). This method is used if it is assumed that all of the historical failure counts from 1 through t are representative of the future failure process. Equations (1) and (2) are used to estimate β and α, respectively [SCH 75, FAR 83, FAR 91].

$$\frac{1}{\exp(\beta)-1} - \frac{t}{\exp(\beta t)-1} = \sum_{k=0}^{t-1} k \frac{x_{k+1}}{X_t} \tag{1}$$

$$\alpha = \frac{\beta X_t}{1-\exp(-\beta t)} \tag{2}$$

where x_{k+1} are failure counts in $1,2,\ldots,k+1,\ldots,t$ and X_t is the cumulative failure count in 1,t.

b. Parameter estimation: Method 2

Use failure counts only in the intervals s through t ($1 \le s \le t$). This method is used if it is assumed that only the historical failure counts from s through t are representative of the future failure process. Equations (3) and (4) are used to estimate β and α, respectively [SCH 75, FAR 83, FAR 91].

$$\frac{1}{\exp(\beta)-1} - \frac{t-s+1}{\exp(\beta(t-s+1))-1} = \sum_{k=0}^{t-s} k \frac{x_{s+k}}{X_{s,t}} \tag{3}$$

$$\alpha = \frac{\beta X_{s,t}}{1-\exp(-\beta(t-s+1))} \tag{4}$$

where x_{s+k} are failure counts in $s,s+1,\ldots,s+k,\ldots,t$ and $X_{s,t}$ is the cumulative failure count in s,t. We note that Method 2 is equivalent to Method 1 for s=1 (i.e., (1) and (2) are obtained by substituting s=1 in (3) and (4), respectively).

c. Parameter estimation: Method 3

Use the cumulative failure count in the interval 1 through s-1 and individual failure counts in the intervals s through t ($2 \le s \le t$). This method is used if it is assumed that the historical cumulative failure count from 1 through s-1 and the individual failure counts from s through t are representative of the future failure process. This method is intermediate to Method 1, which uses all the data, and Method 2, which discards "old" data. Equations (5) and (6) are used to estimate β and α, respectively [SCH 75, FAR 83, FAR 91].

$$\frac{(s-1)X_{s-1}}{\exp(\beta(s-1))-1}+\frac{X_{s,t}}{\exp(\beta)-1}-\frac{tX_t}{\exp(\beta t)-1}=\sum_{k=0}^{t-s}(s+k-1)x_{s+k} \qquad (5)$$

$$\alpha=\frac{\beta X_t}{1-\exp(-\beta t)} \qquad (6)$$

where X_{s-1} is the cumulative failure count in 1,s-1. We note that Method 3 is equivalent to Method 1 for s=2 (i.e., (1) is obtained by substituting s=2 in (5)).

The three methods are summarized in Table 1 with respect to the observed parameter estimation range and the prediction range -- observed ($i \leq t$) and future ($i > t$) -- where T is the upper limit of the prediction range.

Table 1

Parameter and Prediction Ranges

Method	Parameter Range (Observed)	Prediction Range (Observed)	Prediction Range (Future)
1	s=1	$1 \leq i \leq t$	$t < i \leq T$
2	$1 \leq s \leq t$	$s \leq i \leq t$	$t < i \leq T$
3	$2 \leq s \leq t$	$1 \leq i \leq t$	$t < i \leq T$

OPTIMAL SELECTION OF FAILURE DATA USING METHOD 2

As stated, Method 2 disregards failure counts for intervals 1,..,s-1, where $1 \leq s \leq t$. In this section we apply Method 2 with respect to the MSE criterion. In all examples α and β are estimated in the range i=1-20 and failure count predictions are made in the range i=21-30, where an interval is 30 days of continuous execution of the Space Shuttle software.

Failure Prediction

Once α and β have been estimated for a given s, using one of the three methods, various predictions can be made. However, since we want to find (α^*, β^*, s^*), the computational procedure is to first use the MSE criterion, which is described in the next section, to find the optimal triple and then use it in the prediction equation. The predicted cumulative number of failures is given by (7) for Method 2. This equation is derived from (4), where $F_i - X_{s-1}$ replaces $X_{s,i}$, reflecting the fact that $X_{s,i}$ only accounts for failures in the range s,t. Failures in the range 1,s-1, which are accounted for by X_{s-1}, must be included in (7).

$$F_i(s)=(\alpha/\beta)[1-\exp(-\beta(i-s+1))]+X_{s-1} \qquad (7)$$

The equation for Methods 1 and 3 is obtained by setting s=1, using the values of α^*, β^* obtained from the respective parameter estimation methods, and setting $X_{s-1}=0$.

Mean Square Error Criterion

The Mean Square criterion for cumulative failures is to minimize (8).

$$MSE(s)=\frac{\sum_{i=s}^{t}[\alpha/\beta(1-\exp(-\beta(i-s+1)))-(X_i-X_{s-1})]^2}{t-s+1} \quad (8)$$

The rationale of MSE (mean squared difference between predicted and actual cumulative failure counts X_i-X_{s-1} in the range s,t) is to minimize the sum of the variance and the square of the bias of the predicted failure count [JEN 68]. However, since a substantial amount of computation could be involved in computing MSE for all values of s, we adopt a modified rule of using the value of s where MSE starts to increase after initially decreasing from s=1; we call this value $s^{\cdot}$ to distinguish it from s^* the value of s where MSE is minimum. This approach results in less computation and minimum discarding of "old" data. Our experience indicates that $s^{\cdot}$ will provide accurate predictions and much better ones than s=1 in those cases where s=1 is not optimal. Furthermore, we recognize that there is no assurance that s^* computed in the *parameter estimation range* will necessarily result in the minimum MSE or most accurate prediction in the *prediction range*. In fact, as will be seen, in some cases our heuristic produces better predictions than that obtained with s^*. The MSE for Methods 1 and 3 is obtained by making the adjustments described in the previous section. Equation (8) is plotted in Figure 1 for Module 1 of the Space Shuttle software for both the *parameter estimation range* and the *prediction range*. To obtain the latter, we modify (8) to use summation limits of t+1 to T and to use a denominator of T-t, where T is the upper limit of the prediction range. This figure shows $s^{\cdot}$=4 and s^*=11 in both curves.

In order to provide a measure of prediction accuracy that is independent of the MSE criterion, we compute the mean relative error (MRE) for the *prediction range*, which is given by (9) [KHO]

$$MRE=\Sigma_i(|X_i-F_i|/X_i)/(T-t). \quad (9)$$

This result is shown in Figure 2, where MRE and MSE (repeated from Figure 1) are plotted for Module 1. For MRE, we again have $s^{\cdot}$=4 and s^*=11. Now, we compare $F_i(4)$ with $F_i(1)$ in Figure 3 and see that $s^{\cdot}$=4 provides a better prediction than s=1, with the latter showing too much overshoot.

These procedures are repeated for Module 2 in Figures 4, 5 and 6 and for Module 3 in Figures 7, 8 and 9. The prediction curves in Figures 3, 6 and 9, which all show better prediction accuracy for $s^{\cdot}$ as compared to s=1 (s=2 is used for comparison for Module 2 because estimates of α and β could not be obtained for this module), dramatize the importance of using data aging, where appropriate. The analysis of the starting

interval is summarized in Table 2. When s^* is obtained from MSE for the *parameter estimation range*, it produces the "best" s for Module 2 (MRE(6) differs from MRE(7) by only .03) and Module 3, as determined by the MSE and MRE for the *prediction range* and provides better predictions than s=1 for all three modules. As the execution of the software continues for T>30, the described procedures would be repeated with t=30 (i.e., new upper limit of *parameter estimation range*).

Table 2

Analysis of Starting Interval

Module	MSE: Parameter Estimation Range	MSE: Prediction Range	MRE: Prediction Range
1	$s'=4$, $s^*=11$	$s'=4$, $s^*=11$	$s'=4$, $s^*=11$
2	$s'=7$, $s^*=7$	$s'=6$, $s^*=6$	$s'=6$, $s^*=6$
3	$s'=4$, $s^*=10$	$s'=4$, $s^*=4$	$s'=4$, $s^*=4$

SUMMARY, CONCLUSIONS AND FUTURE RESEARCH

We found that MSE does a good job of identifying s'; it has no dependence on model assumptions and it minimizes the sum of the variance and the square of the bias of the predicted failure count. We noted that once MSE reaches a minimum, as a function of s, and starts to increase, the computation can be terminated at that point because s' provides a good (better than s=1) prediction, although not necessarily the best prediction. Since the future failure process may not mirror the past, no criterion can produce the best prediction in all cases. What we can accomplish is to produce *better* predictions than would be the case in using all the data. This we have demonstrated with the examples. Since the other Space Shuttle modules have failure count distributions over execution time that are similar to the ones analyzed, we believe data aging is applicable in general to the Space Shuttle software. Our results suggest that other software reliability models could benefit from using data aging.

The next stage of our research will involve the use of Jet Propulsion Laboratory planetary mission data and Shuttle mission ground control data from the Johnson Space Center to determine whether data aging is applicable to different environments. In addition we will analyze the MSE criterion relative to the use of Method 3 and we will report our results in obtaining improved time to next failure predictions by using data aging.

DISCLAIMER

The analysis of experimental results of the intermediate software failure data in this paper should not be construed as a prediction of the final Space Shuttle software reliability. Rather, the Space Shuttle data is used as real project examples for the purpose of developing, enhancing and validating software reliability models.

ACKNOWLEDGEMENTS

We wish to acknowledge the support provided for this project by Dr. William Farr, Naval Surface Warfare Center; and Mr. Ted Keller and Mr. David Hamilton, International Business Machines Corporation.

REFERENCES

[AIA 92] Recommended Practice for Software Reliability, R-013-1992 (Draft), American Institute of Aeronautics and Astronautics, April 1992.

[ABD 86] A. A. Abdel-Ghaly, et al., "Evaluation of Competing Software Reliability Predictions", IEEE Transactions on Software Engineering, Vol. SE-12, No. 9, September 1986, pp. 950-967.

[BRO 63] Robert Goodell Brown, Smoothing, Forecasting and Prediction of Discrete Time Series, Prentice-Hall, Inc., 1963.

[FAR 83] William H. Farr, A survey of Software Reliability Modeling and Estimation, NSWC TR 82-171, Naval Surface Weapons Center, September 1983.

[FAR 91] William H. Farr and Oliver D. Smith, Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide, NAVSWC TR-84-373, Revision 2, Naval Surface Weapons Center, March 1991.

[ibid] Library Access Guide, NAVSWC TR-84-371.

[GOE 85] Amrit L. Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability, IEEE Transactions on Software Engineering, Vol. SE-11, No. 12, December 1985, pp. 1411-1423.

[JEN 68] Gwilym M. Jenkins and Donald G. Watts, Spectral Analysis and its Applications, Holden-Day, 1968.

[KHO 92] Taghi M. Khoshgoftarr, et al., "Predictive Modeling Techniques of Software Quality from Software Measures", Transactions on Software Engineering, Vol. 18, No. 11, November 1992, pp. 979-987.

[LIT 80] B. Littlewood, "Theories of Software Reliability: How Good Are They and How Can They Be Improved", Transactions on Software Engineering, Vol. SE-6, No. 5, September 1980, pp. 489-500.

[SCH 75] Norman F. Schneidewind, "Analysis of Error Processes in Computer Software", Proceedings of the International Conference on Reliable Software, IEEE Computer Society, 21-23 April 1975, pp. 337-346.

[ibid] Sigplan Notices, Volume 10, Number 6, 1975.

[SCH 92] Norman F. Schneidewind and T.W. Keller, "Application of Reliability Models to the Space Shuttle", IEEE Software, July 1992 pp. 28-33.

[SCH 93] Norman F. Schneidewind, "A Software Reliability Model with Optimal Selection of Failure Data", Proceedings of the Fifth Annual Oregon Workshop on Software Metrics, Silver Falls, Oregon, March 21-23, 1993.

[XIE 92] M. Xie and M. Zhao, "The Schneidewind Software Reliability Model Revisited", Third International Symposium on Software Reliability Engineering, IEEE Computer Society Press, October 1992, pp. 184-192.

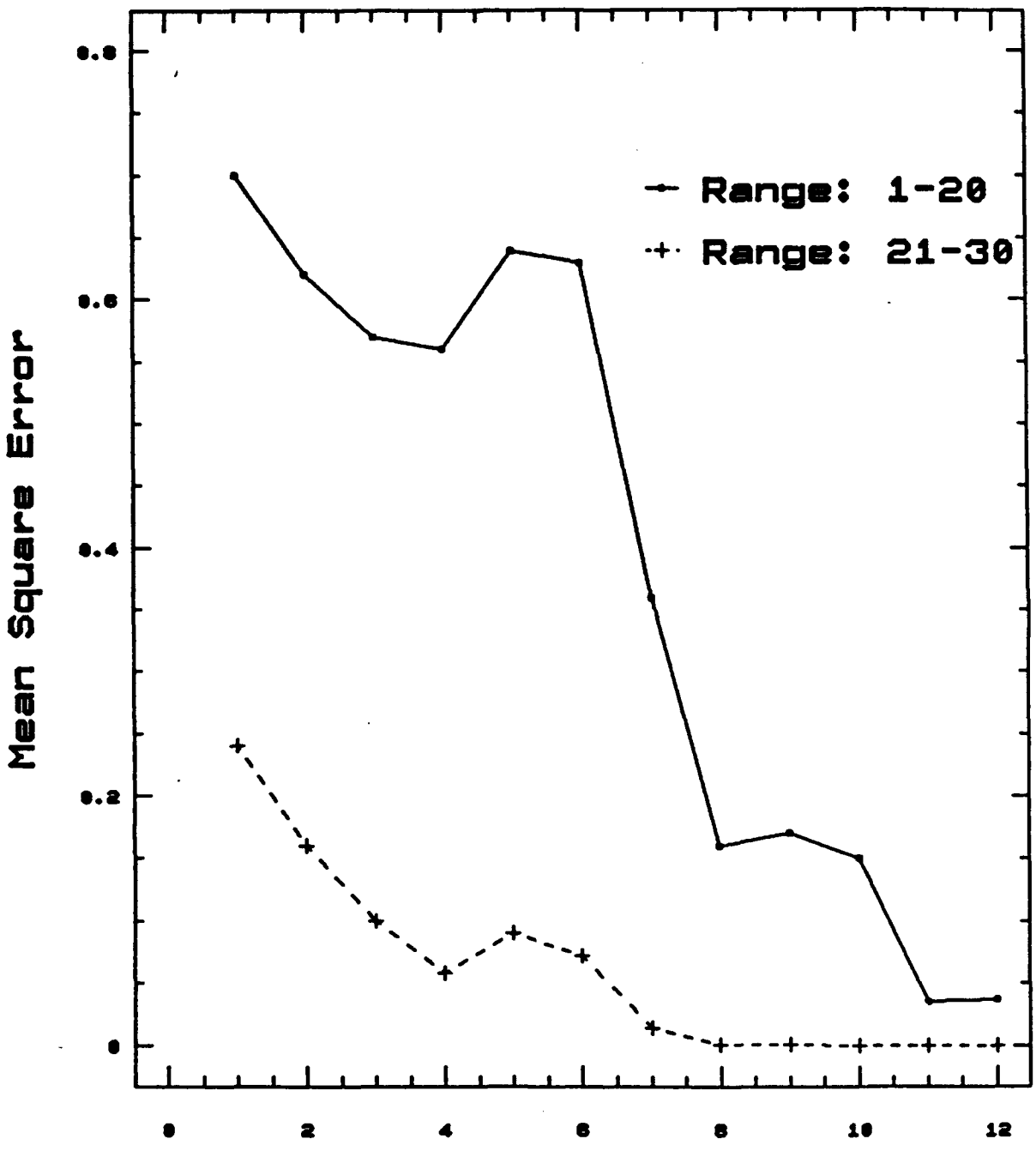


Figure 1 ... s (Starting Interval)
Method 2. Module 1

Mean Square Error (MSE) & Mean Relative Error (MRE) in Prediction Range 21-30

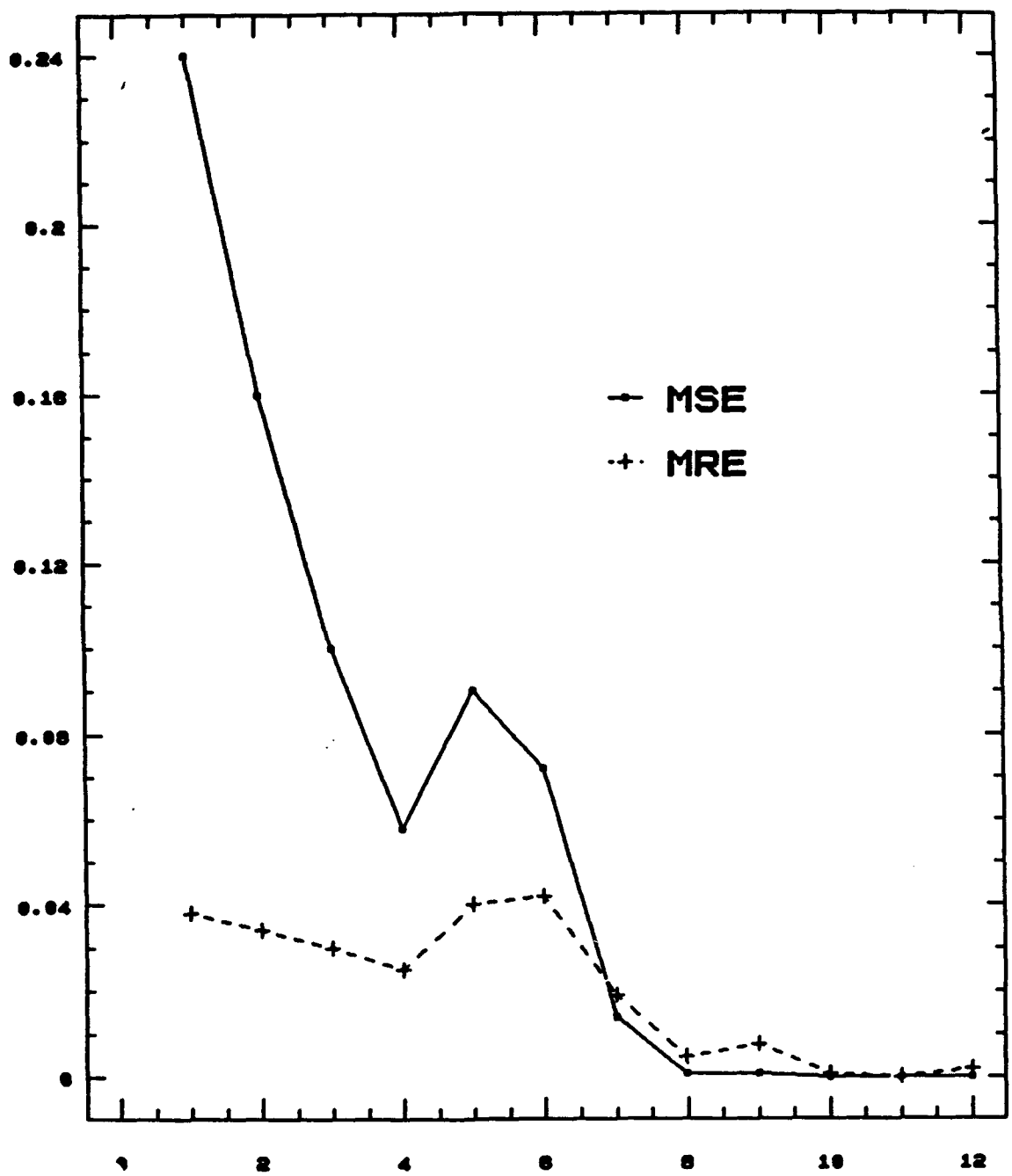


Figure 2 ... s (Starting Interval) Method 2. Module 1.

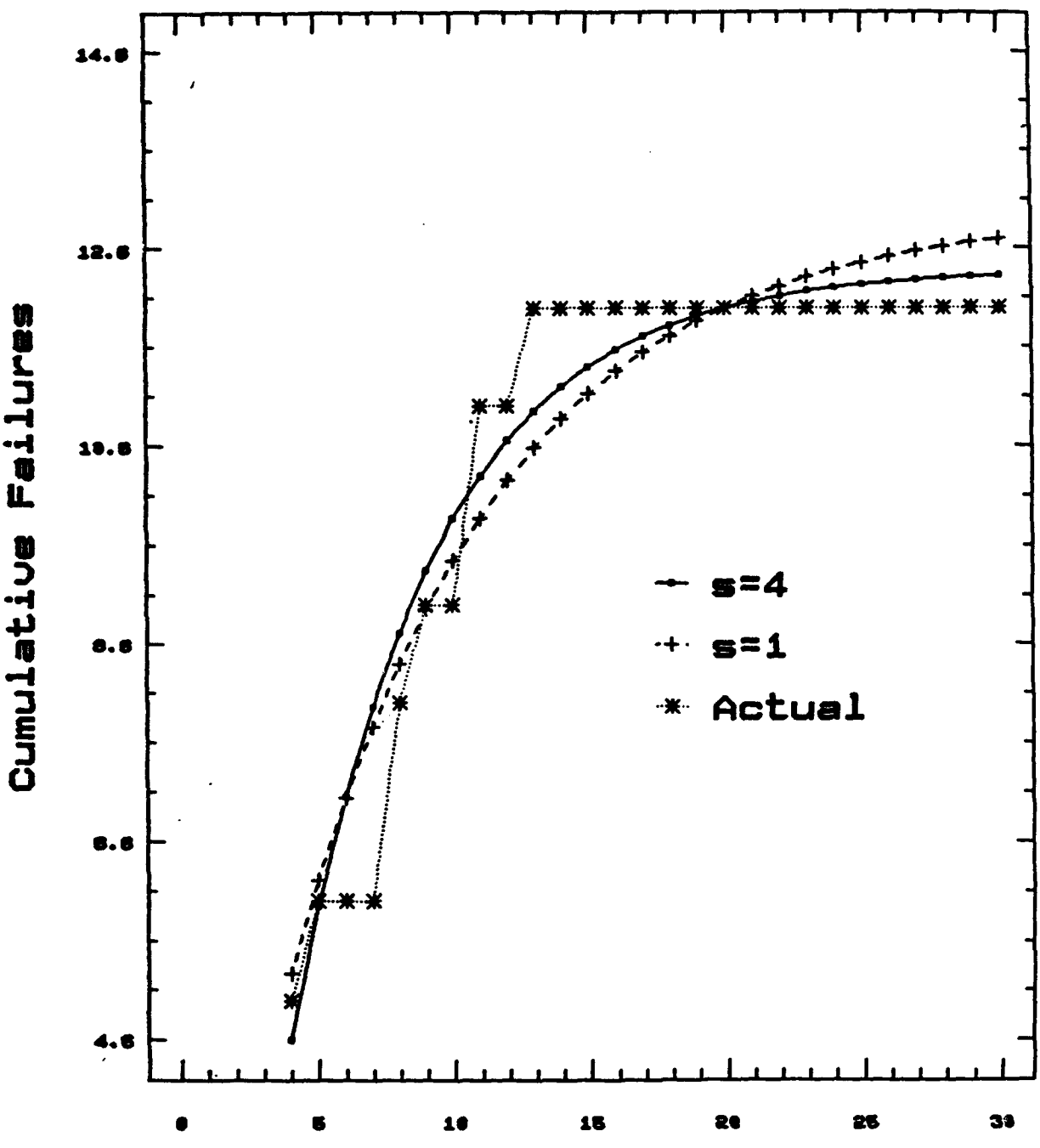


Figure 3 ... Execution Time (Intervals) Method 2. Module 1.

Mean Square Error: Parameter Estimation
Range(1-20) and Prediction Range(21-30)

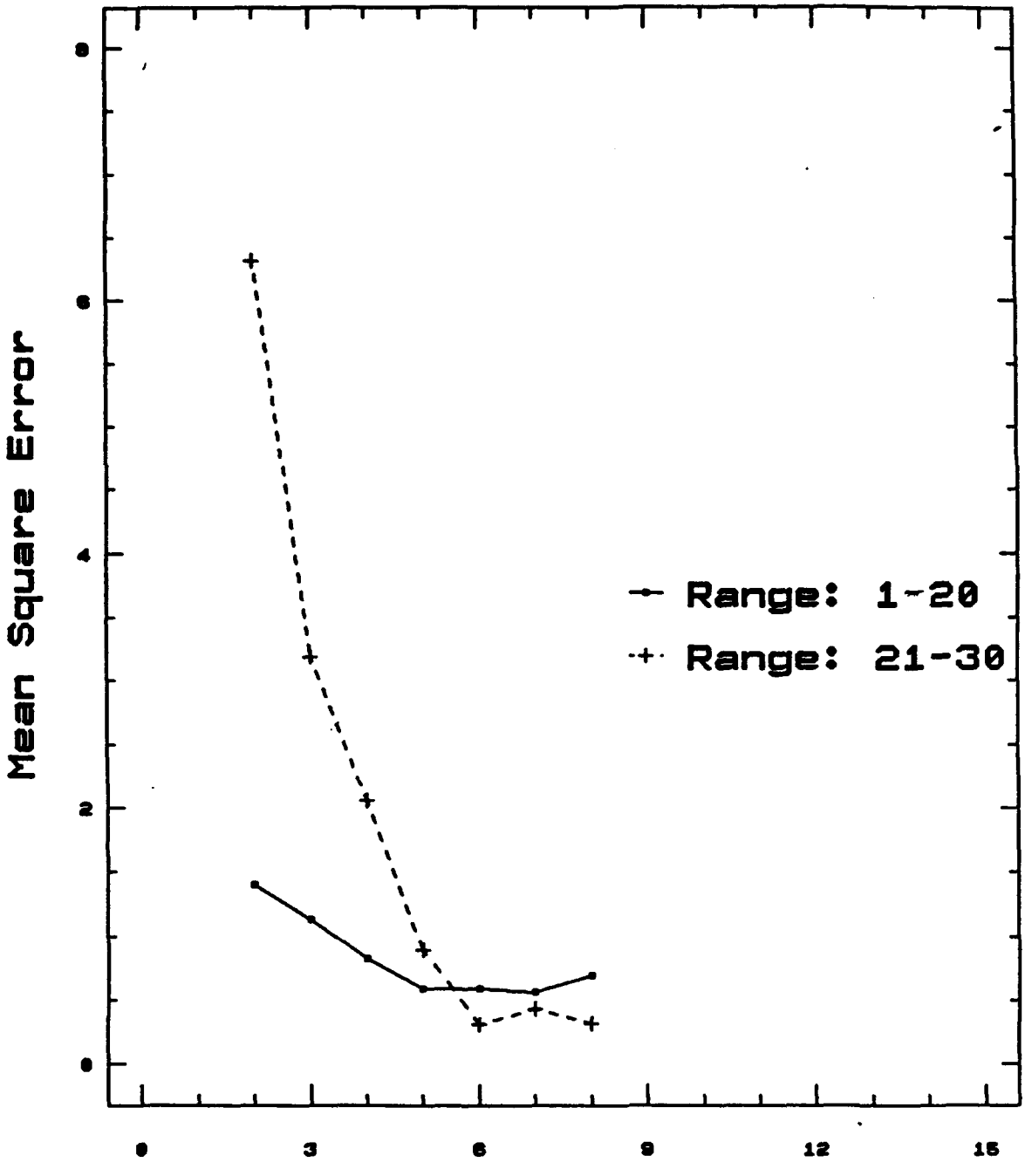


Figure 4 ... s (Starting Interval)
Method 2. Module 2.

Mean Square Error (MSE) & Mean Relative Error (MRE) in Prediction Range 21-30

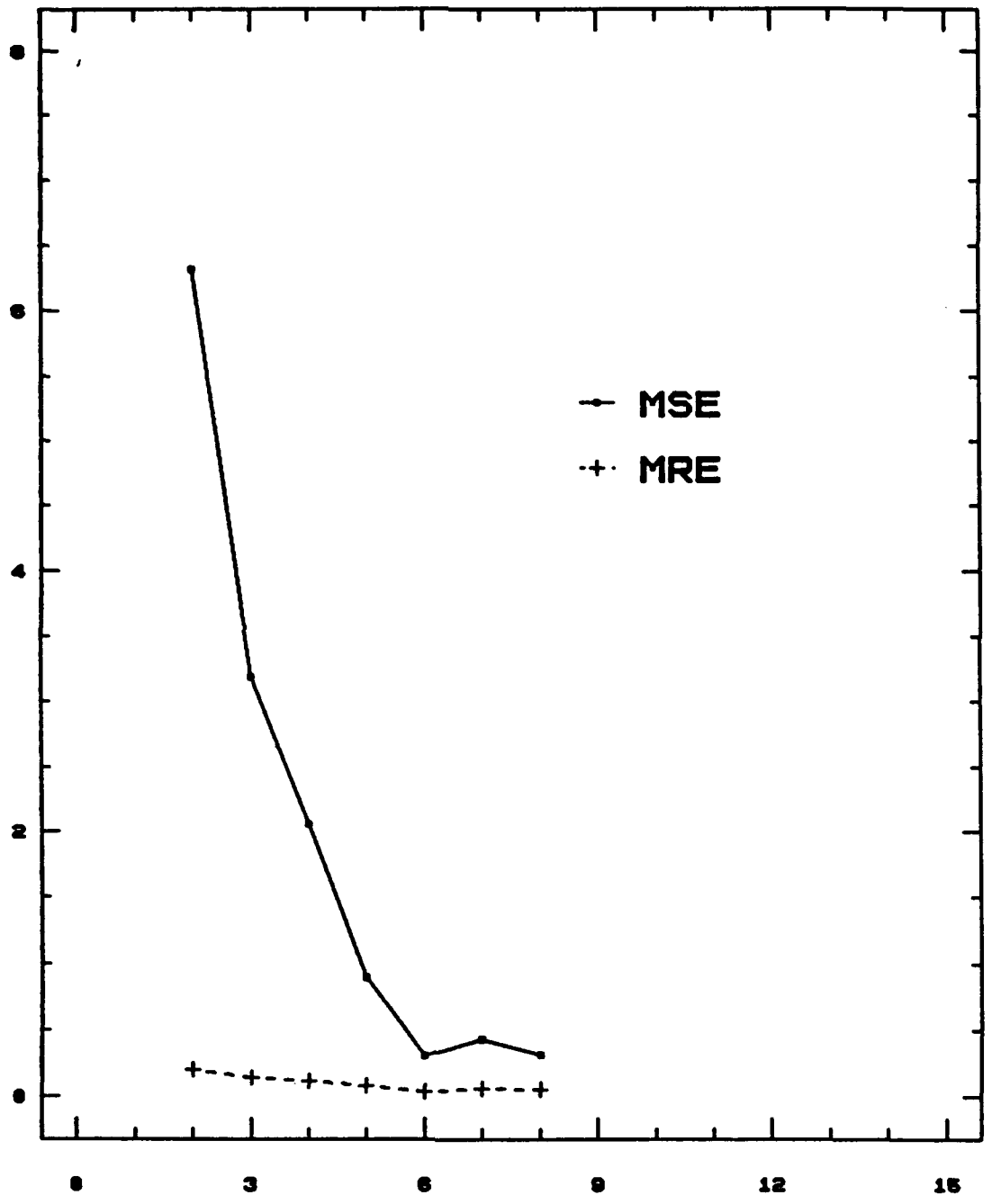


Figure 5 ... s (Startimg Interval) Method 2. Module 2.

Predicted & Actual Cumulative Failures,
s=7 (MSE Criterion), s=2

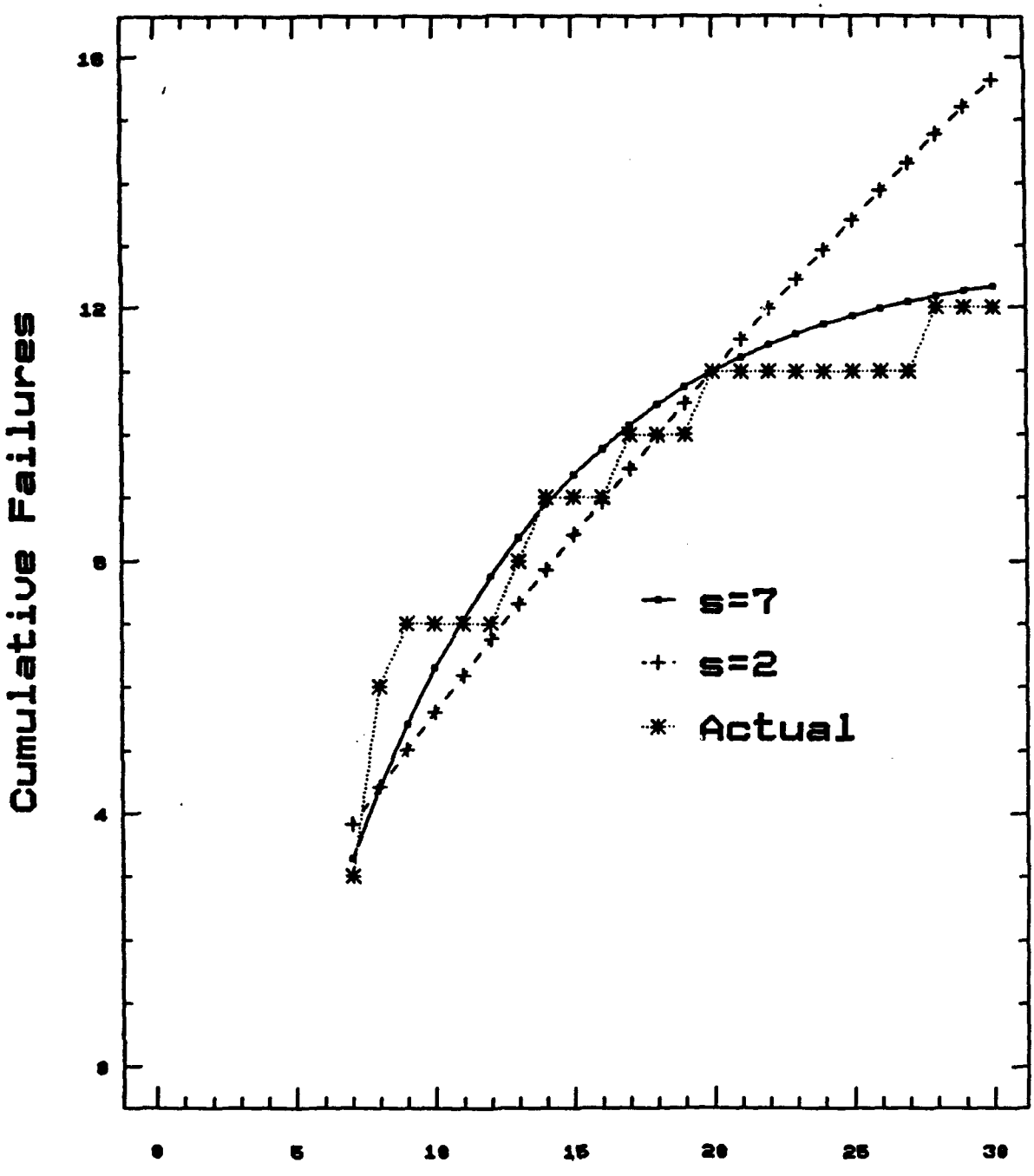


FIgure 6 ... Execution Time (Intervals)
Method 2. Module 2.

Mean Square Error: Parameter Estimation
Range(1-20) and Prediction Range(21-30)

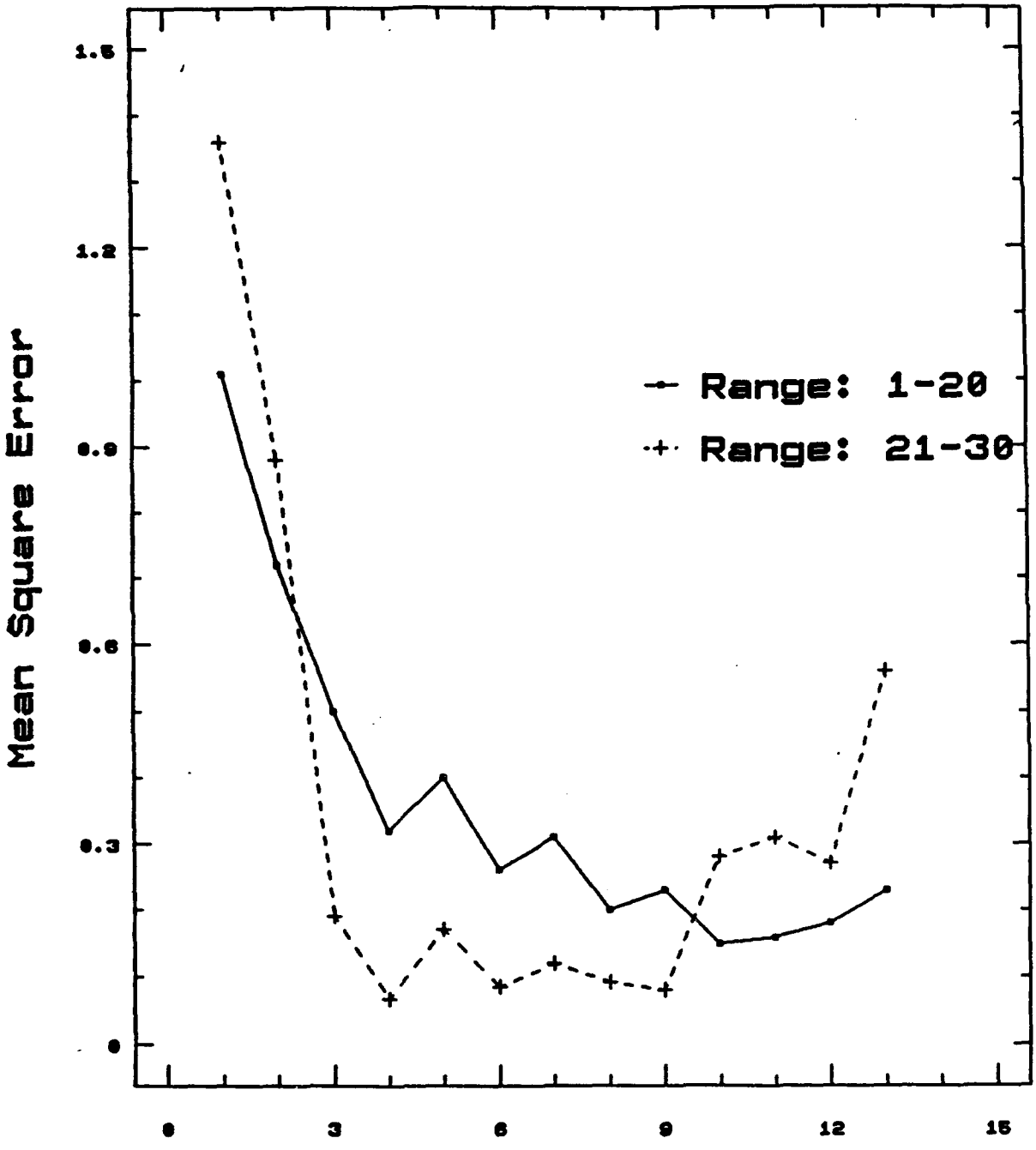


Figure 7 ... s (Starting Interval)
Method 2. Module 3.

Mean Square Error (MSE) & Mean Relative Error (MRE) in Prediction Range 21-30

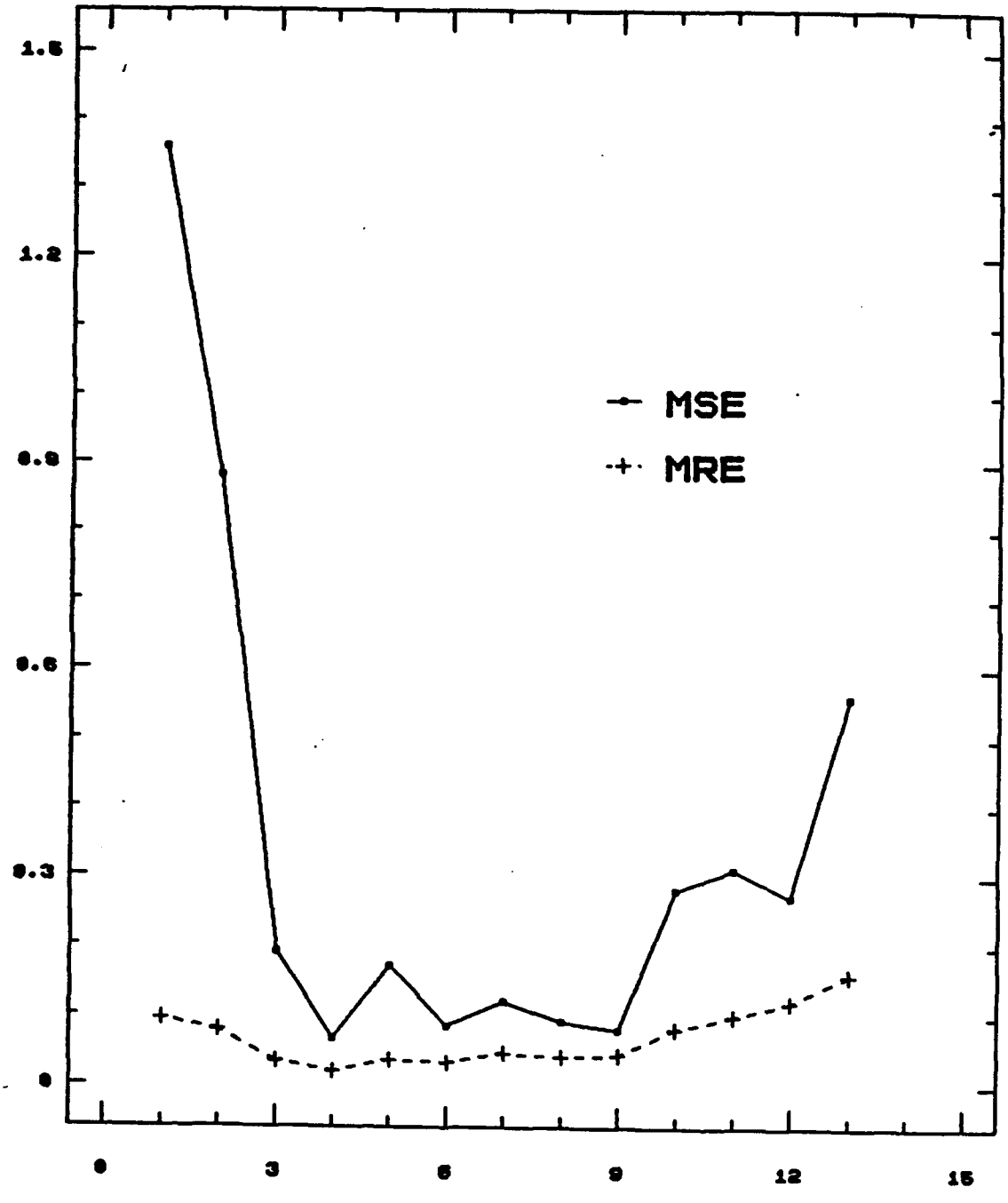


Figure 8 ... s (Starting Interval) Method 2. Module 3.

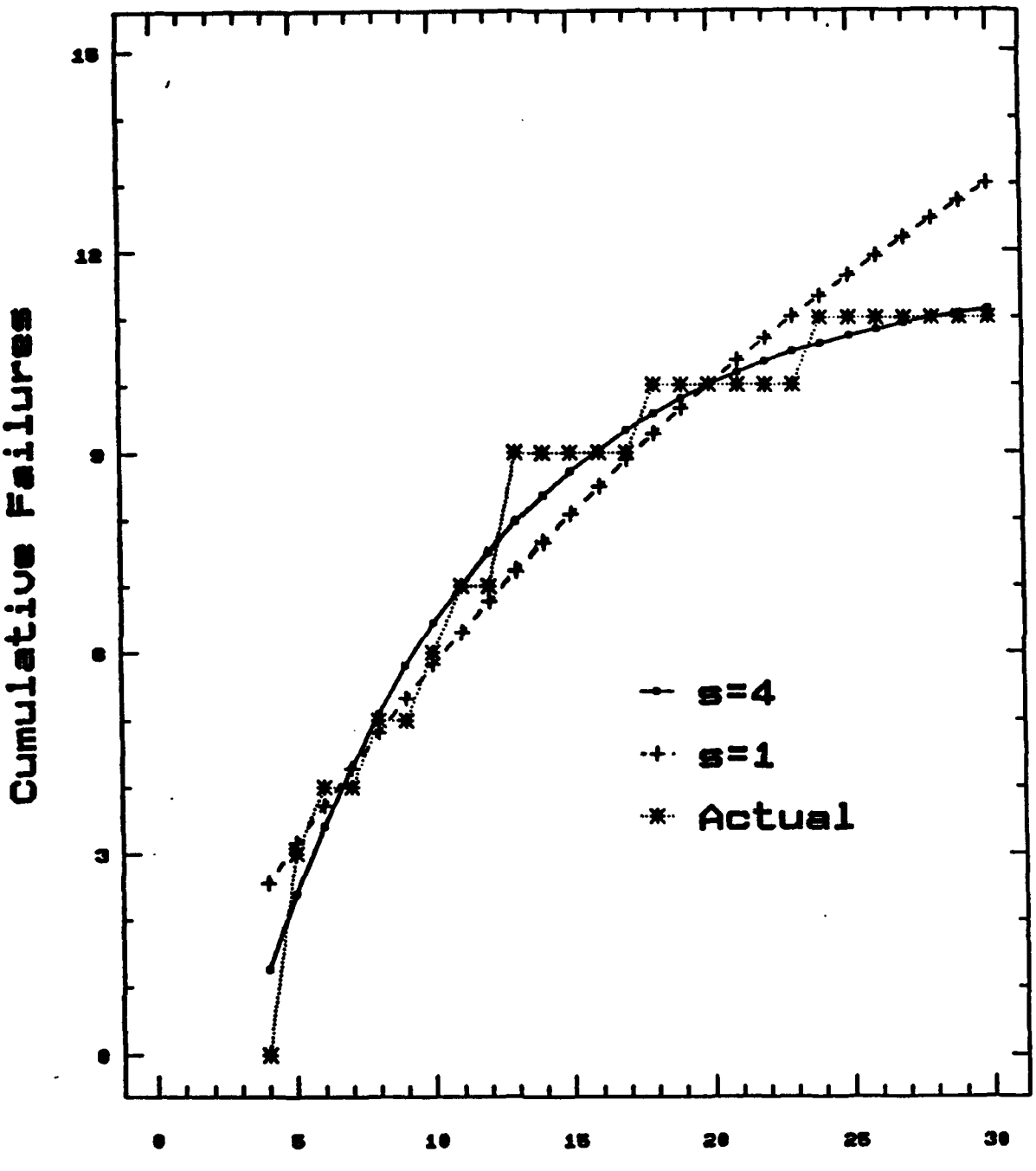


Figure 9 ... Execution Time (Intervals) Method 2. Module 3.

DESIGN STRUCTURING FOR SYSTEM ENGINEERING

1993 Complex Systems Engineering Synthesis and Assessment Technology Workshop (CSESAW '93)

July 20–22, 1993
Washington, DC

Jee–In Kim, Evan Lock
Computer Command and Control Company
2300 Chestnut Street, Suite 230
Philadelphia, PA 19103
Tel: 215–854–0555, Fax: 215–854–0665
Email: lock@cccc.com

Design Structuring for System Engineering

1. Overview of Design Structuring

The Design Structuring and Allocation Optimization (DESTINATION) methodology provides a systems engineer with a mechanism for making design decisions based on design optimization and trade–off analysis. The methodology can be used as a front–end methodology for building large, complex, real–time systems.

Most existing front–end methodologies provide a mechanism for specifying system design but lack a method of helping the systems engineer in determining a "good" system design. The DESTINATION methodology provides tools of Design Structuring, Resource Allocation, Design Evaluation and Optimization. Figure 1 shows an organization of the DESTINATION methodology.

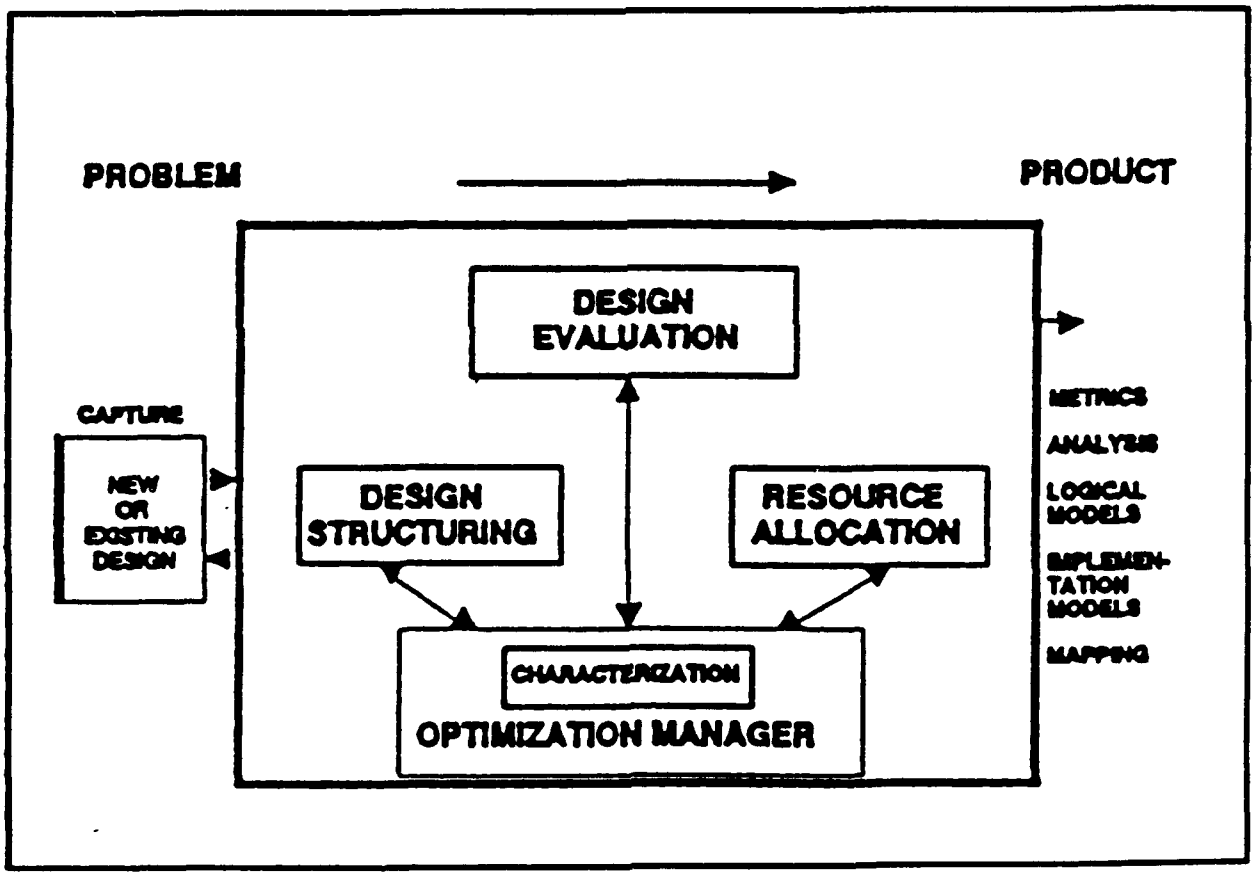


Figure 1: The DESTINATION Methodology.

This paper describes the Design Structuring component of DESTINATION. The system design specification and the system requirements of software, hardware and human interfaces of a system to be built are inputs for the Design Structuring component. The system design will be optimized through an iterative process of Design Structuring and trade–off analysis in conjunction with Resource Allocation, Design Evaluation and Optimization.

This paper is organized as follows: Section 2 explains terminology used in this paper. The objectives of Design Structuring are discussed in Section 3. The Design Structuring techniques are presented in Section 4. An example of Design Structuring is given in Section 5. This paper is summarized and concluded in Section 6.

2. Terminology

2.1. Design Element

A system design can be represented by a graph. The nodes and the edges of the graph have attributes which express more detailed system design information. A design element is a portion of such a graphical representation of a system design. A design element consists of a single or multiple collection of nodes and/or edges.

2.2. Design Structuring

Design Structuring is an engineering activity to construct graphical representations of the system design (including hardware, software and people). The goal of this activity is to produce a design that satisfies requirements in an optimal, or near optimal, manner. From a functional perspective, Design Structuring starts with system requirements, optimization criteria, and possibly an existing design and produces a new systems design. There are five basic design structuring operations—Decomposition and Recomposition, Fragmentation and Defragmentation and Replacement—which are defined below.

2.3. Decomposition

Decomposition generates additional design constructs at a more detailed level in the graphical design representation hierarchy (lower level of abstraction). Figure 2 illustrates that the details of a design element can be viewed via Decomposition. Node A and edges x, y and z of the data flow diagram is decomposed into nodes (A1, A2, A3 and A4) and the corresponding edges (x1, x2, y and z) connecting them.

2.4. Recomposition

Recomposition aggregates a design element of the hierarchy which consists of multiple nodes and connections. Thus Recomposition is an inverse operation of Decomposition. In Figure 2, nodes, A1, A2, A3 and A4, and their edges (x1, x2, y and z) are merged into a node, A, and edges, x, y and z, via Recomposition.

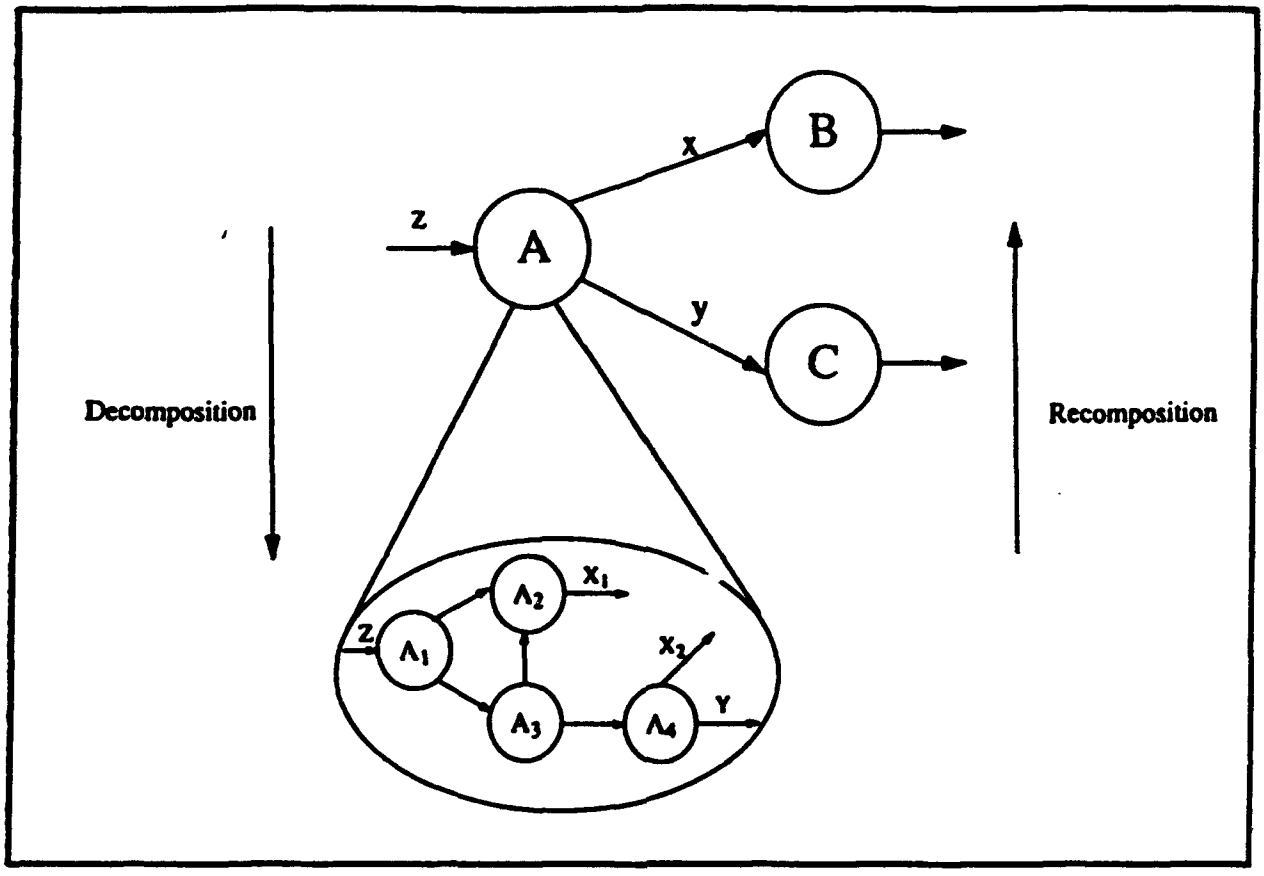


Figure 2: Decomposition/Recomposition Example—Data Flow Diagram.

2.5. Fragmentation

Fragmentation replaces a design element with a more complicated design element within a given level of the graphical design representation hierarchy (same level of abstraction). Replication of the same design element is a special case of Fragmentation. As shown Figure 3, a node of a data flow diagram, A, and edges, x and y, can be replaced by a set of nodes, A1, A2, A3 and A4, and edges via Fragmentation.

2.6. Defragmentation

Defragmentation replaces a design element which consists of multiple nodes and edges with a simpler design element which has less number of nodes and edges. In Figure 2, a design elements with multiple nodes, A1, A2, A3 and A4, and their edges, x and y, are merged into a simpler design element with a single node, A, and edges, x and y, via Defragmentation. Thus Defragmentation is an inverse operation of Fragmentation.

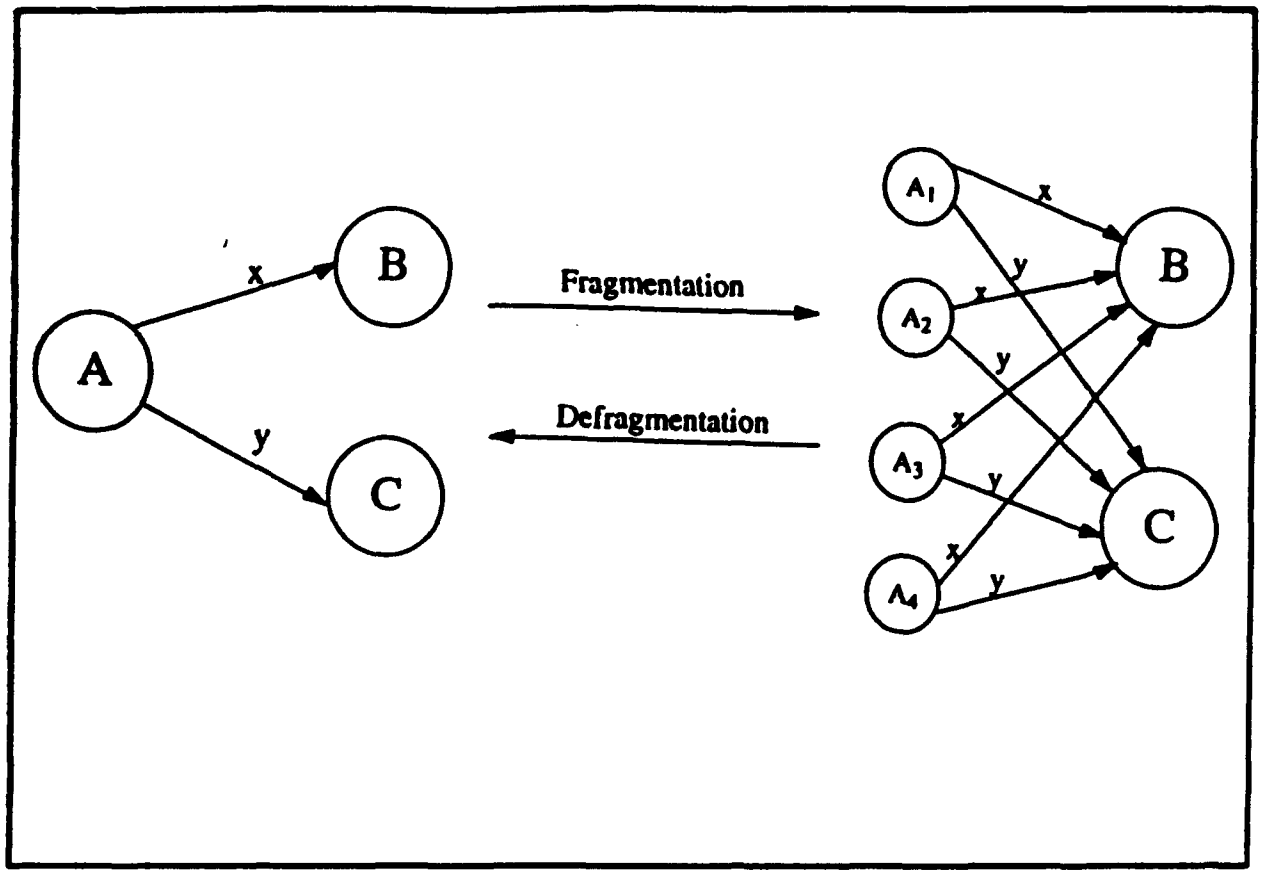


Figure 3: Fragmentation/Defragmentation Example—Data Flow Diagram.

2.7. Replacement

Replacement refers to a design structuring operation whereby a design element (either within the same or a different level of abstraction) is exchanged with another design element. Replacement does not include either Decomposition/Recomposition or Fragmentation/Defragmentation. The interface (i.e., connections) to the replaced design element remains the same. There is not necessarily a one–to–one mapping of nodes. Figure 4 shows an example of Replacing a design element in a graph which corresponds to three decision points. The corresponding metrics such as McCabe, Myer's Extension, etc. shown at the bottom. Figure 4 illustrates why one design structuring may be preferable [HMKD82].

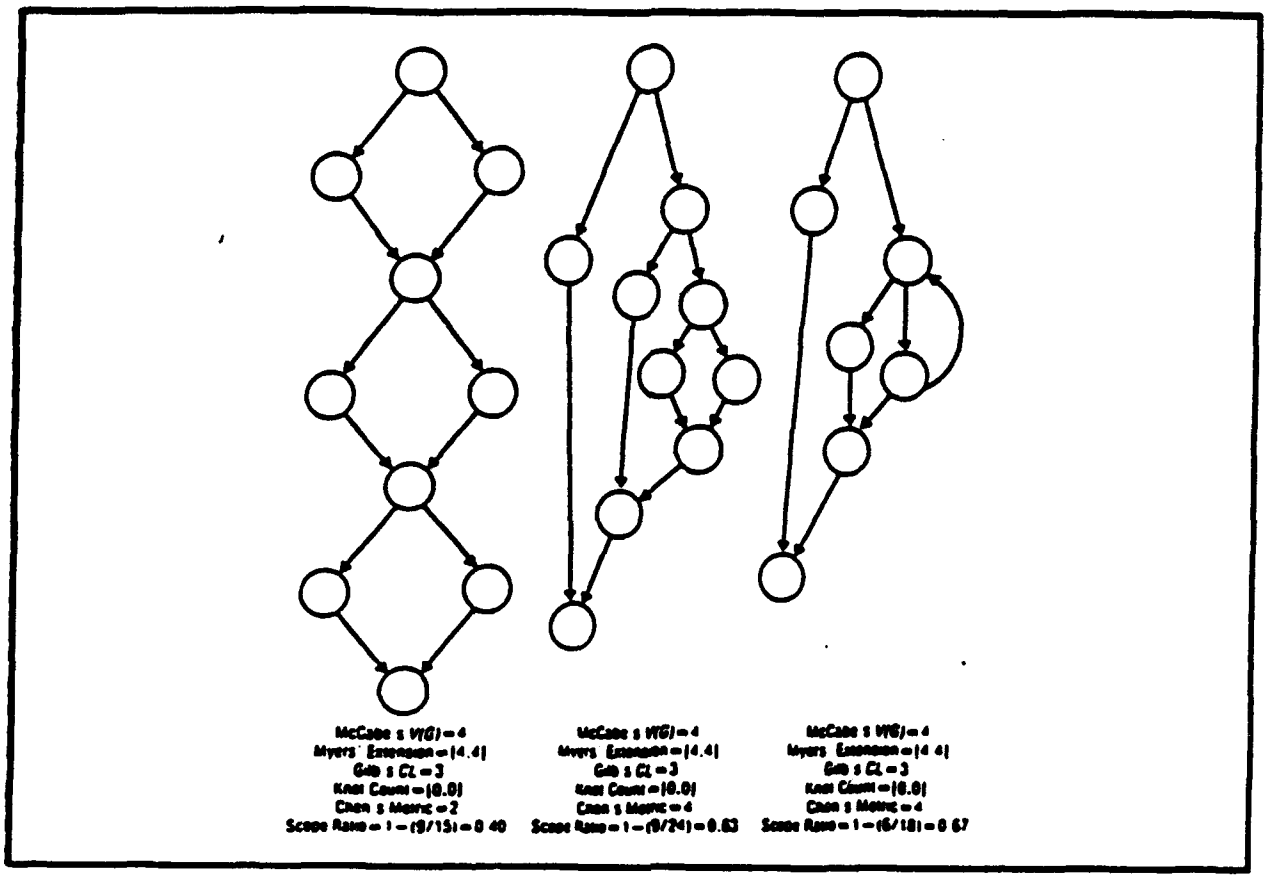


Figure 4: Replacement Example.

3. Objectives

There can be three general motivations behind why a systems engineer is interested in design structuring.

3.1. Facilitate Mapping across Design Capture Models

The system engineer may want to structure a design in such a way so as to make it easier, more intuitive, or for facilitating optimization when mapping logical models onto implementation models and vice versa. This also applies to the mapping of resources within the implementation model, particularly mapping software onto hardware.

3.2. Optimize System Design Factors (SDF)

The system engineer would like to synthesize and trade–off multiple designs that optimize for single criteria and eventually for multiple criteria or the values of system design factors (i.e., non–functional requirements). The list below contains examples of three particular SDFs that are of particular interest.

(1). Performance
Performance can be improved by replicating (a special case of Fragmentation) design units to increase parallelism or by defragmenting to decrease communication overhead. Defragmentation and replication represent design structuring techniques.

(2). Dependability
Similar to Performance, Dependability may be improved through Replication or Defragmentation.

(3). Maintainability
Improving system maintainability is strongly influenced by understandability, which is similar to the first objective. One example of design structuring which impacts maintainability would be to analyze a graph which depicts the relationships of global variables and restructures it to minimize module coupling.

3.3. Improve Design Understandability

The most common way to measure understandability has been through various complexity metrics. Examples include size, McCabe, scope, etc.

4. Technique

Design Structuring techniques (be they heuristics or algorithms) arise from one of three areas: the application domain, a design methodology, or graph theory. Each of these is briefly mentioned below.

4.1. Application Domain

There are many domain–specific factors in system design. For example, Fragmentation is useful in improving parallelism in a massively parallel processing environment. On the other hand, Fragmentation can make a diagram complicated if the diagram is to be displayed in a graphically based system. Certainly in Navy applications, such as sonar processing, there are specific devices and functions that are specific to a family of systems. Experts in a particular domain have developed design structuring principles over time that can be captured and applied.

4.2. Design Methodologies

Heuristics depend on a design methodology of a system such as Object–Oriented Design, Structured Design, Task Structuring, Partitioning, etc. For example, the Task Structuring technique of ADARTS (Ada Based Design Approach for real Time Systems) [ADRT88] can be used.

4.3. Graph Theory

Complexity of a design can be measured as a number of nodes and connections of a diagram. For instance, it is desired to have seven plus or minus two bubbles in any individual diagram to facilitate human comprehension. However, this kind of simplistic guideline may easily be broken. Other graph theories, involve the number of decision points, crossing edges, etc.

5. Example

Figure 5 graphically represents domain of Design Structuring. A system engineer performs Design Structuring according to objective, operation, technique, system perspective, and design capture view.

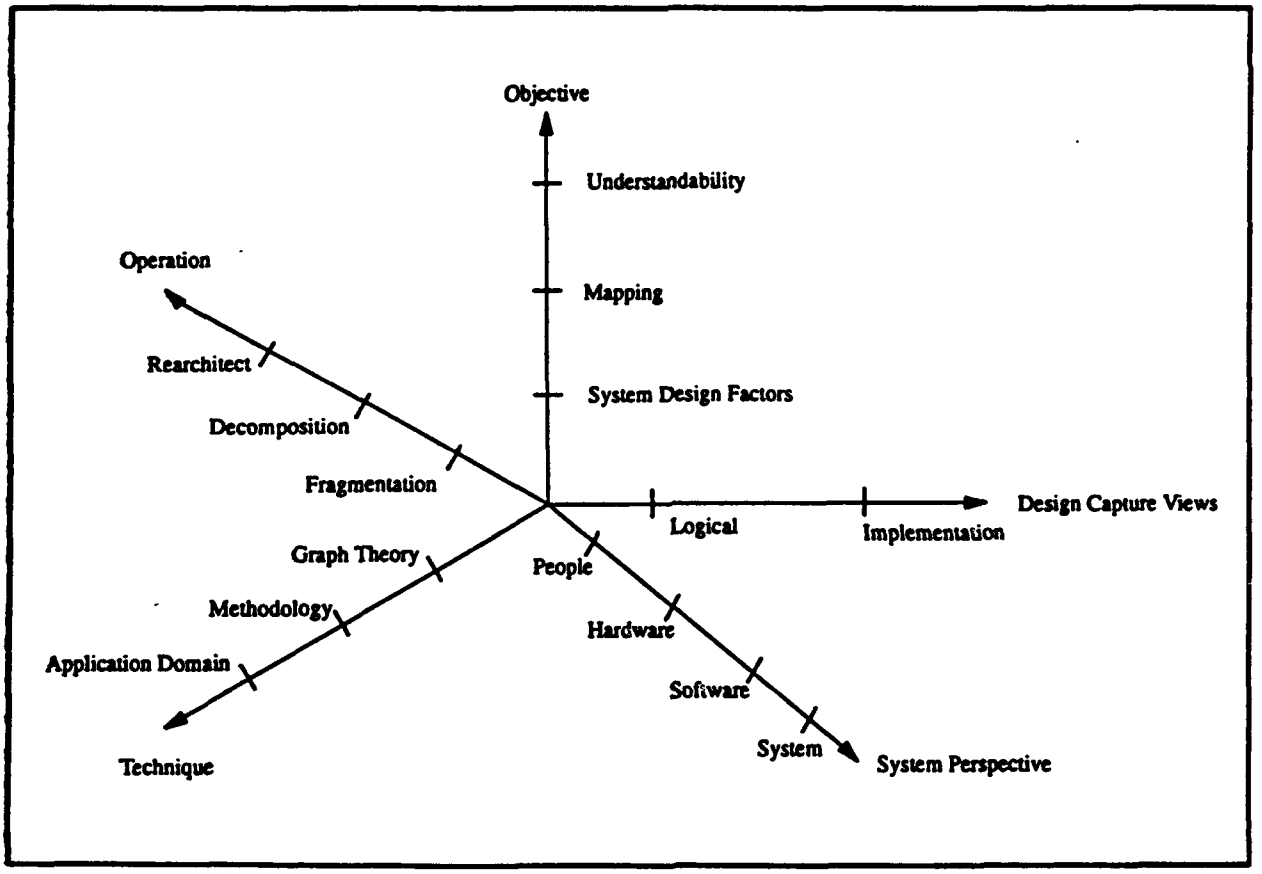


Figure 5: Domain of Design Structuring.

This section presents a sample Design Structuring technique based on ADARTS, a system design methodology for structuring a system into concurrent tasks (or active objects). The performance of the system can be increased through the task structuring process. In order to develop more maintainable and reusable components, ADARTS provides criteria of identifying modules (or passive objects). The ADARTS methodology is supported by many companies such as Software Productivity Consortium and Cadre. The DESTINATION methodology can utilize the tools from these companies. It can use ADARTS for system design and optimize the system design through iteration of software design with hardware decisions and organizational lessons with respect to System Design Factors (SDF) [SDF92].

5.1. ADARTS

Figure 6 illustrates steps of the ADARTS methodology. The first step of ADARTS is to express a system design in Real Time Structured Analysis (RTSA). Then concurrent tasks (in Dynamic View or Task Architecture Diagram) and modules (in Static View or System Architecture Diagram) are identified. A system architecture is derived from those diagrams.

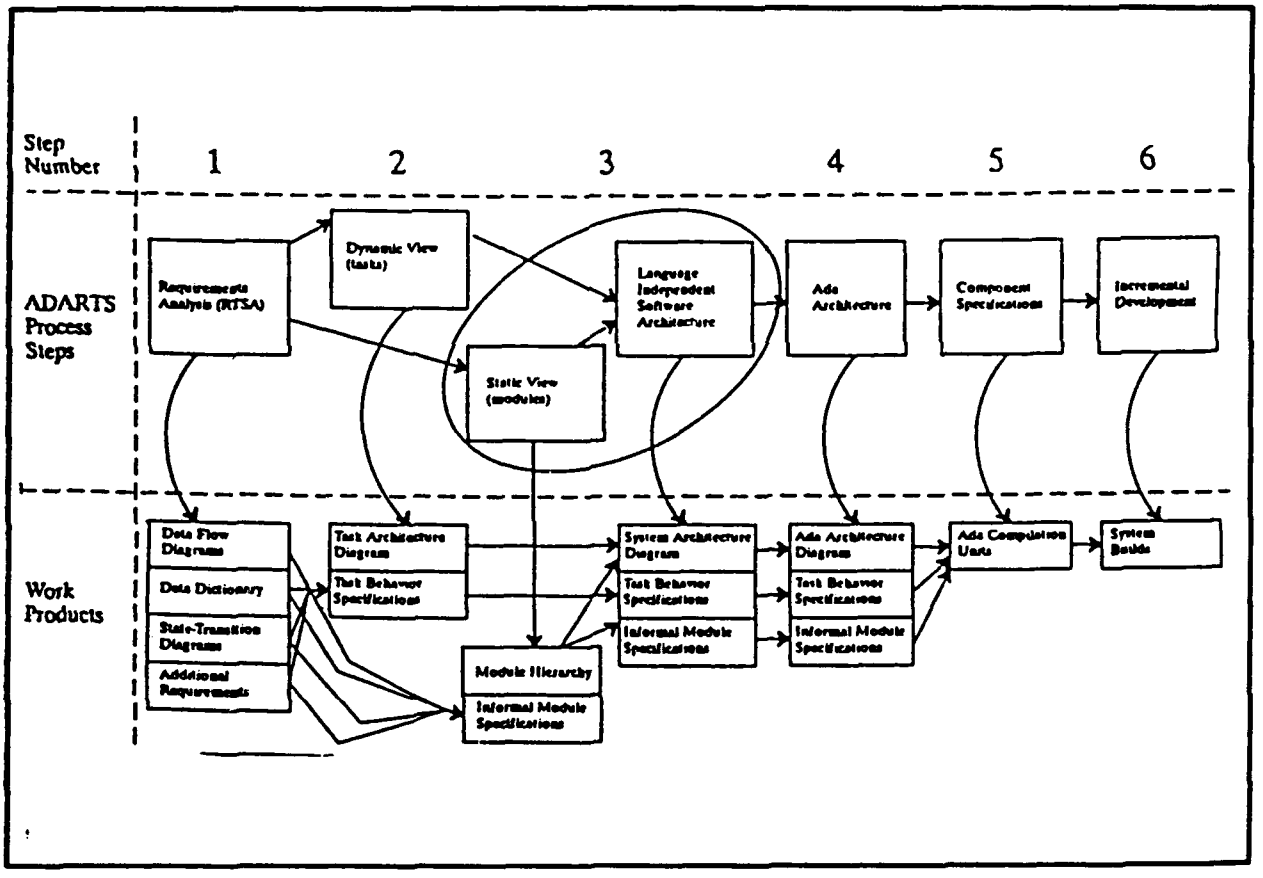


Figure 6: Steps in the ADARTS Methodology.

5.2. Cruise Control System Example

A cruise control system of an automobile is used to demonstrate the Design Structuring method based on ADARTS. An overall data flow diagram of a cruise control system is given in Figure 7. The ADARTS task structuring criteria are applied to the data flow diagram of the cruise control system design. A system engineer obtains a task architecture diagram as shown in Figure 8 from the data flow diagram.

The criteria of object–oriented design are used for information hiding of hardware, software and people of the system. A number of design structuring operations are applied to the task architecture diagram in Figure 8. For example, a task, "Monitor Auto Sensors", in Figure 8 is "decomposed". The same task in Figure 9 contains three objects denoting sensing devices. Another task, "Auto Speed Control", in Figure 8 is "recomposed" to become an object, "Auto Speed Control", in Figure 9. The message queue, "Sensor Change", in Figure 8 is "replaced" by a new object, "Cruise Control Event Buffer". The result is a system architecture diagram shown in Figure 9.

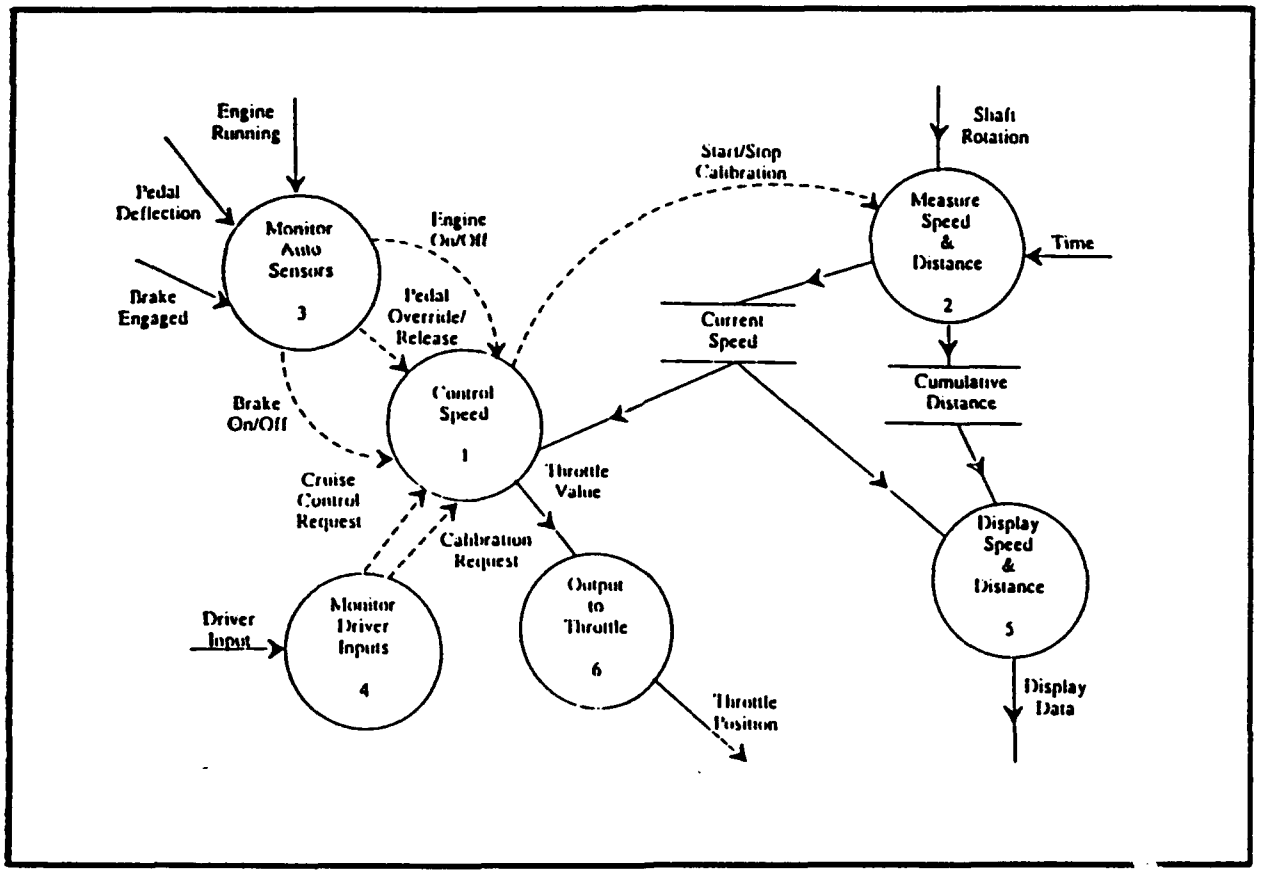


Figure 7. Overall Data Flow Diagram—Cruise Control System.

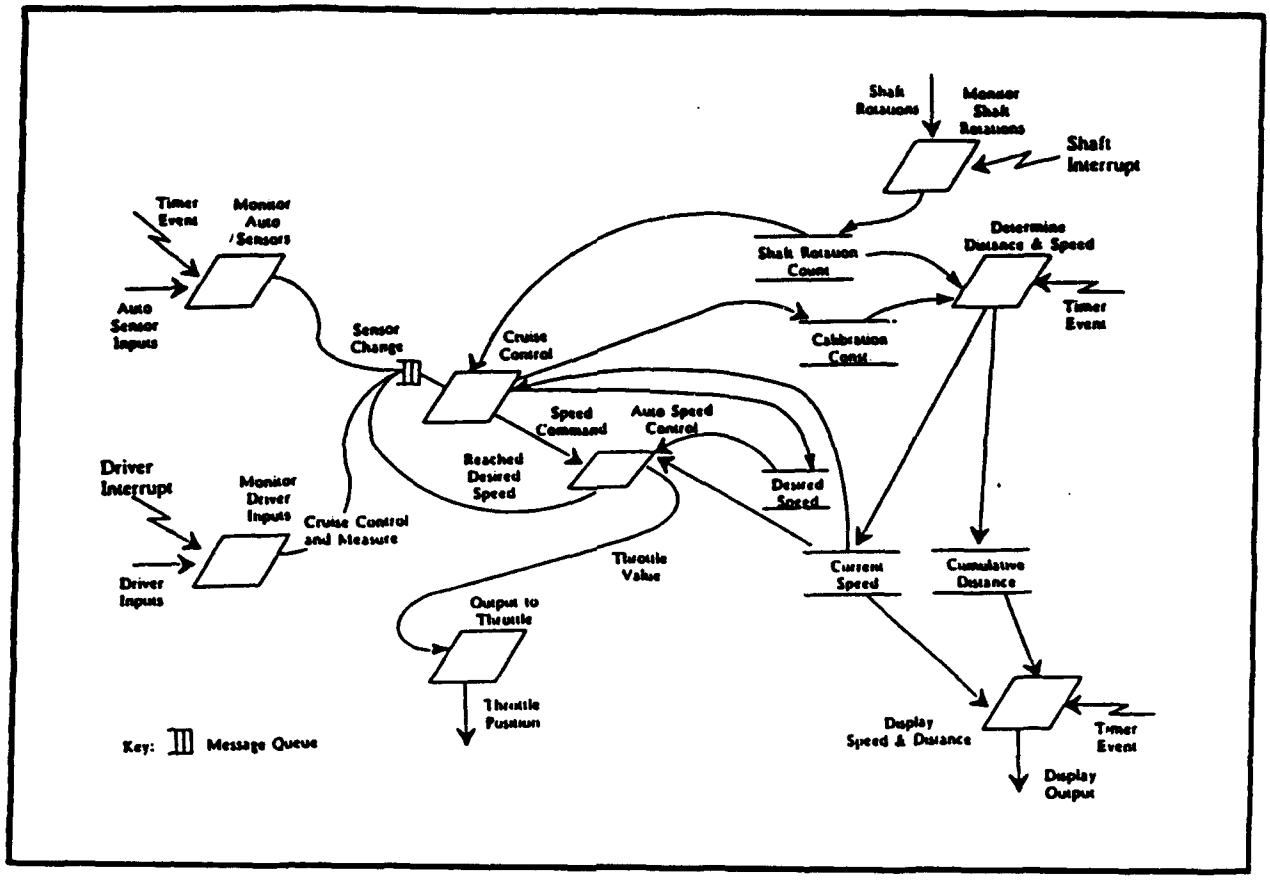


Figure 8. Task Architecture Diagram—Cruise Control System.

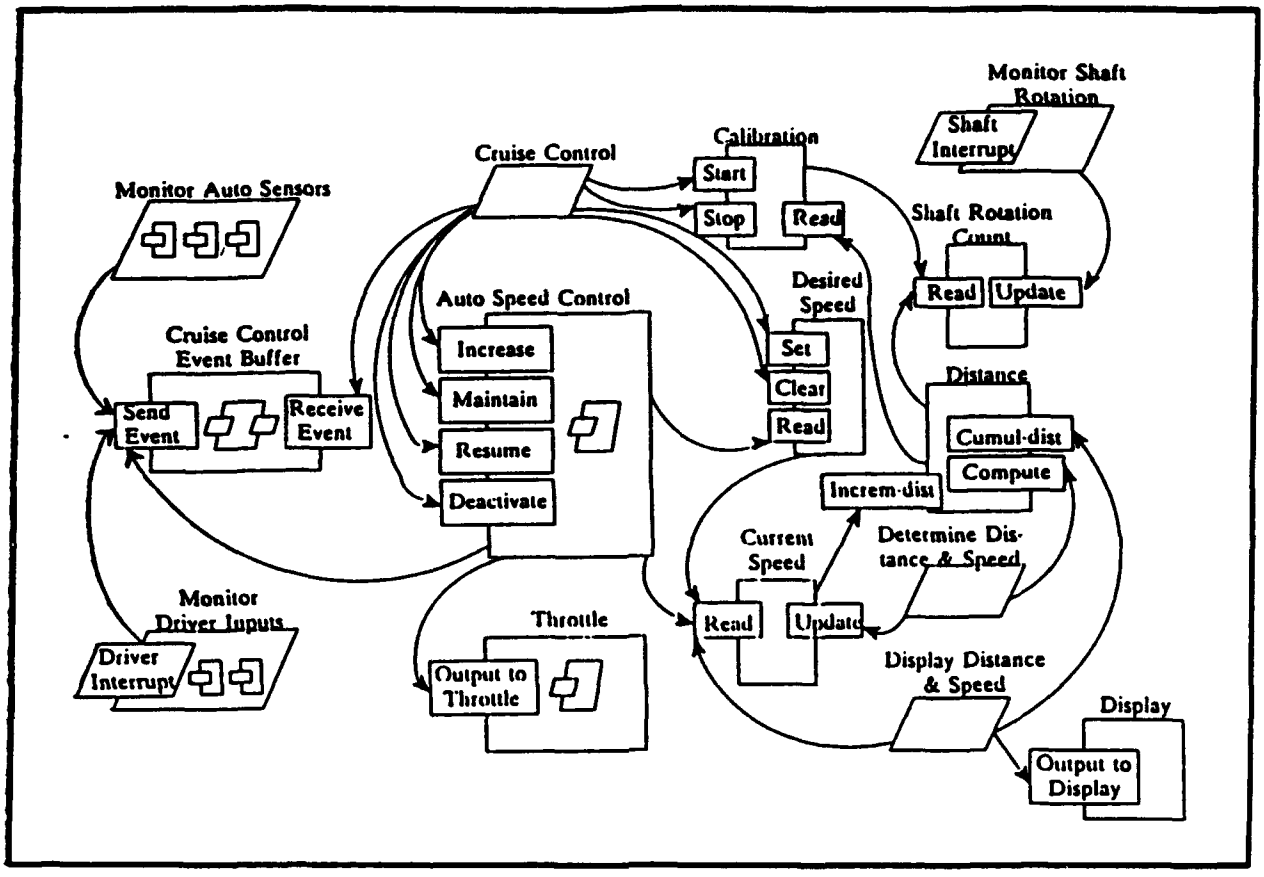


Figure 9. System Architecture Diagram—Cruise Control System.

6. Conclusion

A research effort of developing a Design Structuring method has been presented in this paper. The goal is to produce optimal or near optimal system design that satisfies system requirements. We have discussed terminology, objectives, technique, and an example of Design Structuring.

Currently, we investigate a System Design Structuring methodology based on the ADARTS approach. The methodology involves system engineering through iteration of software design with hardware decisions and organizational lessons with respect to SDFs. A set of Design Structuring criteria according to SDFs (System Design Factors) is researched. A method of optimizing a system design with respect to SDFs is studied.

A prototype of DESTINATION will be constructed for demonstration. The Design Structuring component of the prototype will utilize the Design Structuring criteria, the optimization method and the CASE tools supporting ADARTS.

7. References

[ADRT88] Hassan Gomaa, "ADARTS—An Ada Based Design Approach for Real Time Systems, Version 1.0," Software Productivity Consortium Technical Report, SPC–TR–88021, August 1988.

[DEST] S. L. Howell, C. Nguyen and P. Q. Hwang, "Design Structuring and Allocation Optimization (DeStinAtiOn): A Front–End Methodology for prototyping Large, Complex, Real–Time Systems," Proc. Hawaii International Conference on System Sciences, IEEE Computer Society Press, Los Alamistos, CA January 1992, Vol. II, pp. 517–528.

[SDF92] C. Nguyen, S. L. Howell and P. Q. Hwang, "Systems Design Factors, Version 0.01," Naval Surface Warfare Center, NAVSWC TR–92–XXX, February 1992.

[HMKD82] W. Harrison, K. Magel, R. Kluczny and A. DeKock, "Applying Software Complexity Metrics to Program Maintenance," IEEE Computer (September 1982), pp. 65–79.

ASSESSMENT

A Platform for Complex Real-Time Applications

Alexander D. Stoyenko
***Lonnie R. Welch**
Phillip Laplante[†]
Thomas J. Marlowe[‡]
Carlos Amaro
Bo-Chao Cheng
Matthew Harelick
Xue Jin
A. K. Ganesh
Gray Yu

Abstract

A platform for complex real-time applications is presented. The platform consists of a number of on- and off-line components: a RISC architecture, a runtime kernel, a generic interconnected network, a specification and design language, a programming language, compiler, linker, schedulability analyzer, assignment tool, and a graphical user interface. All components adhere to stringent requirements of predictable real-time computation. The integrated platform has been prototyped and a successful initial evaluation has taken place. We are now extending the platform to accommodate the requirements of two collaborative Navy projects — in task allocation and optimization, and in component re-engineering.

1 Introduction

The emerging generation of complex real-time applications requires availability of application development and execution platforms which integrate a number of traditional and novel components. Essentially, a suitable platform needs to accommodate the inherent complexity, distribution, parallelism, adaptibility and non-functional requirements (such as time criticalness, fault tolerance, dependability and security) — characteristics not necessarily found in older real-time applications — of these new applications. In this paper, we report briefly on such a platform, that we have designed and prototyped at New Jersey Institute of Technology's Real-Time Computing Laboratory.

We recognize the following steps — and thus the corresponding need for a platform component — in the development and execution lifetime of a complex real-time application. Initially, an application is specified and then designed, using a visual specification and design language. Next, the application is constructed and/or synthesized from pre-existing, reusable components. The pre-existing components are selected on the basis of their interfaces. These components are managed through the Component

*Stoyenko, Welch, Amaro, Cheng, Harelick, Ganesh, Jin, Younis, and Yu are with The Real-Time Computing Laboratory, Department of Computer and Information Science, New Jersey Institute of Technology, Newark, NJ 07102 USA, E-mail: alex@vulcan.njit.edu or welch@vienna.njit.edu. This work is supported in part by the U.S. ONR Grant N00014-92-J-1367, by the U.S. Army Grant DAAL03-91-C-0034, by the NATO Grant CRG-90-1077, by the U.S. NSWC Grant N60921-93-M-1912, by the AT&T UEDP Grant 91-134, and by the NJIT SBR Grants 421250 and 421290

[†]Department of Mathematics and Computer Science, Fairleigh Dickinson University, Madison, NJ, laplante@fdumad.fdu.edu, also associate member of NJIT RTCL

[‡]Department of Mathematics and Computer Science, Seton Hall University, South Orange, NJ 07079, marlowe@cs.rutgers.edu, also associate member of NJIT RTCL

Manager, not discussed here but presented in [4]. New components are developed through translation of the specification and design language as well as naturally through programming in a high-level language. The resulting application program is consequently compiled piece-by-piece down to executable form, and then linked. During compilation, essential timing and other non-functional information is extracted (using the front-end function of a schedulability analyzer). A Directed Acyclic Graph (DAG) of software application components — processes and objects — is constructed, for display and further analysis. The DAG is consequently analyzed by the assignment analysis tool, and the software components are assigned to processing elements (PEs) of the generic interconnected network. Next, the components are loaded and the application commences execution. Specifically, each PE runs a copy of a runtimé kernel, which executes instructions of the software components and initiates communication. The network (with its own kernel) processes communication messages. The application and the platform — or more specifically, the DAG and the interconnected network of PEs — are displayed graphically in windows. In addition, the performance of the application and the platform is monitored and displayed. Finally, there is a command window which works in conjunction with other windows, and allows the user to re-assign components, change performance monitor attributes, provide application data and so forth.

In what follows, we outline each application platform component and summarize relevant current and future activities.

2 Specification and Design Language

To present multiple functional and non-functional views of complex real-time applications, we have designed a rigorous specification and design description language *RT-Chart*. RT-Chart specifications representation a system as a set of real-time processes, each composed of a set of actions. The first action of a process is performed at the start of each process activation. Upon completion, the initial action invokes a successor action, passing some data. Each action requires a set of resources (hardware or software), which are claimed upon commencement of the action and are released upon completion of the action. RT-Chart provides a resource algebra for stating the modes in which resources can be used: (1) may be used concurrently (2) must be used concurrently (3) cannot be used concurrently and (4) cannot be used concurrently and must be used in a particular order. Additionally, and-gates allow the specification of parallel actions, and or-gates indicate conditional execution of one among a set of actions. Furthermore, in RT-Chart, the actions may be hierarchical, to enable macro-level reasoning and specification. In addition to functionality, timing and parallelism, there are other important system aspects that RT-Chart allows to be expressed. Security classification levels may be indicated for information flows, code (actions), resources, levels of hierarchy, and implementation details. To allow dependability to be dealt with, degree of redundancy or reliability can be specified for actions or processes. Another aspect of systems is relative criticality, which can also be expressed for actions and processes.

Currently, RT-CHART is implemented only partially. There is a visual, graphical interactive user interface, to create and modify RT-CHART specifications. However, the translator from RT-CHART to RTX or call-DAGs (see Sections 3 and 4) has not yet been completed.

3 High-Level Language

A predictable real-time language *RTX* has been designed, based on the real-time model of Real-Time Euclid [7, 3] applied to the ADT/ADO model of RESOLVE [5, 14]. The resulting language features abstract data objects, real-time processes and time-bounded constructs. No arbitrarily-long computation is allowed. Thus, recursion is forbidden, loops are unrollable, and dynamic data operations are not allowed.

One interesting feature of the RTX — which makes accurate timing predictions even more challenging than in other real-time languages — is that a call in RTX may proceed in parallel with the caller that made the call until either a control or data dependency forces synchronization or the call returns. Moreover, since RTX inherits RESOLVE's strict object-orientation, every statement in an RTX program is in fact either a primitive operation or a call (thus contributing to the complexity of timing analysis).

4 Compiler, Schedulability Analyzer, DAG Generator and Related Tools

A compiler for RTX has been developed. While the compiler supports such interesting features as generic ADTs, the emphasis on this work is on supporting real-time features. Consequently, the compiler operates in an integrated fashion with a front-end schedulability analyzer (see below). In addition to schedulability information and conventional RISC instructions to be executed later (see Section 7), the compiler also outputs information used by the DAG generator (see below).

A schedulability analyzer operates similarly to the one for Real-Time Euclid [9, 7, 1]. A program is analyzed for schedulability in two stages. The schedulability analyser consequently consists of two parts: a partially language-dependent front end and a language-independent back end. The front end is incorporated into the code emitter, and its task is to extract, on the basis of program structure and the code being generated, timing information and calling information from each subprogram, process or object, and to build language-independent program trees. The front end of the analyser does not estimate interprocess contention. However, it does compute the amount of time individual statements and subprogram and process bodies take to execute in the absence of calls and contention. These times, serving as lower bounds on response times, are reported back to the programmer.

The back end of the schedulability analyser is actually a separate, language-independent program. Its task is to correlate all information gathered and recorded in program trees by the front end, and to predict guaranteed response times for the entire real time application. To achieve this task, this part of the analyser maps the program trees onto an instance of a real-time task model, and then computes the response time guarantees.

The back end of the analyzer employs a potentially exponential time technique — frame superimposition — to estimate contention accurately. To reduce the problem space frame — in terms of the number of possible execution paths — that frame superimposition has to consider, we employ program transformations and resource contention [10], conditional linking [11], and aperiodic process conversion to periodic (using common techniques such as in [2]). The transformer balances and pads alternate execution paths in conditional statements, ensuring that regardless of the path taken, timewise both the execution of the thread in question and the contention the thread generates is the same as in the case of one original, dominating path. The linker eliminates paths that are infeasible, given logical relationships among conditional test variables. Restricted resource contention means that arbitrary contention schemes for resources are reduced to a dominating small number of schemes, at the expense of some time loss. A typical scheme may force multiple threads contending for the same resource to be released only after all of them have used the resource. Finally, aperiodic-to-periodic process conversion involves replacing processes with longer frames (minimal activation separation periods [7]) but otherwise arbitrary times, with regular (such as periodic or jittery) processes with identical per-activation resource requirements but tighter frames.

Daggen (the DAG generating tool) translates RTX compiler output into a sequence of numbers used by the DAG Browser of the Graphical User Interface to draw a Directed Acyclic Graph. The Directed Acyclic Graph (DAG) extracted by Daggen represents a collection of processes and objects. DAG edges represent calls from processes or operations of objects to operations of (other) objects. (Thus, we also refer to this graph as the *call*-DAG). Since RTX disallows recursion (operation- or object-level),

the graph is indeed acyclic. Daggen builds the call-DAG of the application by processing each object declaration in a top-down recursive fashion beginning with the top level processes. Adjacency lists are constructed, along with parameters, their directions and other attributes, for each call possibility. For display purposes, at most a single edge is represented for any two possible nodes.

5 Assignment of Processes and Objects

We have developed and continue improving upon accurate and efficient performance prediction functions for real-time software components (processes and objects) that are being assigned [13, 14, 6] to processing elements (PEs) in parallel systems and that allow RTX-style asynchronous operation calls. The functions (1) consider load-balancing, parallelism and deadlines, (2) predict performance on the basis of projected service rates, and (3) treat PEs and communication links in an integrated manner.

We have undertaken a quantitative evaluation which has demonstrated that the functions perform quite well [12]. The assignment tool is currently used before execution, to assign processes and objects to the PEs of our platform (see Section 6). Eventually, the tool will also be used to re-assign processes and objects dynamically, as the need arises.

6 Processing Elements and Networking

The platform architecture consists of processing elements (PEs) interconnected by a generic network. Each PE is controlled by a replica of the PE kernel (see Section 7). The PE architecture is currently a RISC computer based on [14]. While the architecture is mostly conventional, it includes features for efficient loading, execution and cloning of objects and processes. Furthermore, the architecture is mostly predictable in its real-timing, in the sense of the architecture in [1]. Potentially unpredictable features of the architecture — such as dynamic memory allocation and pipelining — are made predictable by the compiler (which disallows memory allocation past module load, and swaps/pads post-branch instructions, for instance).

Each PE is equipped with communication ports. There is no limit on the number of PEs in the system, and we have experimented with a number of topologies (a ring, a bus, a mesh, a hypercube). We are building a generic network topology constructor to enable arbitrary interconnection of common topological structures (e.g. five hypercubes and three rings connected by a bus).

The architecture provides PE and communication link resources. Each resource is assumed to be schedulable separately, for generality. Every resource has its own queue for requests and is free to employ any predictable scheduling policy.

7 PE and Network Kernels

The operating system component of the platform consists of small kernels for PE and network control, respectively. Each PE is equipped with a replica of the PE kernel, whose tasks include process scheduling, initiating and receiving messages, and gathering execution information. These functions are done in a standard fashion, employing common and straightforward decisions such as process priority inheritance and deadline detection. Moreover, the PE kernel supports both synchronous and asynchronous calls (see Section 3) — calls inherit caller priority. Both preemptive and non-preemptive scheduling are supported. The PE kernel allows the user to select synchronous or asynchronous call mode. The PE kernel passes information on performance statistics and execution progress (as in who is calling whom) to the graphical user interface (Section 8). Finally, the PE kernel is used to control the speed of program execution (user-selectable — to enable execution progress at human speed for

observation).

The network kernel executes the network, including message transmission, link scheduling, delay updates (transmission and propagation delays) and other such functions. This kernel too interfaces with the graphical user interface. Since the design of the architecture has deliberately kept the PEs and the interconnection (and consequently their kernels) separate, there is a need to maintain common granularity and value of time. This is currently achieved through a common router for messages and time keeping. Future implementations are likely to also employ simple and reliable time management, such as monitoring official time [1], rather than high-overhead, theoretical synchronization protocols.

8 Graphical User Interface

We have built a graphical user interface (GUI) to display the activities in the system and to accept commands from the user. The GUI currently consists of the command window, the call-DAG window, the architecture window and the performance monitoring window.

The command window controls platform and application execution. Through this window, the user may instruct the platform to load, execute or terminate and application, to (re-)assign a process or an object (this is done by identifying the command, the object in the call-DAG window and then the PE in the architecture window — the PE kernel does not currently support this feature), to set the program execution speed dynamically, and to open the other windows.

The call-DAG window displays an application in its call-DAG form. Using colors, line sickness and other display features, the window clearly indicates executing, idle or blocked nodes, calls-in-progress over edges, and call and return parts of calls. The window displays names or numbers of nodes.

The architecture window displays interconnected PEs. Again using colors, shapes and so forth, the status of each PE's processor (executing, idle, waiting), its communication ports (idle, sending, receiving, waiting), and the status of communication links (transmitting, blocked, reserved, idle) is indicated. Call and return messages are shown. Finally, the user may request to display the portions of the call-DAG corresponding to the processes and objects — optionally with their call-DAG neighbors — to be displayed for any PE.

Finally, the performance window displays sets of bar graphs and an index table, each with timing information on processes (and objects and their operations) that run to completion. Specifically, the statistics displayed (and requested by the user through this window) include observed, predicted (by the assignment tool or the schedulability analyzer) and historical response times, laxities, frames and deadlines, and the deviations among the predicted and observed sets of times. Information on the last termination for every process or object is kept and displayed upon request, as is information on the last N terminations, for a user-specified N. The index table maps process or object numbers to declared names and their associated frames and deadlines, if applicable.

9 State of the Platform, Current and Future Activities

Our current prototype has been implemented and running since Spring'93. PE and network kernels, the router and the GUI are run as Unix[1] processes that communicate via sockets. The GUI and RT-CHART windows run under X (X-view and Motif). The PE kernel uses shortest-latency-first with inheritance to schedule processes, and the two kernels use FIFO for link and message scheduling. The time granularity is kept consistent (between the PEs and the network) by the router (that also keeps the master clock).

We are currently building a graphical DAG constructor to enable rapid prototyping of complex

[1] Unix is a trademark of the AT&T Bell Laboratories

applications. The constructor will fill in cycle-burners in place of real instructions, so that the resulting symbolic application will exhibit the same desired timing and other non-functional behavior as the ultimate real application. Now that we have a "proof-of-concept" prototype, we are about to undertake in a thorough re-design and re-engineering of the platform. One of our tasks in re-design is to facilitate the use of common languages, such as Ada9X, in the spirit of real-time execution [8] and while enabling re-use, re-engineering and other desirable features.

Finally, we are engaged in two collaborative projects with NSWC: one in task allocation and optimization, and the other in software component re-engineering. The platform is used in both projects and we have begun extending it to meet the NSWC requirements.

We are indebted to the Office of Naval Research and the Naval Surface Warfare Center (White Oak and Dahlgren) for making this project possible — and especially for facilitating what has now become a productive, collaborative effort. We would like to thank all other members of the Real-Time Computing Laboratory — past and present, regular and visitors — who have contributed immensely to the creative and professional environment within the Lab.

References

[1] W. A. Halang, A. D. Stoyenko, *Constructing Predictable Real Time Systems*, Kluwer Academic Publishers, August 1991.

[2] A. K. Mok, "The Design of Real-Time Programming Systems Based on Process Models", *Proceedings of the IEEE 1984 Real-Time Systems Symposium*, December 1984, pp. 5-17.

[3] E. Kligerman, A. D. Stoyenko, "Real-Time Euclid: A Language for Reliable Real-Time Systems," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 9, pp. 940-949, September 1986.

[4] W. Rossak, A.D. Stoyenko and L. R. Welch, "The Component Manager: A Hybrid Reuse Tool Supporting Interactive and Automated Retrieval of Software Components," *Proceedings of 1992 Complex Systems Design Synthesis Technology Workshop*, Naval Surface Warfare Center, Silver Spring, Maryland, July 1992.

[5] M. Sitaraman, L. R. Welch, D. E. Harms, "Influences of a Component-Based Industry on the Expression of Specifications of Reusable Software," *The International Journal of Software Engineering and Knowledge Engineering*, June 1993.

[6] R. A. Steigerwald and L. R. Welch, "Reusable Component Retrieval for Real-Time Applications," *Proceedings of IEEE Workshop on Real-Time Applications*, N. Y., N. Y., May 1993.

[7] A. D. Stoyenko, *A Real-Time Language with A Schedulability Analyzer*, Ph.D. Thesis, Department of Computer Science, University of Toronto, 1987.

[8] A. D. Stoyenko, T. Baker, "Real-Time Schedulability-Analyzable Mechanisms in Ada9X," *Proceedings of the IEEE*, January 1994.

[9] A. D. Stoyenko, V. C. Hamacher, R. C. Holt, "Analyzing Hard-Real-Time Programs for Guaranteed Schedulability," *IEEE Transactions on Software Engineering*, pp. 737–750, SE-17, No. 8, August 1991.

[10] A. D. Stoyenko, T. J. Marlowe, *Polynomial-Time Transformations and Schedulability Analysis of Parallel Real-Time Programs with Restricted Resource Contention, Journal of Real-Time Systems*, Volume 4, Number 4, Fall 1992.

[11] A. D. Stoyenko, T. J. Marlowe, W. A. Halang, M. Younis, "Enabling Efficient Schedulability Analysis through Conditional Linking and Program Transformations," *Control Engineering Practice*, Volume 1, Number 1, 1993.

[12] A. D. Stoyenko, L. R. Welch, "Response Time Prediction in Object-Based, Parallel Embedded Systems," to appear in *Euromicro Journal*, 1993.

[13] L. R. Welch, A. D. Stoyenko and S. Chen, "Assignment of ADT Modules with Random Neural Networks," *The Hawaii International Conference on System Sciences*, IEEE, Jan. 1993.

[14] L. R. Welch, *Architectural Support for, and Parallel Execution of, Programs Constructed from Reusable Software Components*, Ph.D. thesis, Department of Computer Science, The Ohio State University, December 1990.

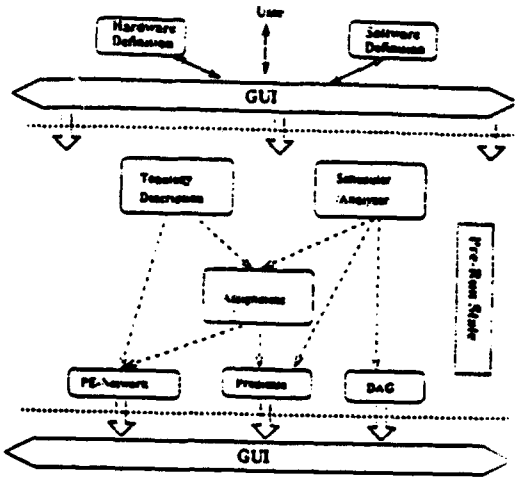


Fig 1 Pre-Run State

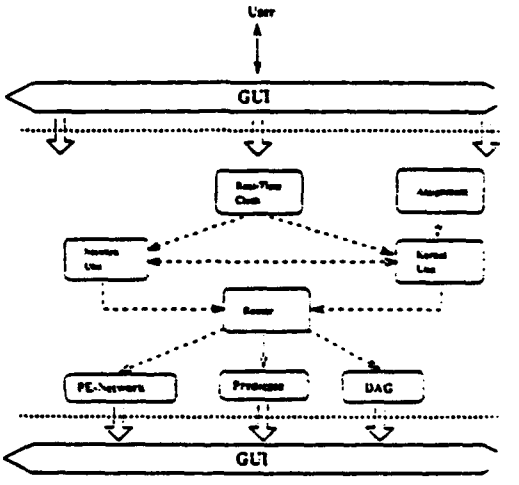


Fig 2 Run State

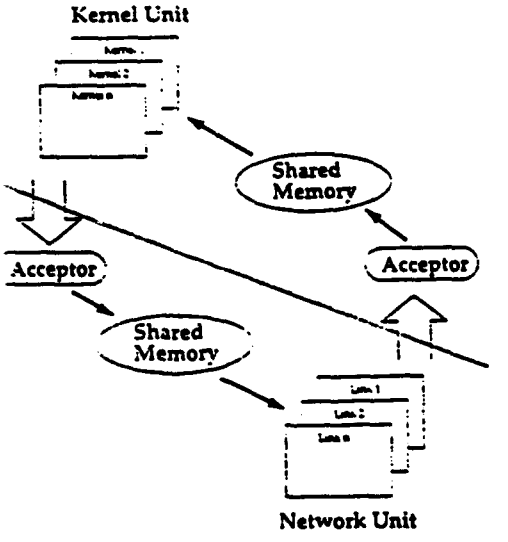


Fig 3 kernel-Network Interface

A Testbed for Prototyping Distributed and Fault-Tolerant Protocols

Farnam Jahanian
IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
e-mail: farnam@watson.ibm.com
tel: (914) 784-7498

Ragunathan Rajkumar
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
e-mail: rr@sei.cmu.edu

John J. Turek
IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
e-mail: jjt@watson.ibm.com

1 Introduction

The trend towards collections of powerful processors connected by a communication network has motivated the development of a number of distributed operating systems [3, 7, 8, 11] and many paradigms for building distributed applications [1, 2, 4, 6, 9, 10]. The shift from centralized large computer systems to closely-coupled clusters of processor which provide the illusion of a single system to the clients has been accelerated partially by the introduction of inexpensive micro-processors and dense memory chips. However, the complexity of software design, development, testing and integration of distributed systems poses a significant challenge particularly for mission-critical applications. Strict timing constraints and availability requirements often introduce additional complexity in the design of mission-critical distributed systems.

This paper introduces a testbed for prototyping fault-tolerant distributed systems. The testbed is structured as a set of services layered between the operating system kernel and the application software. The primary objectives of the testbed are: a) to allow development of complex mission-critical applications from a collection of building blocks with well-defined APIs, and b) to build the infrastructure for experimenting with different distributed protocols.

2 Building Blocks and Common API

We use the term *cluster* to refer to a collection of (perhaps) homogeneous processors connected by a communication network that functions as a server to external clients. The term server is used to denote a collection of processes (on the

cluster of processors) that provide the illusion of a single system to the clients. A server may be a command & control system, a real-time database machine, or an automated system for manufacturing and process control.' Such clusters are closely-coupled in the sense that a server is seen as a single system to the clients. The server software running on the processors hides the distribution and replication of the resources in the cluster. The client's view is defined by the server interface. One can view a closely-coupled server as a special case of distributed systems in which the collection of server software running on the processors makes the distribution of resources transparent to the clients.

Building and managing a reliable closely-coupled cluster that functions as a single coherent system is complicated by several factors:

- Asynchronous nature of communication (including random communication delay)
- Distribution/replication of resources across the system
- Changes to the set of processors that comprise the cluster due to a processor failing, leaving or rejoining the system.

In mission-critical systems, other requirements introduce additional complexity:

- Strict timing constraints imposed on response time of the system
- Dependability requirements the delivery of certain services
- Interoperability in an open system environment

The primary objective of this work is to provide a testbed for experimenting with primitives for building dependable servers. This is done by providing a collection of building blocks with well-defined APIs.

3 Overview of Testbed

The testbed consists of a collection of protocols for managing replicated and distributed resources in a system. It consists of six software layers, each exporting a well-defined interface to the other layers or to applications that are built on top of the testbed. Figure 1 illustrates the software layers in the testbed. Each software layer, referred to as a service, supports one or more protocols. A brief description of each service layer follows:

- *multicast communication service:* provides a reliable datagram communication service for sending a message to a collection of destinations. This service allows the exploitation of available communication protocols (e.g., Netbios vs. UDP) and possible hardware support (e.g., hardware broadcast facility) on a given system without exposing the implementation to the higher layer services.
- *processor membership and monitoring service:* provides a consistent view of the operational status of a group of processors in the presence of processor/process failures/joins and communication failures. Three membership protocols with varying degrees of consistency in the views of the members are supported.
- *clock synchronization service:* provides a bound on the deviation between logical clocks on processors in the presence of hardware clock drifts and failures.
- *reliable naming service:* provides a reliable service mapping the name of an object to a list of processors in the system. This layer supports multiple namespaces.

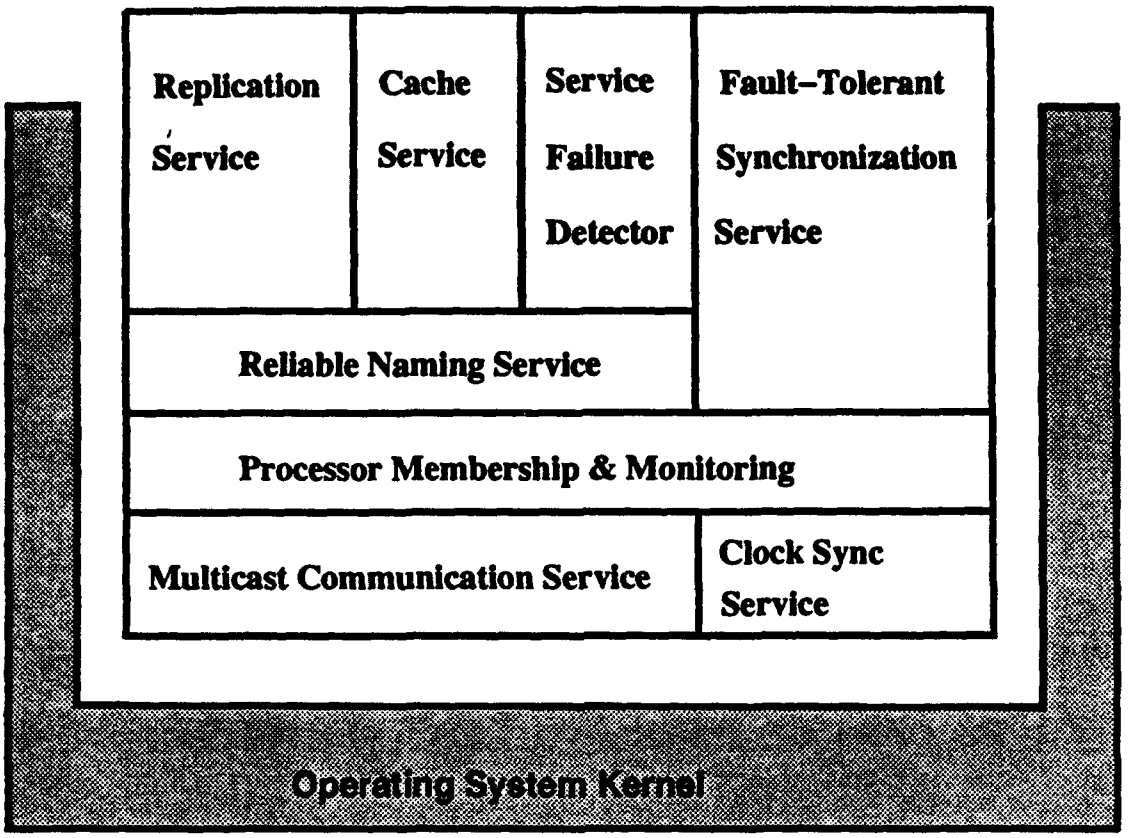


Figure 1: The Software Layers of the Testbed.

- *distributed cache service:* provides shared/exclusive access to remote objects with local caching. This layer supports multiple coherency protocols including cache invalidation and write-through policies.
- *replication service:* provides a mechanism for maintaining multiple copies of objects in a cluster. This layer supports several replication protocols with different consistency semantics for updating replicas.
- *distributed synchronization service:* provides fault-tolerant and scalable synchronization protocols for serializing access to shared/exclusive resources. The distributed synchronization service can recover from the failure of a lock holder/coordinator and communication failures.
- *service failure detector:* provides notification/query service for monitoring status changes of a collection of subsystems grouped together as a server. A status change to a group can occur because of a subsystem failure or an update to an application-defined status field.

4 Design Goals

The primary design goals of the testbed are application independence, portability to different operating systems and scalability to a large number of processors in a cluster.

Application independence/portability:

The current design and implementation of the services in the testbed ensures independence from the applications. The services are generic and are common to a range of servers that can be built on a cluster. Each software layer has an application programming interface. The implementation can be ported easily to other UNIX-like operating systems. [1] Since a large portion of the operating system dependencies are encapsulated in a communication layer, these services are easily portable to other workstation operating systems such as OS/2 and Mach. The layered design of the services allows experimenting with several protocols for each layer without rewriting a substantial part of the code. For example, in the current version of the software, three processor group membership protocols are supported (as described in [5]). The implementation of each protocol consists of less than 1000 additional lines of code in C, while the surrounding 5000 lines in the membership layer are untouched.

Performance:

Analysis and preliminary experimental results indicate that the current protocols in the testbed are scalable to clusters with moderate number of nodes. The current prototype is running in a cluster environment with 32 nodes. The layered design enhances performance by taking advantage of the hardware architecture of the cluster. For example, the membership and the replication protocols can take advantage of hardware multicast support if available in a cluster. Similarly, the existence of a physical shared memory among a set of processors would allow a more efficient implementation of certain protocols and avoid the need for some protocols. With the exception of the processor membership service, which is the layer gluing all others together, there is no overhead for unused services. Since many of the protocols in the testbed are intended for managing distributed resources and maintaining consistent copies of replicas, the scalability of these protocols is an important goal. Our approach to meeting this goal is discussed next.

Scalability of distributed protocols:

Distribution or replication of resources (or objects) frequently leads to performance bottlenecks in a loosely-coupled cluster. Local caching and accessing of shared objects is used to enhance performance, while cache coherency protocols are used to maintain object consistency. The protocols are designed for cases in which objects are replicated a small number of times, which enhances availability without compromising performance. In particular, we combine the replication protocol with a caching protocol completely transparent to the user of the service. These performance-driven tradeoffs allow the abstraction of a single reliable and coherent system to be scalable to a large number of processors, without a serious adverse impact on availability. Since our design philosophy is to replicate a small (fewer than 4) number of replicas for any object, this allows us to optimize the replication protocols. The obvious disadvantage of a small number of replicas is that the system may become unavailable in the event of multiple simultaneous failures, but such an occurrence is relatively rare in practice.

Weaker consistency semantics:

An important objective is to provide several protocols to satisfy different consistency requirements on multiple copies of objects. In many cases, updates to a replicated object must be atomic. For example, multiple copies of a system configuration must be kept consistent such that data integrity is ensured in the presence of failures. However, the consistency requirements of other objects, such as sensor data, are much less stringent, and a recent version of the object can be considered to be a good approximation of the latest version of the object. This is particularly true of dynamic attributes of an object which are updated only at periodic intervals. For example, if the load on a machine is queried, a recent value may be sufficient as opposed to the latest snapshot. Better performance can be achieved when the consistency requirements of an object can be relaxed. We are now experimenting with the notion of *periodic replication servers* for maintaining replicated objects in

[1] UNIX is a registered trademark of AT&T Bell Labs.

time-critical applications with strict timing constraints.

5 Concluding Remarks

The current implementation of this testbed has been built on a network of RISC System/6000 processors running AIX. [2] The modularity of its design allows portability to other workstation OS with relative ease. The current implementation has been built on several communication transport layers including the UDP datagram service, a multicast communication service and the NetBios datagram service. We are also experimenting with a transport layer on a very fast point-to-point communication network.

The software layers in this testbed are divided into eight distinct layers residing between the application and the operating system kernel. Each layer consists of one or more distributed protocols. Furthermore, an application programming interface cleanly defines the functions provided by each layer. The layered approach to structuring the software services has allowed us to experiment with different protocols and to support different semantics for a service in each layer.

Thus far this work has been focused mostly on asynchronous distributed protocol. With the exception of a synchronized clock service and the periodic replication service, the protocol in the testbed provide no hard real-time guarantees. We are currently investigating a variation of the proposed software architecture tailored as a testbed for experimenting in time-critical systems.

References

[1] K. P. Birman. The process group approach to reliable distributed computing. Technical Report TR 91-1216, Department of Computer Science, Cornell University, July 1991.

[2] A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[3] D. R. Cheriton. The v distributed system. *Communications of the ACM*, 31(3):314–333, March 1988.

[4] F. Cristian and R. Dancey. Fault-tolerance in the advanced automation system. Technical Report RJ7424, IBM Almaden Research Center, April 1990.

[5] F. Jahanian and W. L. Moran Jr. Strong, weak and hybrid group membership. In *Workshop on Replicated Data Management II*, Nov 1992.

[6] B. Liskov and R. Scheifler. Guardians and actions: Linguistic support for robust, distributed programs. *ACM Transactions on Programming Languages and Systems*, 5(3):381–404, July 1983.

[7] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch. The sprite network operating system. *Computer*, 21(2):23–36, February 1988.

[8] M. Rozier and et al. Chorus distributed operating systems. *Computing Systems*, 1(4), 1988.

[9] S. K. Shrivastava, G. N. Dixon, and G. D. Parrington. An overview of the arjuna distributed programming system. *IEEE Software*, pages 66–72, January 1991.

[2] RISC System/6000 and AIX are trademarks of IBM Corporation.

[10] A. Spector. Distributed transactions for reliable systems. In *Proceedings 10th ACM Symposium on Operating Systems Principles*, pages 127–146, Orcas Island, 1985. ACM SIGOPS.

[11] A. S. Tanenbaum, R. van Renesse, H. van Staveren, S. Mullender, G. Sharp, and G. van Rossum. Experiences with the amoeba distributed operating system. *Communications of the ACM*, 33(12):46–63, December 1990.

ARCHITECTURAL SYNTHESIS OF MISSION-CRITICAL COMPUTING SYSTEMS

Raed Alqadi Parameswaran Ramanathan

Department of Electrical and Computer Engineering
University of Wisconsin–Madison
Madison, WI 53706-1691.
parmesh@ece.wisc.edu, (608) 263-0557

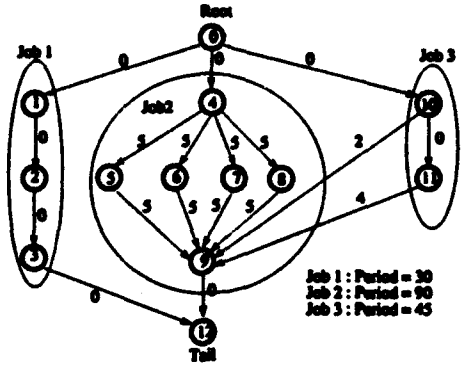
ABSTRACT

The complexity of mission-critical computer systems has grown rapidly in recent years. As a result, it is no longer possible to investigate different design options without the assistance of computer-aided (CAD) tools. In this paper, we briefly describe a synthesis system, called SHARMA, that is being developed at the University of Wisconsin, Madison. SHARMA (Synthesis of Heterogenous Architecture for Real-time Mission-critical Applications) is a set of design tools for architectural synthesis of computer controllers for mission-critical applications.

1 Introduction

A mission-critical application is comprised of several cooperating tasks. Each task may have constraints such as precedence, release time, deadline, resources, performance, and reliability. For instance, in a surface ship radar application [2], an incoming missile must be identified within 0.2 seconds of its detection. If necessary, intercept missiles must be engaged within 5 seconds after detection, and launched within 0.5 seconds within engagement. Failure to meet such timing constraints may have catastrophic consequences.

One way of meeting these constraints is to make use of a distributed computing system, which is a set of processors/nodes interconnected by means of a communication network. Designing such a system involves: (i) choosing a set of building blocks such as processors, interconnects and I/O interfaces, and (ii) mapping the tasks in the application onto these building blocks. Not all designs using these building blocks will satisfy the constraints of the application. Identifying designs which satisfy the constraints is very difficult due to the large number of alternatives that can be constructed using the building blocks. Computer-aided tools can simplify the identification of feasible designs by facilitating the search and evaluation of several alternatives. Presented below is an outline of an integrated set of tools, called SHARMA, for identifying feasible designs.



Task	Computation	Release Time	Deadline	Resource-types
0	0	-	-	-
1	10	0	30	1, 2
2	10	30	60	1, 2
3	10	60	90	1, 2
4	20	0	-	1, 2
5	20	-	-	3, 4
6	20	-	-	3, 4
7	20	-	-	1, 2
8	20	-	-	1, 2
9	10	-	90	1, 2
10	10	0	45	3, 4
11	10	45	90	3, 4
12	0	-	-	-

Figure 1: Example of a real-time application

2 Overview of SHARMA

SHARMA is a set of tools for synthesizing heterogeneous computer systems for real-time mission-critical applications. It is comprised of two main modules: a *synthesizer* and a *scheduler*. The synthesizer is responsible for selecting a set of modules from given building block components. The building block components may include processors of different speeds, sensors, actuators, interconnects, I/O interfaces, etc. The scheduler is responsible for verifying that the constraints of the application can be satisfied using the selected building block components. Since the components may have different costs, the main objective of SHARMA is to identify a low cost architecture that can meet the constraints of the application.

The application is specified to the tools as an annotated directed acyclic graph. The vertices of the graph represent the tasks and the edges represent the precedence constraints. Each vertex is annotated with the constraints of the corresponding task such as its computational requirement, release time, deadline, and a set of the required resources. Each edge is annotated with the amount of information exchanged between the corresponding pair of tasks. For example, consider the application in Figure 1. The application is comprised of three periodic jobs with periods 30, 90, and 45, respectively. Jobs 1 and 3 have only one task whereas Job 2 has six tasks. The figure shows the precedence constraints among all the tasks that must complete in the time interval 0 to 90 (i.e., the least common multiple of the three periods 30, 90, and 45). Note that, the task graph is also annotated with a

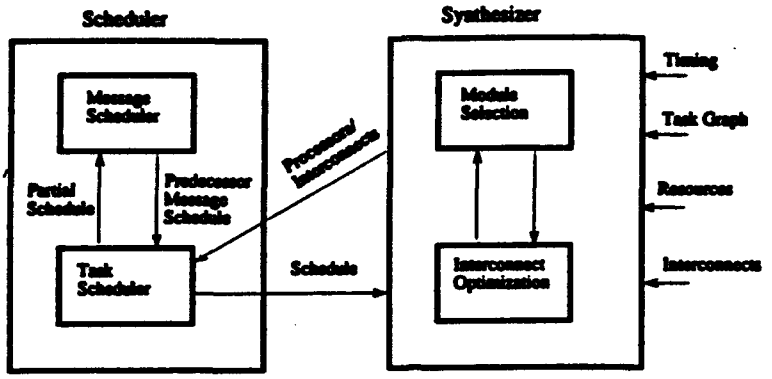


Figure 2: Block Diagram of the SHARMA system

table which specifies the computation requirements, the release time, the deadlines, and a list of processor types on which the tasks can execute.

In addition to the application, SHARMA is provided with a library of resources and building block components available for synthesizing a suitable architecture. The cost and performance is also usually specified for each component. From these inputs, the CAD tools in SHARMA choose suitable components, design an interconnection between them, map the real-time tasks onto the components, and identify a schedule that meets the constraints of the application.

To develop the tools in SHARMA, the synthesis process has been split into four major steps: module selection, interconnect optimization, task scheduling, and message scheduling. Each of the steps is being implemented as a separate CAD tool which interacts with the other tools to synthesize the desired system (see Figure 2). Presented below is a brief description of the approach used in these CAD tools.

2.1 Module Selection

The designer specifies the set of resources required for each task. For each resource, the designer also specifies a library of different types of the resource. These types differ in performance and cost. For example, a processor design library may contain a 20 MIPS processor for $20, a 30 MIPS processor for $40, and a 40 MIPS processors for $60. Module selection is the process of selecting the types and the number of units of each resource type required to satisfy the constraints of the application. The current version of our module selection tool handles multiple types of one resource (e.g. processor). Extensions which can handle multiple types of more than one resource are currently in progress.

Our module selection algorithm consists of two major steps. In the first step, an extension of the lower bound analysis in [1] is used to estimate the number of units of each resource type and the number of communication channels required to satisfy the constraints. Using these estimates, the algorithm constructs homogeneous architectures, one for each type, and then invokes the task and message scheduler to determine whether the constraints

can be satisfied. Once feasible solutions have been identified, the second step reduces the cost of the architectures by transforming them into heterogeneous systems. This is accomplished by recursively replacing the higher cost units by lower cost units until it is no longer possible to satisfy the constraints of the application. The lowest cost system which satisfies all constraints is chosen as the final architecture.

2.2 Interconnect Optimization

There are many different types of interconnect components such as buses, point-to-point links, and crossbars. The choice of interconnects should be based on the communication patterns and the communication delay constraints of the application which in turn depend on the allocation and scheduling of the tasks in the application. However, since allocation and scheduling of tasks cannot be done without the knowledge of the interconnection, choosing the optimal set of interconnects for a given set of building block components is very difficult, if not impossible.

The current version of our interconnect optimization tools is limited to selecting one or more multiplé access channels. A lower bound analysis [1] is first used to estimate the number of channels of a given type that is required to meet the communication needs of the application. A recursive technique is then used to lower the cost of the interconnects. This technique is similar to the one used in the module selection step.

2.3 Task Scheduler

Several algorithms have been reported in the literature for scheduling tasks in real time applications [3–6]. The main problem with most of these algorithms is that they are restricted to homogeneous processing units. Since we are dealing with a heterogeneous system, a new algorithm was developed for scheduling and allocating heterogeneous processing units. Currently, the algorithm assumes that the interconnection network is a set of multiple access channels where each processing unit can access any channel. Other types of interconnection networks will be dealt with in future.

The module selection and interconnect optimization steps provide the task scheduler an architecture on which the scheduling is to be performed. The task scheduler is an iterative list scheduling algorithm that repeatedly invokes the message scheduler to ensure that the communication network can meet the timing requirements of the synthesized schedule. The basic idea of our task scheduling algorithm can be explained as follows. Each task is initially assigned to a processor unit on which it can execute. This assignment specifies an execution time and the latest start time for each task. The assignment is then discarded and a new assignment and schedule is generated as explained below.

A task is considered ready for scheduling when all its predecessors have completed their execution. The ready tasks are ordered according to their latest start time. The ready task with the least latest start time is first considered for scheduling. The task is scheduled on the processor on which it can complete the earliest. To identify this earliest completion time,

all possible assignments for the task are considered. Each possible assignment generates a different set of predecessor messages that have to be scheduled on the communication network. For each possible assignment, the message scheduler is invoked to determine the earliest completion time of all the predecessor messages of the task under consideration. The least schedulable time after all predecessor messages have arrived plus the corresponding execution time for the task is the earliest completion time of the task on a given processor. The minimum earliest completion time over all possible processors is the time at which the task is scheduled; the task is scheduled on the corresponding processor. After a task is scheduled, the ready list is updated to possibly include the immediate successors of the just scheduled task.

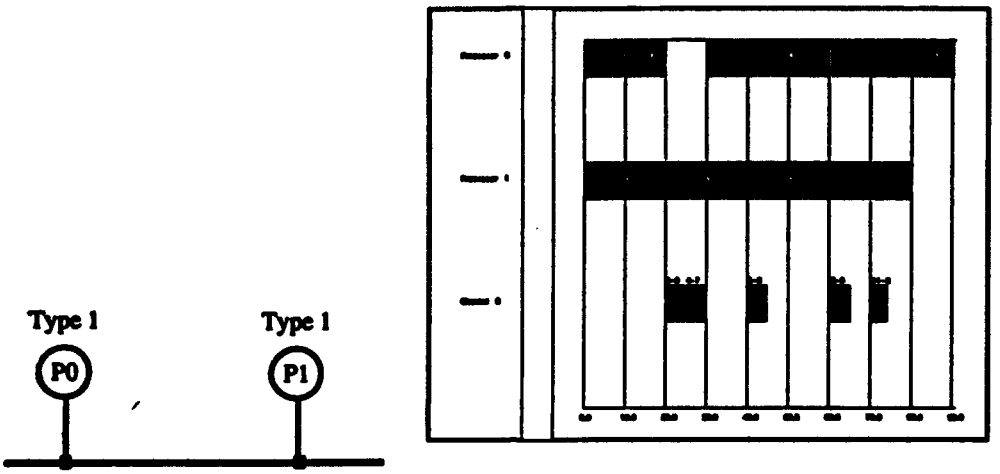
The algorithm continues in this fashion until all tasks have been tentatively scheduled. If some tasks do not meet their deadlines, then the whole process is repeated after recomputing the latest start time of all tasks based on the new processor assignment just generated. Note that, a new assignment results in a different execution time and different communication pattern for some tasks. Consequently, there is a change in the latest completion time of some tasks, which in turn, changes the priority order among the tasks in the next iteration. The algorithm terminates either when a feasible schedule is identified or when a designer specified iteration limit is exceeded.

2.4 Message Scheduler

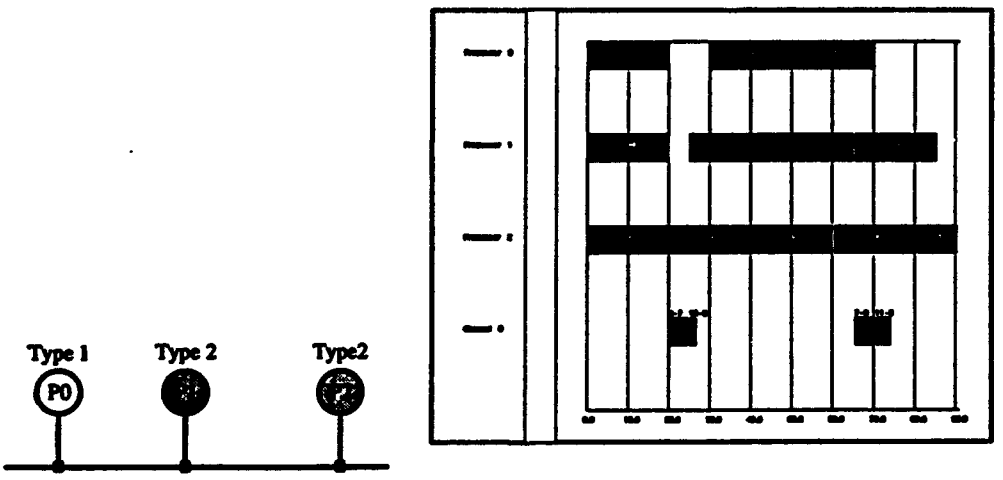
The message scheduler is invoked by the task scheduler to determine the earliest completion time of all predecessor messages of a given task. The message scheduler orders the predecessor messages according to their earliest start times. The messages are then considered for scheduling in the increasing order of the earliest start times. Each message is scheduled on the communication channel in which it can complete the earliest. To determine the earliest completion time, each channel is individually considered and the message is scheduled at the earliest possible time on that channel. The completion time on a channel is the earliest possible time plus the time required for communication. The message is finally scheduled on the channel with the minimum earliest completion time.

3 Preliminary Results

In this section, we present the results generated by the tools in SHARMA for two example real-time applications. The first example is the one shown in Figure 1 except that only two types of processors are considered. It is also assumed all tasks can run on either type of processor. Processor of type 1 has a normalized performance of 1 computation per time unit whereas processor type 2 has a normalized performed of 0.5 computations per time unit. The cost of a processor of type 1 is assumed to be 4 and that of a processor of type 2 is assumed to be 1. Further, it is assumed that the cost of a communication link is 0.5. Figure 3(a) shows a homogeneous architecture and the corresponding schedule which meets the constraints of the application. In this architecture, there are two processors of type 1 and no processor of type 2. The overall cost of this architecture is 8.5. Figure 3(b) shows



(a) Homogeneous architecture



(b) Heterogeneous architecture

Figure 3: Architectures synthesized by SHARMA for the example application in Figure 1

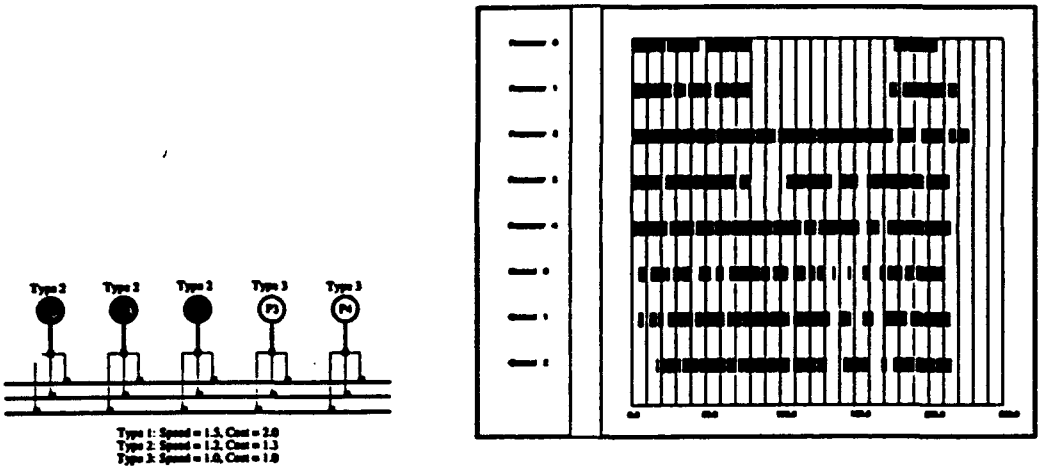


Figure 4: Architecture synthesized by SHARMA for a large example application

a heterogeneous architecture and the corresponding schedule which meets the constraints of the application. In this architecture, there is one processor of type 1 and two processors of type 2. Note that, the overall cost of this architecture is only 6.5 as compared to 8.5 for the homogeneous architecture.

The second example presented here is an application with 100 tasks. The precedence relation between the tasks were generated randomly. The tasks with no successors were assigned deadlines equal to the length of their critical paths on the slowest processor. The design library used in the synthesis processors had three types of processors with normalized performance of 1.0, 1.2, and 1.5, and normalized cost of 1.0, 1.3, and 2.0, respectively. The synthesized heterogeneous system and the corresponding feasible schedule generated by the tools in SHARMA are shown in Figure 4. Note that, the synthesis design has three processors of type 2 and two processors of type 3. Also, three communication channels were required to meet the constraints of the application.

4 Conclusion

We presented an overview of the set of CAD tools being developed at the University of Wisconsin-Madison for synthesizing heterogeneous computer systems for real-time mission-critical applications. Preliminary results obtained by these tools were also presented. Future work includes enhancing these tools and evaluating their performance on other applications.

References

[1] M. A. Al-Mouhammed, "Lower bound on the number of processors and time for scheduling precedence graphs with communication costs," *IEEE Transactions on Software Engineering*, vol. 16, no. 12, pp. 1390–1401, December 1990.

[2] J. J. Molini, S. K. Maimon, and P. H. Watson, "Real-time system scenarios," in *Proceedings of Real-Time Systems Symposium*, pp. 214–225, December 1990.

[3] K. Ramamritham, "Allocation and scheduling of complex periodic tasks," in *Proceedings of International Conference on Distributed Computing Systems*, pp. 108–115, May 1990.

[4] J. P. C. Verhoosel, E. J. Luit, and D. K. Hammer, "A static scheduling algorithm for distributed hard real-time systems," *Real-Time Systems*, vol. 3, no. 3, pp. 227–246, September 1991.

[5] J. Xu and D. L. Parnas, "Scheduling processes with release times, deadlines, precedence and exclusion relations," *IEEE Transactions on Software Engineering*, vol. 16, no. 3, pp. 360–369, March 1990.

[6] W. Zhao, K. Ramamritham, and J. A. Stankovic, "Scheduling tasks with resource requirements in hard real-time systems," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 5, pp. 564–577, May 1987.

An Intelligent Real-Time System Assessment Tool

Ed Andert Jr.
Conceptual Software Systems, Inc.
P.O. Box 727, Yorba Linda, CA 92686

Larry Peters
Software Consultants International Ltd.
P.O. Box 5712, Kent, WA 98031

ABSTRACT

A crucial requirement of many Navy systems is the need to perform critical functions within specified real-time deadlines during stress situations. Current system design support methodologies and tools are insufficient for the complexity encountered in complex real-time system development. Support techniques inadequately communicate requirements and do not assist in the complex task of identifying critical components and problem areas.

This paper discusses a prototype Intelligent Real-Time System Assessment Tool (ExpeR/T) that integrates requirements specification, system design, design analysis, and design evaluation. It provides the client and engineer with an automated tool that identifies critical and problem areas in a design and suggests improvements. The tool approach utilizes an innovative combination of expert system analysis, graphical system design, and design modeling/simulation. The project leverages off of the commercial ProTEM™ graphical system design diagramming and simulation tool.

INTRODUCTION

The ExpeR/T tool performs a variety of functions. The functions of the system are segmented into the following categories:

- *Requirements specification* - allows the specification of requirements through an expandable menu system. Requirement details are specified in a template that allows linkage to design elements.
- *Design performance and problem analysis* - is made up of performance analysis, critical component identification, and problem analysis. Performance analysis includes a time criticality factor/strategy. Critical component identification includes time focus factor, critical path analysis, and critical state analysis. Problem analysis consists of condition search and rule-based analysis.
- *Problem and critical component correction assistance* - provides assistance for problems and critical areas identified by the design performance, critical component, and problem analysis modules. An expert system knowledge base is used to designate appropriate designer assistance.
- *Graphical design and simulation* - allows system developers to use a graphical design diagram and simulation tool. This component is based on ProTEM advanced Petri-net-based commercial software.
- *Automated conversion* of DFDs and compatible design diagrams to ProTEM Petri nets - gives compatibility with complementary system design methodologies.
- *Design quality evaluation* - provides metrics for the evaluation of system designs. The analysis techniques include behavior analysis, resource loading, and weighted/combinatorial factors.

APPROACH

Current complex real-time systems development methods, and the computer-aided design tools supporting them rely heavily on an informal and inaccurate process for communicating requirements among the client, developing contractor, user, and contracting authority. This has resulted in systems whose performance is inconsistent with requirements and which require expensive revisions late in the development life cycle.

Researchers that have noted the inadequacy of current tools for complex system development have suggested second generation metaCASE tools that capture requirements, simulate system designs, and advise designers on-line [Forte 92]. The Intelligent Real-Time System Assessment Tool (ExpeR/T) described here addresses these needs. The tool is designed to address the factors that contribute to complex system development problems. It also has the flexibility to address new problem areas as they are encountered. ExpeR/T provides an integrated means of defining, designing and analyzing complex real-time systems which incorporates formal methods, informal methods, evaluation schemes, and engineering experience. But more importantly, it provides the client and engineer with suggested improvement for

detecting problems with a system model. One of ExpeR/T's primary objectives is to aid in the identification of critical areas early in the development life cycle to provide a strategic advantage to both client and developer.

ExpeR/T's larger capability set is the result of an innovative combination of methods, techniques and concepts which are currently treated in the technical literature as being disjoint and separate. For example, software structured analysis through data flow diagrams (DFDs) and compatible representations encapsulated in CASE tools, systems analysis using a variety of formalized techniques, and requirements specification are all treated as disjoint concerns. ExpeR/T combines these concepts through integrated English-like requirement specification, graphical system design, design analysis, and design modeling/ simulation. It enables engineers to realize their full potential by assisting them in focusing their attention on those aspects of system definition, analysis and design for which human beings are most well equipped, that is, the solution of poorly structured, poorly understood problems. It provides an objective means of evaluating these solutions by identifying and quantifying critical areas, as well as suggesting improvements.

Requirements Specification

The requirements component allows selection of "factors" from a menu. Selecting a factor from the menu leads to a list sub-factors allowing further selection, etc. This hierarchy is demonstrated in Figure 1. The taxonomy of factors and subfactors is essentially derived from those defined in [Nguyen 92].

Details for a subfactor can be defined using a "template" in the spirit of [Nguyen 92], but more with a looser format. This template also allows definition of a connection to design items (i.e. Petri-net transitions) as shown in Table 1. For example, if a requirements function is defined as beamformer, then the performance criteria of 23 beams per second can be mapped into the group of Petri-net design diagram items that implement the beam former. Potentially, the Petri-net model can generate an actual beams per second performance number from the group of design items fulfilling the requirement (given a set of input circumstances). The actual performance can then be compared to the specified beams per second which generates a performance metric. If the function is defined as "system" signifying the entire system, then the overall Petri-net model would be used to generate performance numbers.

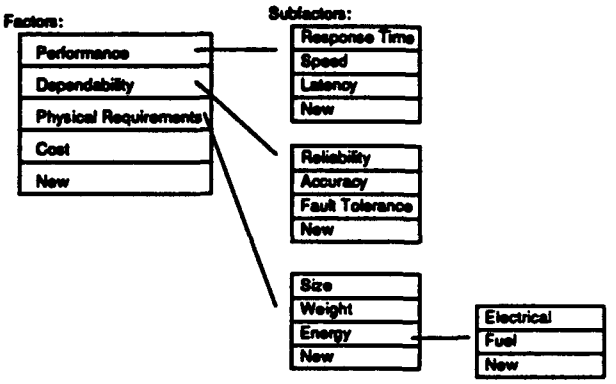


Figure 1: Requirements taxonomy hierarchy example

Name:	Reliability
Function:	beam former
Type:	Numeric
Range:	> 0
Units:	Seconds
Measure:	Mean time between failures (MTBF)
Connection to Design:	Transition XY, Transition XYZ
Additional:	Rationale - life critical function

Table 1: Requirement subfactor detail template

The ExpeR/T program is driven by a menu user-interface. The level 1 menu is shown in Figure 2. Submenus to the Requirements option successively proceed down the requirements subfactor hierarchy.

Performance Analysis

The approach for performance analysis in ExpeR/T allows definition of performance characteristics for design components (transitions) in Petri-net design diagrams. Specification of performance characteristics for requirement factors/subfactors are also allowed. At some point in the

system development process the designer can connect requirement functions to Petri-net transitions. The performance analysis component then takes the performance model generated by the Petri-net and compares it to the required performance for the appropriate function.

An expert system rule set in the performance analysis component analyzes the relevance and criticality of the model analysis deviations from the specified performance requirements. These rules identify deviations that indicate important problems and report them as alerts to the user. For example a large set of inputs might be applied to the Petri-net and in some outlying cases, 1% total, response time requirements cannot be met. It this case a rule can specify that the user only be notified in a "warning" in a detailed hard-copy listing of the analysis as opposed to an on-line alert that is also highlighted as an alert in the hard-copy. For example:

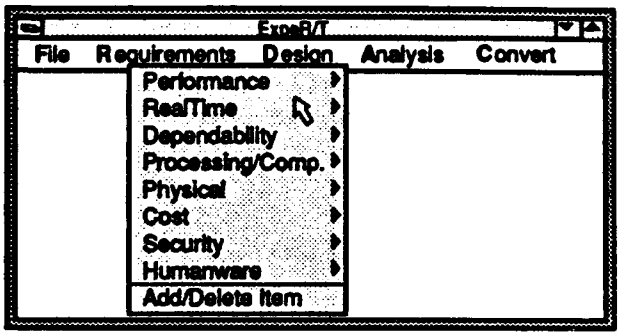


Figure 2: Level 1 menu with pulled-down level 2 menu.

IF response requirement deviates from model simulation
AND quantity of deviations is less-than 5% of test cases
THEN report deviation in warning only

Critical Component Identification

Critical component identification analyzes the results of the performance analysis module to identify components of the design that create bottlenecks in processing or are in the critical path. Bottleneck components are those which cause higher speed processing in associated components earlier in the processing stream to wait for processing in the subject component. A critical path component is one which stands alone in an important processing stream which could potentially disrupt processing in event of failure or low-performance.

There are three specific "figures of merit" that are utilized to identify critical components and processing bottlenecks. These include the time focus factor, critical path analysis, and critical state analysis. Time focus factor identifies those 20% or so critical components that are responsible for 80% or so of system execution time. With critical path analysis, simulation determines which paths (patterns of state traversals) occur most and least frequently. The most frequently occurring will implies a set of components which require the utmost care to ensure system integrity. Critical state analysis uses simulation to help identify those states that occur most frequently which indicates the most active and therefore critical components. Critical state analysis is further enhanced by an expert system program that determines critical components. The expert system rules adjust the component identification based on the degree to which critical states occur more frequently than other states.

Problem Analysis

This analysis identifies problem areas in a system design. This problem analysis is performed by algorithms and expert system rules. The areas of analysis performed include condition search and rule-based analysis. Condition search enables the designer to specify a set of unwanted design conditions which are searched for in the Petri net design. For example, if Slot 7 represented the condition, "EjectionSeatArmed" and Slot 41 represented the condition "WeightOnWheelsTrue", the user would select these two conditions and the tool determines if this, potentially unsafe phenomenon, could occur. Rule-based analysis utilizes requirements to identify problems such as violated reliability constraints, unheeded error recovery considerations, and excessive cost factors. The design diagram (Petri net) supports the definition of needed characteristics such as error recovery and cost.

Problem and Critical Component Correction Assistance

Problems and critical areas identified by the design performance, critical component, and problem analysis modules is further processed to report applicable designer assistance. A rule set includes knowledge about how to resolve problem areas and modify designs to incorporate these problem resolutions. An example of the type of rule that provides correction assistance is as follows:

IF component_x is identified as a bottleneck
AND component_x is software
THEN suggest dataflow and processing division onto two hardware resources

Graphical Design and Simulation

Commercial ProTEM capability is incorporated into the ExpeR/T tool to yield real-time system graphical design and simulation facilities. The supported Petri net methodology is based on Petri net theory and is the result of extending this theory in order to make it more practical and effective in complex systems development. This extended form of Petri net technology enables users to model all facets of complex systems including sequentiality, concurrency, asynchrony and specific capabilities for real-time systems such as priorities and timing. The following section discusses the Petri net objects and rules in slightly more detail.

Automated Conversion of Design Diagrams to ProTEM Petri nets

Dataflow and other design diagrams in all of their various forms and variations represent the most widely used means of describing real-time and other software systems. Although they have proven themselves to be valuable in describing the information flow relationships within a system, they do not contain sufficient detail to enable an engineer to gain insight into the performance of the modeled system. The technology required to do that is available in the form of Petri nets.

The advantage that Petri nets in their extended or enhanced form (e.g. the form used in the ProTEM system) have over data flow diagrams in the area of real-time strategic and tactical systems include their ability to model parallel and asynchronous operations, account for probabilities, portray priorities and monitor and detail the utilization of resources (e.g. people, devices, software components).

The biggest problem facing anyone who wishes to use Petri nets in any form is the fact that the population of objects in the net is not easily surmised from a cursory inspection of a system level diagram. The paradox which exists here is that data flow diagrams are easy to use and understand but do not provide sufficient information for us to be able to determine whether or not a particular system design will work and if so to what extent while extended Petri nets can tell us what we need to know but they are not intuitively obvious. The purpose of this design is to detail how the gap can be bridged.

Petri nets are composed of Tokens (indicators of a condition being true), Places (used to specify the status of a condition) and Transitions (processes which transform one or more Tokens which w.e input to the Transition into more or one Tokens on the output side). Extensions to Petri net technology have been instituted [Peters 92, Peters 93] which transform Petri nets into a powerful tool for modeling real-time systems. These enhancements include timing, token typing, multiple behaviors, non-homogeneous places, priority and resources. An example of one type of Petri net graphics (without attribute details) is presented in Figure 3.

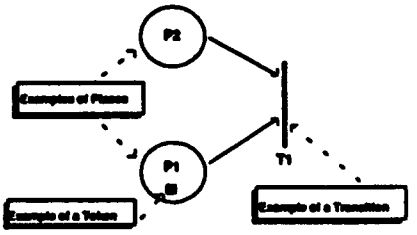


Figure 3: Example of one form of Petri net graphics

The process of transforming DFDs and other more compatible design diagrams into ProTEM Type Petri Nets (ProNETs) may be summarized as shown in Figure 4 and as detailed in the following steps:

- Identify the population of processing objects, Transitions, which comprise the system
- For each object, determine what conditions and resources are necessary for the Transition to execute

- Document the conditions in the form of Places and form a network.
- When necessary, make assumptions regarding the execution time of each Transition and refine this after making some initial single runs to validate the network

Design Quality Evaluation

The use of several software design quality evaluation figures of merit can greatly enhance the software engineer's ability to quickly and cost effectively assess the relative merits of one design over another. This is true of both new, yet to be built systems, and existing ones. Figures of merit included in ExpeR/T that enable this evaluation rely heavily on the results which the ProTEM Petri net makes available. Design quality evaluation figures of merit in ExpeR/T include behavior analysis, resource loading and weighted/combinatorial factors.

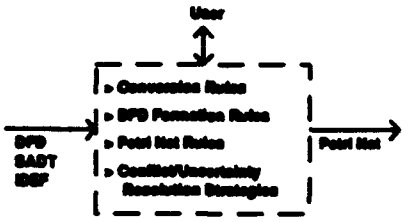


Figure 4: Overview of design diagram to ProNET conversion

Behavior analysis relies on the observation that the wider the range of behaviors possible by a component, the higher the likelihood that the component will fail [Halstead 75]. This is because multiple behaviors imply poorly thought out pseudocode or logic and/or complex interfaces to other components. Using simulation runs, behavior analysis breaks out and assigns overall system behaviors to individual system components. Inordinately large behavior factors for some components imply further investigation into decomposing the component is warranted.

Resource loading analysis measures the observation that use of a resource (e.g. communications buss) by a component constitutes a potential blocking situation. One way to avoid this is the use of a prioritization scheme whereby the highest priority tasks will gain access to the needed resources when those resources are needed. Two important factors to consider here are 1) What makes a component a high priority component - what it does or what it is processing? and 2) Can preemption cause a failure?

By multiplying or combining some of the individual analysis factors mathematically, the differences between and among various system components can be made more pronounced. Weighted/combinatorial factors can include multiplying the figure of merit for the most used component by the number of times it gets preempted or it preempts another component.

CONCLUSIONS

During a Phase I feasibility study, we successfully developed the conceptual requirements, high-level design, and a proof-of-concept prototype for the ExpeR/T intelligent real-time system assessment tool. The prototype is operational with some features completely implemented. It shows that the system design and assessment tool concept is viable and that its full implementation is feasible. The benefits of successfully developing the ExpeR/T tool will be improved complex real-time systems development through automated requirements specification, systems design, critical area identification, and design evaluation. ExpeR/T can be applied to computer systems development for DoD, Federal agencies, and commercial industries including avionics, nuclear systems and telecommunications.

REFERENCES

[Forte 92] Forte and Norman, *A self-assessment by the software engineering community*, Communications of the ACM, April 1992.

[Halstead 75] M. H. Halstead, *Software physics: basic principles*, IBM Research Report, RJ1582, Yorktown Heights, NY, 1975]

[Nguyen 92] C. M. Nguyen and S. L. Howell, *System design factors*, Proc. 1992 Complex Systems Engineering Synthesis and Assessment Technology Workshop, NSWC, Silver Spring, MD.

[Peters 92] L.J. Peters, *Design methods for real-time embedded systems: simulating execution times and state sets*, Fourth International Conference on Strategic Software Systems, Huntsville, AL, March 1992.

[Peters 93] L.J. Peters and R. Schultz, *The application of petri nets in object oriented enterprise simulations*, 26th Hawaii International Conference on System Sciences, Maui, Hawaii, January 1993.

An Environment for Analysis of Parallel Systems (EAPS)

Mohsen Pazirandeh
President, Innovative Research Incorporated
180 Cook St., Suite 315, Denver, Co. 80206
(303)-321-4955

Oliver McBryan
Professor, Computer Science Department
University of Colorado, Boulder, Co 80303

Areas of interest

Dr. Mohsen Pazirandeh is the President of Innovative Research Inc. (IRI). IRI is a small aerospace company engaged in design of applications and assessment of performance of parallel systems. He has a doctorate in Mathematics and Operations Research from University of California.

Dr. Oliver McBryan is the professor of Computer Sciences and the Director of Center for Applied Parallel processing at the University of Colorado at Boulder. He received his doctorate in Mathematics from Harvard University.

1. Introduction

This paper discusses the design of an environment for evaluating performance of parallel systems. It resulted from a Phase I research project funded by the Naval Underwater Systems Center (NSWC) under the SBIR program. Three technical objectives were identified: (1) find a convenient way to describe and synthesize a large class of applications, (2) similarly, define a method to describe and synthesize a large class of parallel systems, and (3) find a convenient method for mapping application programs to the system nodes and predict the performance of the resulting system. In addition to showing the feasibility of the above objectives, the research resulted in the development of a prototype of the environment. The prototype, based on an Innovative Research design evaluation tool, established the feasibility of the first two objectives and developed the outlines of the approach for the third. This paper discusses these findings and plans for its Phase II development. A proposal for developing many of the functionalities discussed in this paper has been submitted to NSWC.

2. Background

It is becoming increasingly evident that parallel systems will play a major role in meeting the computational needs of large scientific, military, and commercial applications. This is partially due to the rapid increase in the processing requirements of large applications which are growing at a rate faster than is provided for by the introduction of new processor generations. A simple example serves to illustrate the point.

To solve a two dimensional Poisson equation using finite differences on an nxn grid requires the solution of a set of linear equations of the type $Au = f$ where A is a square matrix of dimension n^2. The number of operations using the crudest Gaussian Elimination solution technique will be of the order of n^6. For a problem of reasonable size, say n=1,000, the number of operations is therefore of order 10^{18}. The magnitude of this problem is realized when we consider that a fast CRAY Y-MP performs at 2 GFLOPS, i.e. $2x10^9$ operations per second. Thus to solve the problem on this machine requires 317 years. Most real applications are more complex than this example. For example, solving a three dimensional Poisson problem increases the computational requirement and solution time by a factor of n^3. Of course, there are a number of techniques (e.g. iterative solution or FFT based methods), and properties of the problem (e.g. sparsity of the matrix) that can significantly reduce the processing time - to as low as $O(n^3 \log(n))$ operations in this special case. However realistic numerical simulations often require the repeated solution of such a problem perhaps thousands of times (e.g. the number of timesteps in a fluid simulation). Further, the memory requirements of the problem are very high and will limit the choice and performance of an architecture. This simple example illustrates that even on the fastest vector machines, it is quite difficult to solve large problems in a reasonable time. Thus, alternative options must be considered.

The most promising alternative so far has been parallel systems. A number of such systems have been introduced and the number is increasing rapidly. Recent experience has shown that parallel systems are no longer entirely experimental, and that useful systems have already been developed (e.g. the Connection Machine CM-2 and CM-5, Intel iPSC860, etc.). Furthermore, it has been shown that they can be used to solve real problems (e.g. linear algebra, partial differential equations, etc.) and at performances many times faster than a conventional vector computer such as the CRAY YMP/8 (reference [6] describes 24 Gflops computations on a CM-5). However, it has not been shown that such systems can be used as general purpose computers, and in fact they may never be. The relevant question is not whether a Teraflops system can be built, but rather how such powerful systems can be used effectively. In many cases only a small fraction of a system's potential power is utilized. Poor performance of such systems can be attributed to one or more of the following causes:

- The incompatibility of the application and the architecture is a major cause of poor performance. For example, to solve a small to moderate sized set of linear equations, LU decomposition may work well on vector machines such as CRAY, while the same algorithm may perform poorly on a massively parallel machine. Generally, non-structured applications with non-homogeneous subproblems are poor candidates for implementation on SIMD massively parallel systems which are most efficient when performing identical lock-step operations.
- The application's decomposition into tasks and their mapping onto processors have not been performed optimally. To fully exploit the potential power of these systems the application must be decomposed with maximum parallelism into tasks (grains) and mapped to processors in some optimal fashion. The decomposition strategy and the subsequent assignment of tasks is a major determinant of system performance, since the order of processing determines how tasks will communicate with each other and how synchronization among tasks will proceed.

3. Benchmarking

In the absence of good methodologies, benchmarking has been widely applied as an evaluation tool. But benchmarking is highly application specific and its results cannot be generalized. In most cases, performance can vary widely and the benchmarking results must be manually manipulated to *optimize* performance. The basic idea of benchmarking is to define a number of kernels (or pieces of code) which can represent a wide range of applications. The character and the choice of these representative codes can vary widely, but they fall into four categories: synthetic such as Whetstone and Dhrystone, kernels such as Livermore Loops, algorithmic such as LINPACK, and application such as Perfect Club. Benchmarking has gained wide acceptance because the alternative approaches of analytical modeling or discrete event simulation are not feasible beyond the simplest examples, and do not offer much hope. Analytical models are too difficult to build and generally not very accurate. Discrete event simulation quickly becomes too complex and unwieldy. But benchmarking, in its present form, also fails to address several important issues it was originally designed for. Although considerable attempts have been made to produce benchmarks which are more representative of real workload, benchmarking still remains inadequate as an evaluation tool.

Thus, in spite of its widespread use, benchmarking is not the method of choice for measuring (and more importantly predicting) the performance of an application on a parallel system. This is especially true for the end users whose primary interests are the performance of their applications.

4. An Environment for Analysis of Parallel Systems (EAPS)

In view of the above discussion, we proposed to determine the feasibility of developing an environment that can be used to conveniently define parallel architectures, design and decompose applications, map their tasks into the system nodes, and evaluate the performance of the system. Major consideration in such undertaking is that it must serve a diverse group of users with varied background and expertise who will rely on the environment to evaluate the performance of an application. The development of the environment must proceed with this in mind, and must possess tools required to support these and other needs. Flexibility is an important consideration as the environment will be used to analyze a varied class of architectures. Flexibility has a broad interpretation, covering the ability to define a wide range of architectures, applications, performance measures, and operational concepts (e.g., various types of scheduling strategies, synchronization scenarios, parallel algorithms, etc.). Flexibility is also reflected in the number of options the user will need to design and execute a model. The definition of events, identification of resources to be analyzed, partitioning of the problem, the granularity of the tasks, and their assignments to processors are just few of these options essential to developing flexible models.

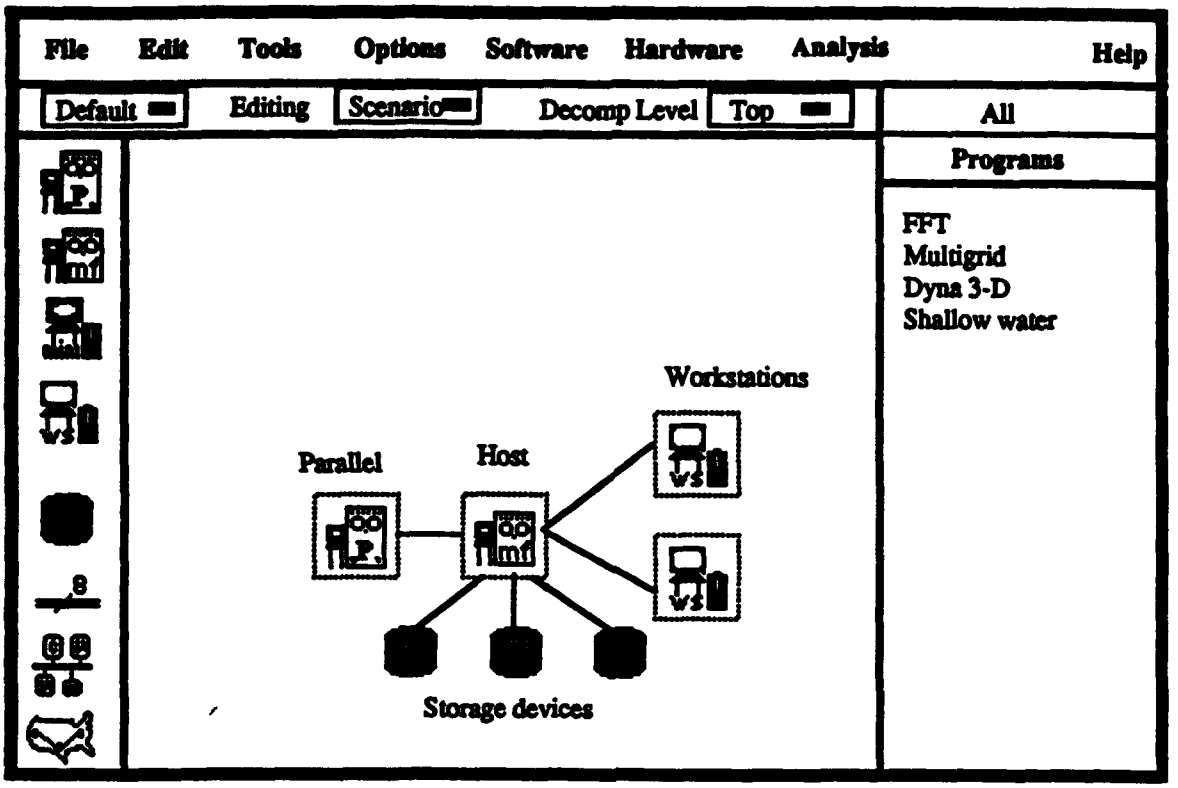


Figure 1. Opening window of EAPS

4.1. Parallel System Definition

All system elements are defined hierarchically. At the highest level a parallel system is a node in a larger system, and thus can be connected to a front-end processor (host), storage devices, or workstations. At this level one needs to only specify the global attributes of the system such as name, speed of processor at a node, speed of communication processor at a node, number of virtual processors per node, number of nodes, memory per node, network, protocol, etc.

The second level decomposition of an architecture can be done either by defining a new decomposition, using a previously defined decomposition, or using a library decomposition. Decomposition of a parallel system consists of defining the node structure and applying it to various nodes, specifying the communication network, and assigning application tasks to nodes.

Two forms of assignment can be performed: node and application decompositions. Two grids are provided. One to specify the nodes a given (node) decomposition apply, and the other specify the nodes a program is assigned. Initially, each gird represents all nodes of the parallel system, and hence a program (or a node decomposition) assigned to the system is assigned to all nodes. The assignment of an entity can be applied to a subclass of nodes by progressively partitioning the grids. Each action will partition the nodes into two equal parts, and thus after N actions, each partition represents $M*2^{-N}$ nodes, where M is the total number of nodes. The partitioning action can be reversed. The assignment of an entity (a node decomposition or a program) to a subclass will assign it to all nodes within that subclass. This capability allows for the representation of SIMD, MIMD, SPMD, and mixed architectures. Selection of a task will highlight its partition. An alternate way of partitioning the nodes, and arbitrary decomposition of a system with arbitrary number of nodes in each subclass will be accommodated via *scripting*

Scripting is a mechanism whereby a previously created script file may be provided which defines a detailed node decomposition, assignment, and/or network description. Also it is possible to ask the system to generate a script for any constructed node, including builtin defaults or library choices. Scripting is a powerful and versatile utility in EAPS with a number of key applications. Its most important use is in assigning decomposed nodes and application programs to arbitrary groups of nodes. Scripts are sets of code developed by the user specifying how system entities (programs, node decompositions, etc.) should be assigned. EAPS will provide the capability for writing Scripts and can read (i.e. execute) them upon user's command. For Phase II, scripting will be limited to the development and the execution of a previously defined set of code to assign programs or a node decomposition to an arbitrary class of nodes.

The assignment of a decomposed node (or a program) to a group of nodes is accomplished by selecting the entity and assigning it to a desired partition of the appropriate grid (which is then highlighted). This enables the user to group nodes into a number of subclasses each with different decomposition. Arbitrary partitioning, not a function of powers of two, is handled either graphically (i.e. by enclosing the desired set of nodes) or via Scripting. An existing decomposition can be reviewed and edited, and put in a library and made available to other users.

The *specific network* descriptions are predefined and cover a range of standard commercial architectures. These will include Intel Paragon, Kendall Square KSR1, CRAY MPP and Thinking Machines CM-5. The inclusion of a number of specific descriptions allows the system to be used as a training tool by both vendors and users of MPP systems.

Library modules are provided for a range of standard networks. These are of two types: generic and specific. *Generic networks* describe rings, meshes, torii, hypercubes, cube-connected cycles and other standard configurations. Hierarchical networks may be built from these units. For example a network such as the KSR1 or SUPRENUM-1 is a two-level hierarchy - using a ring or bus to interconnect a cluster (Ring:0 in the KSR1, or Cluster bus in the SUPRENUM-1) and using a further ring (Ring:1) in the KSR to connect many Ring:0, or a grid of busses (SUPRENUM Bus) in SUPRENUM to interconnect clusters. Network descriptions may be provided either globally, through a script file, or locally by providing node interconnectivity information. Global networks can then be specified by replication. Networks require several parameters for a complete description. These include connection bandwidths, communication latency, message buffering capabilities (number and total size), support for asynchronous communication, ability to overlap communication with computation and ability to communicate simultaneously on multiple channels. All of these characteristics together specify a network.

Another critical aspect of MPP design is the communication protocol between nodes. At the lowest level, protocols may be either store and forward or worm-hole routing. At a higher level one needs to support message typing. We intend to support several of the current protocols as predefined modules. These will include PVM, EXPRESS, PARMACS, MPI and possibly Intel NX2. Other standards will be added as they appear.

Communication protocol will play the same role as processor instruction set does for CPU design. At the highest level, communication for a program or subprogram will be represented in a simplistic way in terms of expected data volume and destination. At a lower level, individual communication operations will be recognized, and simulated taking into account whether asynchronous capabilities are available.

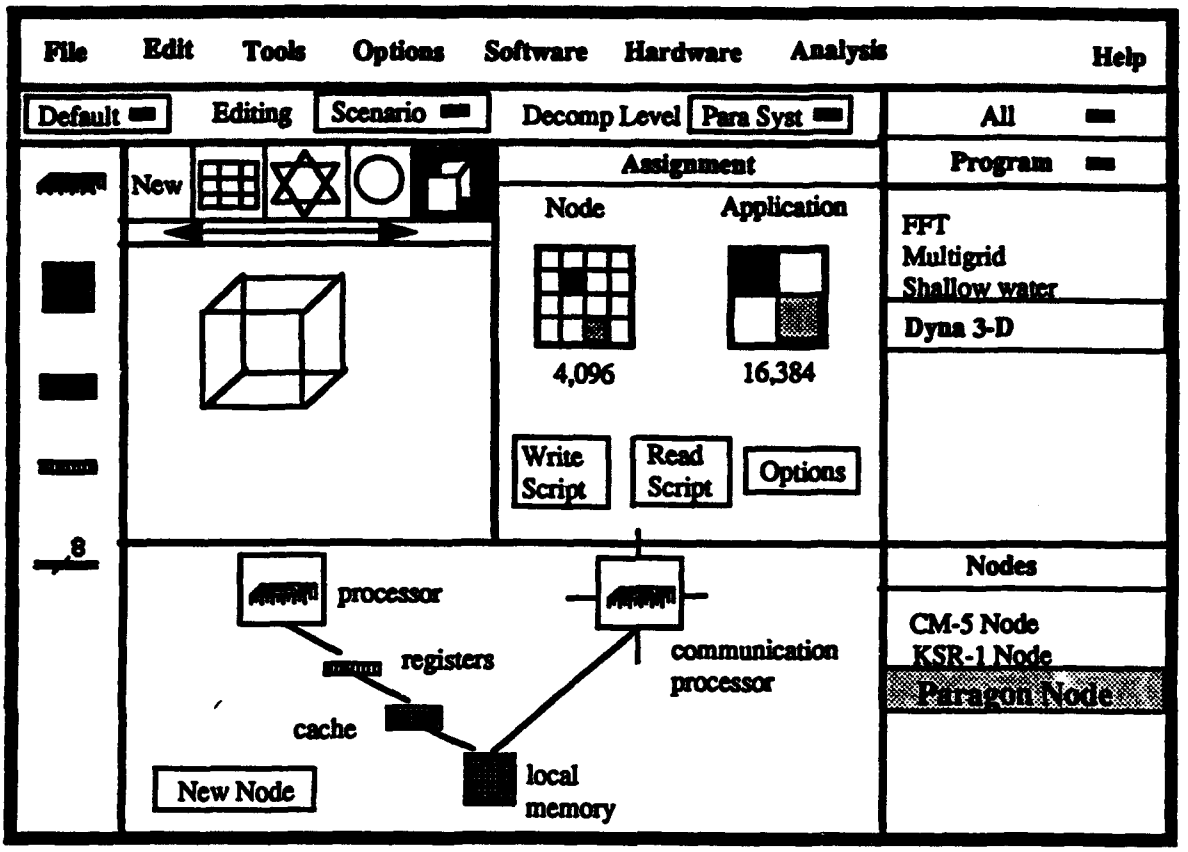


Figure 2. Decomposition of a node of Paragon

4.2. Application Decomposition and Synthesis

Similar to the architecture, the application is defined hierarchically, i.e. it can be defined as an entity and decomposed in terms of a number of programs (or kernels). The kernels themselves can be further decomposed in terms of other kernels. The idea of defining application kernels is not new (e.g. Bailey, D., et. al. [2], and Saavedra-Barrera, R., et. al. [13], among others). SES Workbench expands this capability by providing a limited capability of defining kernels. Our approach differs in a significant way:

Most authors have developed the idea of using kernels in the context of benchmarking. They have attempted to identify a number of *fixed* kernels to represent a large class of applications (mostly scientific). Thus, the user can only use the available kernels to synthesize an application without the ability to alter their characteristics, e.g. change their attributes, decompose them, redefine them, or add to them. We allow the user to define any number of kernels, define their attributes in a number of different ways, manipulate them, decompose them, combine them, and use them freely.

An application can be represented as a single entity and analyzed as such (though it may not be very interesting or useful). Its attributes can be specified in a number of forms. It can be assigned to any node or all nodes of the architecture. Such analysis may be useful in understanding the behavior of a piece of code uniformly assigned to all nodes of a system. Programs, tasks, or functions (collectively called kernels) are the building blocks of the application and can be defined in two different ways: by choosing *program* from the *software menu* or by double clicking on an existing program in the *application* window.

4.3. Performance Analysis

Performance analysis is the process of applying performance metrics, such as the ones described above, to an application and then analyzing the results. There are three basic strategies that can be incorporated: analytical, benchmarking and simulation analyses. One purpose of performance analysis is to be predictive: an engineering design project may need an advance estimate of the computational resources it will need to run specific applications with certain grid sizes. A second use is to optimize performance by analyzing the efficiency of an existing code. An effective tool will provide useful information in both of these situations. The analysis component of EAPS is based on the desire to combine the best features of benchmarking, analytical modeling, and discrete event simulation to enable an end user (an analyst or a scientist) to evaluate (and predict) the performance of an application on a parallel system.

4.3.1. Purpose of analysis

The analysis of a parallel system is performed to serve two distinct purposes:

a. *Application performance*. This reflects the application's performance and is of interest to the user. Metrics such as response time, execution time, elapsed time, and speedup quantify the results of this form of analysis.

b. *System capacity*. This form of analysis quantifies the total system capacity and is usually represented in terms of mega-, giga-, or tera-Flops. The key concern in this form of analysis is to represent the expected system capacity as a percentage of theoretical capacity of the system. This ratio quantifies the unused (or idle) portion of the capacity and how much the system spends for overhead purposes such as communication.

4.3.2. Forms of analysis

As we have already mentioned EAPS will allow the simultaneous performance of a number of heterogeneous analyses. Major analysis forms that eventually will be supported are:

- **Analytical** analysis is based entirely on previous measurements of a system, or on known features of the design of a system. In the simplest case one might measure the vector performance of a single processor and model it as a function of vector length. Indeed this particular example is particularly well known, being described typically by the vector length required to achieve 50% of peak performance. Such a model can then be incorporated in analyzing performance of a code that is dominated by vector operations - provided the lengths of vectors are known, and not determined only at runtime. Even in the case that needed information is available only at runtime, an analytical model can still be incorporated into a performance estimator provided the appropriate runtime parameters are available to it. Analytical analysis is an important tool in providing understanding of the gross behavior of codes, but it is exceedingly hard to extend to any level of detail. However providing even rough "ball-park" estimates of performance can be important - both to system designers and to applications users.
- **Simulation** analysis is based on a simulation of an actual application. Typically the code is represented schematically in some way and then executed on a simulator. Traces of various operations and parameters are accumulated during the simulation. The key is to use an effective representation that captures the main aspects of the code while omitting fine detail that would cause the application simulation to run endlessly. For example an operation such as a Fast Fourier Transform may be represented as a single item in a trace, and its "cost" assigned using an analytical or measured performance from an earlier benchmarking analysis. Sometimes one may run a simulation analysis because the hardware target is either not available, or has not been built yet. Here again, simulating all aspects of execution tends to be impractical in most

cases, and models for behavior of components of a system can greatly speed the simulation without sacrificing too much predictive accuracy.

- **Benchmarking** analysis involves tracing and measurement of an application on real hardware. Depending on the tracing environment used, information obtained may be as simple as execution time or as detailed as a complete trace of all memory accesses in all processors. For applications of significant size, complete traces might run to billions of operations and are not practical. However traces of subroutines or smaller blocks may be extremely helpful in building an analytical or modeled description of such blocks. Benchmarking is also invaluable for locating critical sections and suggesting places where optimization would be most fruitful. The nature of the benchmarking performed will typically depend on the architecture. In the case of shared memory systems the emphasis tends to be on tracing access patterns to shared memory locations, while for distributed memory systems, tracing of interprocessor communication patterns is most relevant.

- **User-defined code.** The performance of an entity can be determined by executing custom developed code which may be a new algorithm, an alternate description of behavior of an entity, or a new simulation of a device.

- **Data from actual code.** An extreme case of flexibility is the incorporation of performance data from the execution of actual code on a target architecture.

4.4. System optimization and dynamic reallocation of tasks

A major consideration in the design of parallel systems is the optimization of performance by reassigning tasks to system nodes. Although at first it may seem that equalizing node utilization may lead to an optimal system, in reality this is far from true for two reasons: (1) equalizing node utilization may lead to higher communication overhead and lower efficiency, and (2) since tasks are not infinitely divisible, achieving uniform node utilization is an impossibility. Therefore, the problem can be viewed as a non-linear constrained optimization problem.

Independent of how the problem is formulated or solved, the optimization is useful only if it can be done dynamically, i.e. during processing tasks are reassigned based on a pre-defined criterion. Dynamic reassignment of tasks adds another dimension to an already difficult problem.

4.5. Libraries

Libraries are the critical building blocks that allow users to reuse previously developed simulation specifications, and to create new ones in a methodical fashion. A library stores a node description, network topology, program or other specification in a form where it may later be accessed by referring to its name and providing appropriate parameters. Libraries also encode default values for attributes and allow applications to access library modules transparently. Libraries are the primary means of providing performance information which has been previously obtained either by analytic or experimental simulation.

Libraries are allowed to be hierarchical. While all elements of a library are regarded as equivalent, a library may utilize a lower-level library to which it places calls. No cycling is allowed - the lower-level library may not call the higher level one for example. Hierarchical libraries may be used to define multilevel descriptions of entities. In this case, the top level library modules record the top level view of a node, network or program, and, under control of a parameter, may either return this description, or may make a call to the next level library to give a more detailed representation.

4.6. Application Program Interface (API)

Application Program Interface (API) expands the capability of EAPS by allowing the user to develop, edit, review, and execute other programs from within EAPS. Upon receipt of the *Execute* command, EAPS is temporarily exited to execute the selected program, and return. Major applications of API are to interface with other tools, to develop and execute custom developed code, and to develop benchmarks and include them in the other forms of analysis.

4.7. Tracing and profiling capability

Tracing and profiling, an intermediate step between system definition and analysis, enables EAPS to accept performance, attribute, and other data about a system entity from an external source, e.g. other tools, or via the execution of actual code. Its major use is deriving (or specifying) the attributes of complex entities that will be difficult to find otherwise. A typical example is defining the attributes of a decomposed node containing computing and communication processors, local memory, cache, and registers. Predicting the performance of such a complex entity, analytically (or via simulation), can be difficult. An alternate way will be to incorporate data derived either from executing an actual code, a benchmark, or from another tool into EAPS. Tracing will be an option in the definition form of items being specified and permits the user to name a file (or a tool) to find the attributes of the desired entity.

5. Bibliography

[1] Bailey, D., et. al., "*The NAS Parallel Benchmarks*", Tech. Rep. RNR-91-002, NAS Systems Division, 1991.

[2] Dongarra, J.J.; Sorenson, D.C., "*Linear Algebra on High Performance Computers*", Invited paper at "Parallel Computer 85", pp 3-32.

[3] Dongarra, J., Marti, J. L., Worlton, J., "*Computer Benchmarking: paths and pitfalls*", IEEE Spectrum, Vol 24, No. 7, July 1987, pp. 38-43.

[4] Grassel, C., Scharwzmeier, J.; "*Performance of Application Programs on Supercomputers: Results of Perfect Benchmarks,*" CRAY Research, Inc., 1990.

[5] McBryan, O., "*Hypercube Algorithms and Implementations*", SISSC, 8, pp. 227-287, 1987.

[6] McBryan, O., "*Performance Evaluation of the Myrias SPS-2 Computer*", to be published.

[7] McBryan, O., "*Optimization of Connection Machine Performance*", International Journal of High Speed Computing, vol. 2, no. 1, pp. 23-48, 1990.

[8] McBryan, O., "*New Architectures: Performance Highlights and New Algorithms*", Parallel Computing, vol. 7, pp. 477-499, North-Holland, 1988.

[9] McBryan, O. and E. Van de Velde, *"Hypercube Algorithms and Implementations"*, SIAM J. Sci. Stat. Comput., 8, pp. 227-287, 1987.

[10] McBryan, O. and Pozo R., *"Performance Evaluation of the Evans and Sutherland ES-1 Computer,"* CS Dept Technical Report CU-CS-506-90, Univ. of Colorado, Boulder, 1990.

[11] Pazirandeh, M., "*An Environment for Simulation of Distributed Systems*", Phase II SBIR Contract, AIRMICS, 1989 and 1991, Contract number DAKF11-91-C-0008.

[12] Pazirandeh, M., "*An Environment for Analysis of Parallel Systems*", Phase I SBIR Contract, NSWC, 193, Contract number N60921-92-C-0106.

[13] Saavedra-Barrera, R., et. al. "*Machine Characterization Based on an Abstract High-Level Language Machine,*" IEEE Trans. on Computers, Vol. 38, No. 12, Dec 1989, pp 1659-1679.

A Dependable System Perspective*

M. M. Hugue, N. Suri, C. J. Walter
Allied-Signal Aerospace Company
Columbia, MD 21045
mmh@batc.allied.com

Abstract

Dependable systems are needed to meet the demands of critical real-time applications. The reliability, response time and recovery time requirements are used to divide the set of dependable systems into three classes: ultra-dependable systems, highly dependable systems, and highly available systems. Other system characteristics implicit in the class can then be extracted. This scheme is applied to a variety existing dependable systems.

1 Introduction

Critical real-time applications depend upon embedded digital systems to perform speedy and precise computations. In the past, fault-tolerance methods that assured error-free results were developed separately from methods that guaranteed real-time performance. The assumptions made to address real-time issues were often in conflict with those needed for fault-tolerance and vice versa. As a result, few systems exist that can be guaranteed to meet critical real-time constraints in the presence of faults. Both current and future applications, such as aircraft flight control, high-speed communication, and on-line data retrieval require dependable systems. Such systems must address real-time and fault-tolerance constraints to meet their error-free service requirements. The generic concept of *dependability* encompasses the issues and techniques most commonly used to identify, implement, and measure system fault-tolerance and real-time performance. As defined in [1], dependability is a qualitative term that describes the trustworthiness of a computer system. Our goal is to use characteristics relevant to both real-time and fault-tolerant concerns in identifying dependable system features appropriate for a given application.

Before proceeding, we review terminology essential to this discussion. Then, we discuss the use of an application's reliability, response time, and recovery time requirements in determining essential features of candidate dependable systems. We conclude by applying this method to existing fault-tolerant and real-time systems.

2 Fundamental Terminology

The design of a dependable real-time system must integrate issues common to both fault tolerant and real time systems. As shown in Figure 1, from [1], dependability can be partitioned into

*Supported in part by ONR Contract # N00014-91-C-0014

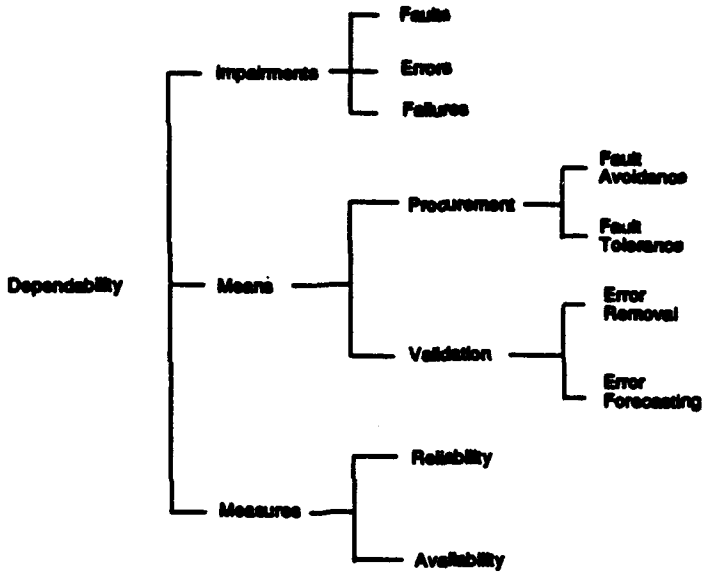


Figure 1: Dependability Issues

impairments, means and measures. Dependability *impairments* consist of failures, errors, and faults which can prevent service requirements from being met. Dependability *means* include *fault avoidance*, where faults are prevented through the use of intrinsically reliable components or formal methods; *fault tolerance*, where the effects of errors are masked through the use of redundant elements; *error removal*, where the presence of latent errors is minimized; and *error forecasting*, where the presence, creation, and consequences of errors are estimated.[1]

Dependability *measures* include availability and reliability. The *availability* of a system, $A(t)$, is the probability that the system is operational at the instant of time t. If the limit of $A(t)$ as t approaches infinity exists, that limit represents the fraction of time that the system is capable of performing useful work. The *reliability* of a system, $R(t)$, is the conditional probability that a system will be operational at time $t = \tau$, given that it was operational at time $t = 0$. Thus, it is typically more difficult to guarantee reliability than availability. [2] The numerous tradeoffs between reliability and availability significantly affect system life-cycle costs. The reliability requirement of correct operation throughout an interval is more stringent (and expensive) than the instantaneous availability requirement. In fact, some applications may be able to tolerate minutes or even hours of unavailability while a failed system recovers. Thus, the *recovery time* requirements also identify dependable system parameters suitable for a given application.

A dependable system must perform functions or provide services within a time frame determined by the application requirements. In a general sense, fault-tolerance focuses on improving system reliability and availability, by supporting continual correct operation after faults occur or restoring operations after system failure. Similarly, real-time research focuses on ensuring the timeliness of services. Real-time systems have three basic task components: reading inputs, performing computations, and producing outputs. Services must be delivered within finite time intervals as dictated by the application *response time* and other performance requirements. There may be

[1]The reader is referred to [1] for a detailed discussion of dependability issues.

several modes of operation (such as: takeoff, cruise, and land) which can cause the workload to increase or decrease. In a *hard* real-time system, computations must be performed with a specified frequency or response time. The failure of a task to start or to complete on time can have intolerable consequences, such as loss of system control or loss of life. Conversely, in a *soft* real-time system missed deadlines may be tolerable. A task may start or complete late without causing system failure, provided that the workload behavior satisfies an acceptable statistical distribution. Systems with hard deadlines often rely on worst case scheduling analyses to guarantee task deadlines. Systems permitting soft deadlines may schedule tasks based on average behavior to improve performance and response time. An in-depth study of real–time computing can be found in [3].

3 Dependable System Classes

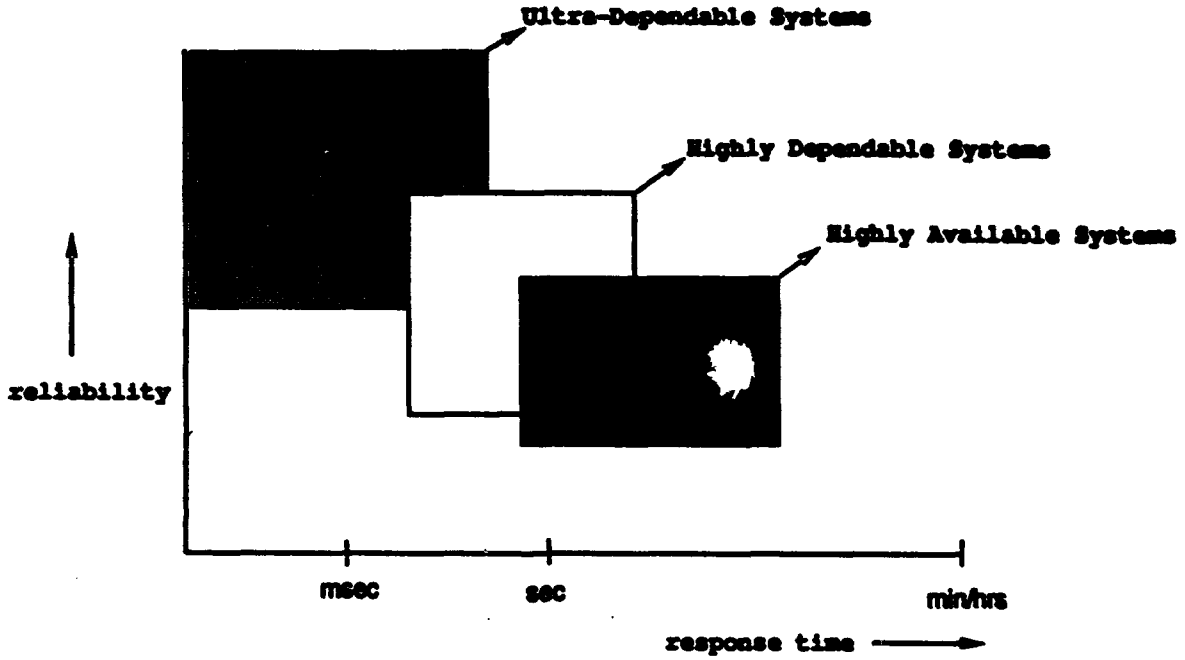
In previous work, fault-tolerant real-time systems were distinguished by their reliability and deadline requirements. This often leads to confusion during the design stage because the target system reliability range and deadlines (hard or soft) are not sufficient to further determine fault-tolerant or real-time system requirements. In this section, we use the additional parameters of response time and recovery time to identify general classes of suitable systems. Other system features are then implicit in the class.

Large markets with specific demands have spawned systems targeted for several types of dependable computing applications, such as telephone service, on-line transaction processing, and real-time control. Each of these systems is required to provide dependable service, but with very different reliabilities and response times. The discussion below is based on three different graphs. The upper graph of Figure 2 shows the application space with respect to the attributes of reliability and response time. The lower graph of Figure 2 shows suitable recovery techniques with respect to reliability and recovery time. These graphs are combined in Figure 3 to yield a composite dependable system overview.

In Figure 2, we have highlighted three separate system classes: Highly Available Systems (HAS), Ultra-Dependable Systems (UDS), and Highly Dependable Systems (HDS). Each region accommodates applications which require the given levels of reliability and response time. High availability systems address the needs of on-line transaction processing applications. Banking, sales, inventory control, and telephone systems must be available continuously, and the integrity of shared databases must be maintained. These systems are distinguished by their ability to tolerate short down-times in the range of minutes to hours. Repair or recovery from faults can often be postponed until predefined maintenance intervals, without loss of life or property. Mission times for HAS are on the order of 100 hours, significantly longer those of other systems.

Highly dependable systems emphasize reliability over availability because particular applications need to be completed without interruption. The class of HDS accommodates applications that accept mission failures on the order of one in 10^5 to 10^7 missions, with mission times between 10 and 100 hours. Since a physical process may be controlled, typical response and recovery times can range from milliseconds to seconds. System applications which are essential, but may have backups or permit human intervention, are typically implemented using HDS. Many submarine and battleship fire control applications can be accommodated by HDS, as can communication systems, security systems, and environmental control systems.

Ultra-dependable systems represent applications where loss of the computer system is unacceptable, as it may cause loss of life or destroy extremely expensive property. Very quick response



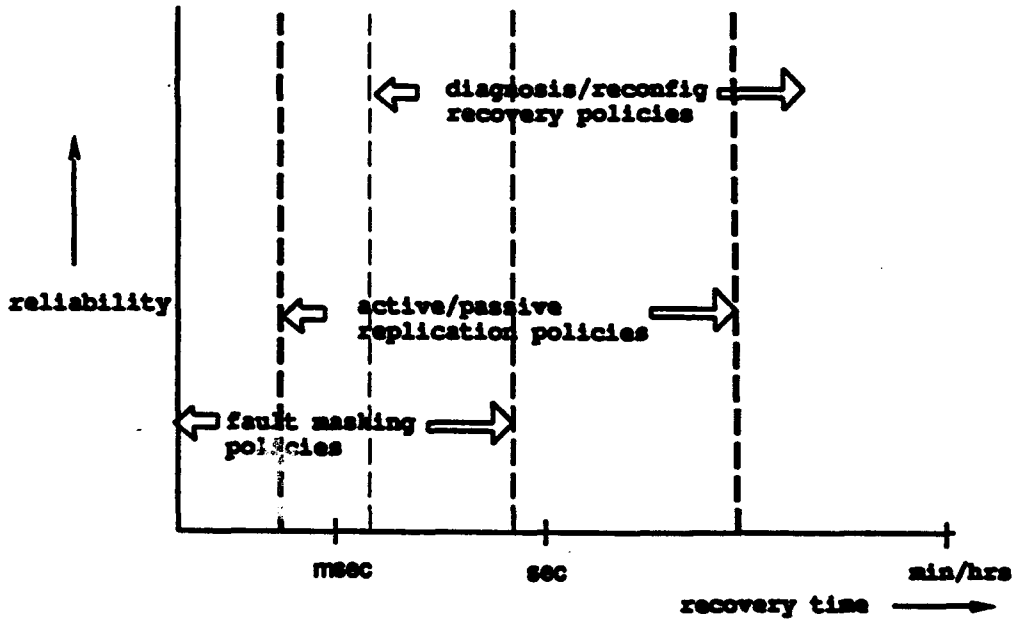


Figure 2: Domain Space for Systems and Techniques

times are needed because even the most basic application functions depend on the computer. For example, a momentary loss of computing power in an aircraft flight controller can cause the plane to become unstable and leave the safe-flight envelope. The reliability requirements are extremely high, with failure probabilities in the range of 10^{-7} to 10^{-9} for mission times of ten hours or less. The acceptable recovery times are so small that expensive resource replication and forward error recovery[2] methods must be used to mask faults before they can cause system failure. Historically, fault-tolerance concerns have dominated real-time performance concerns. Applications designers were often required to hand-craft new task schedules, even when only minor changes occurred in the workload, increasing the potential for errors in the system.

The lower graph of Figure 2 shows the tradeoffs between reliability and recovery time requirements. Applications that emphasise high availability over long mission times can afford slower fault recovery times than applications with shorter mission times. Applications with short mission times must typically react to the physical environment. Continuous availability is expected because *any* interruption in service can be catastrophic if it is not immediately recoverable. When very fast recovery times are needed, fault masking must be used as there is no time to interrupt processing for recovery. Active redundancy management policies become feasible when recovery times are in the range of milliseconds to seconds. Finally, more time consuming policies, such as diagnosis and roll-back or reconfiguration are allowable with longer recovery times.

The above observations are combined in Figure 3 to provide a composite view of dependable system characteristics. We have noted several representative systems for each class. The number of prospective applications continues to increase due to the fast dynamic response of computers and the inherent reliability of integrated digital electronics. The applications can be further examined with respect to the user's emphasis on different characteristics of dependability, specifically: availability, safety, and fault tolerance. Distributed approaches to these problems have been selected since they possess many appealing characteristics, such as avoidance of a single point of failure and the ability to physically distance resources. These classes address the different trends envisioned for the next generation of computing resources and applications. They also identify many intriguing practical problems and research topics in designing and evaluating dependable computing systems for time critical applications.

4 Discussion

In this paper, we have presented three classes of dependable systems suitable for different sets of application requirements. We have added response time and recovery time requirements to the system reliability and deadline requirements commonly used to characterize dependable systems. We divided the set of dependable systems into three classes: highly available systems, highly dependable systems, and ultra-dependable systems. Then, we mapped many existing system approaches into this framework. While the granularity of this division may not be sufficient for all critical real-time applications, it provides a system designer with a guide to appropriate fault-tolerance and performance enhancement techniques.

The issues of concern in developing dependable systems cover all phases of the system life-cycle. It is important to understand that the payoff associated with fault tolerance may be apparent only

[2]Forward error recovery refers to the use of masking techniques to maintain correct system operations in the presence of faults.

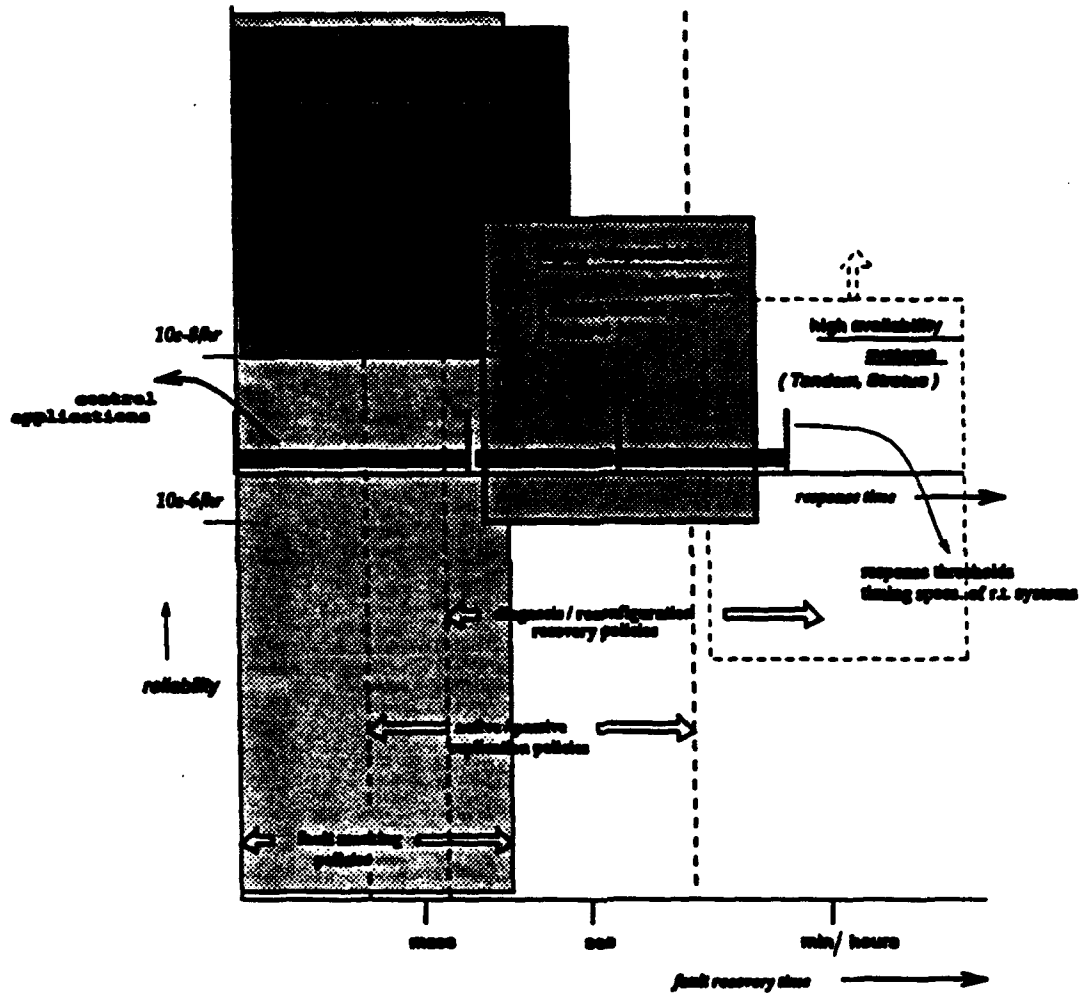


Figure 3: The Composite Dependable System and Real-Time Viewpoint

when the complete life-cycle of the system is examined. The strategies formulated during the initial design stage have a significant impact on the resulting system dependability and cost. Thus, system design and analysis based on this framework can ensure that parameters critical to the application are identified early in the system design stage. While the development of theoretical and practical approaches to dependability continues, the design of dependable computers still remains an art.

References

[1] J. Laprie, *Dependability: Basic Concepts and Terminology.* Springer-Verlag, 1992.

[2] D. Siewiorek and R. Swarz, *The Theory and Practice of Reliable System Design.* Bedford, Massachusetts: Digital Press, 1st ed., 1982.

[3] K. G. Shin and C. Krishna, "Characterization of real-time computers," NASA Contractor Report 3807, NASA, 1984.

IMPLEMENTATION TECHNOLOGY

Real-Time Databases for Complex Embedded Systems: Predictability and Serializability

Kwei-Jay Lin

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Sang H. Son

Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903

April 15, 1993

ABSTRACT

Real-time databases (RTDBS) for complex systems have transactions with explicit timing constraints, such as deadlines. Conventional database systems are not designed for time-critical applications and lack features required for supporting real-time transactions. Meeting the requirements of RTDBS will require a balanced and coordinated effort between concurrency control and transaction scheduling. In this paper, we focus on two issues: predictability and serializability. One approach is to combine existing concurrency control protocols with real-time scheduling algorithms. To meet more deadlines, concurrency control protocols can be modified to favor more urgent transactions. Another approach is to explore the non-serializable semantics in real-time applications. We survey and discuss many techniques that can be used to design and implement real-time databases.

1. Introduction

As our society becomes more integrated with computer technology, information processing for human activities necessitates distributed and fault-tolerant computing that responds to requests in *real-time*. Many computer systems are now used to monitor and control physical devices and large complex systems which must support real-time capability. Since the real world is constantly evolving, it is very important for system designers to design and implement real-time systems so that they can always keep up with the real world.

Real-time database systems (RTDBS) have (at least some) transactions with explicit timing constraints, such as deadlines. RTDBS are becoming increasingly important in a wide range of applications such as aerospace and weapon systems, computer integrated manufacturing, robotics, nuclear power plants, network management, and traffic control systems. Unfortunately, conventional database systems are not designed for time-critical applications and lack features required for supporting real-time transactions. They are designed to provide good average performance, while possibly yielding unacceptable worst-case response times. It has been generally recognized that there is a lack of basic theory for real-time database systems since the traditional models are not adequate for time-critical applications [Abb88, Buc89, Kor90, Lin89, LS90, Raj89, SRL88, Sha91, Son88, Son88b, Son90, Son91, Son92, Son93].

Typical real-time systems try to meet the timing constraints of individual *tasks*, but ignore data consistency problems. Task and *transaction* abstractions are similar in the sense that both are units of work as well as units of scheduling. However, tasks and transactions are different computational concepts and their differences affect how they are controlled. In real-time task scheduling, it is usually assumed that all tasks are preemptable. Preemption of a transaction that uses a file resource in an exclusive mode of writing may result in subsequent transactions reading inconsistent information. In addition, while the runtime behavior of a task is statistically predictable, the behavior of a transaction is dynamic, making it difficult to predict its execution time with accuracy.

Most real-time database operations are characterized by (1) their time constrained access to data, and (2) access to data that has temporal validity. These operations involve gathering data from the environment, processing the gathered information in the context of information acquired in the past, and providing timely responses. The operations also involve processing not only archival data but also temporal data which loses its validity after a certain time interval. Both of the temporal nature of the data and the response time requirements imposed by the environment are transaction timing constraints handled by either periods or deadlines. Therefore, the correctness of real-time database operation depends not only on the logical computations carried out but also on the time at which the results are delivered. The goal of real-time database systems is to meet the timing constraints of transactions.

A key point to note here is that real-time computing is not equivalent to fast computing. Rather than being fast, more important properties of RTDBS should be timeliness, i.e., the ability to produce expected results early or at the right time, and predictability, i.e., the ability to function as deterministic as necessary to satisfy system specifications including timing constraints [Stan88]. Fast computing which is busy

doing the wrong activity at the wrong time is not helpful for real-time computing. While the objective of real-time computing is to meet the individual timing constraint of each activity, the objective of fast computing is to minimize the average response time of a given set of activities. Fast computing is helpful in meeting stringent timing constraints, but fast computing alone does not guarantee timeliness and predictability. In order to guarantee timeliness and predictability, we need to handle explicit timing constraints, and to use time-cognizant techniques to meet deadlines or periodicity associated with activities.

RTDBS have some very unique requirements. The design and implementation of RTDBS introduces many new and interesting problems: What is an appropriate model for real-time transactions and data? What are the language constructs that can be used to specify real-time constraints? What are the measures of system predictability? How should real-time transactions be scheduled? What is the best concurrency control scheme that considers real-time constraints and importance of transactions? Is serializability an appropriate correctness criterion for RTDBS? In this paper we try to answer some of the questions and review current approaches to the design of RTDBS. We focus on two important issues: predictability and serializability.

The remainder of this paper is organized as follows. In Section 2, we first describe some of the characteristics and requirements of RTDBS, and discuss about issues involved with schedulability, predictability, and non-serializable executions. In Section 3, we discuss priority-based conflict resolution mechanisms and review some of the algorithms that are based on serializability. Section 4 presents techniques for generating a set of schedules that are non-serializable but acceptable to RTDBS. Finally, concluding remarks with future research issues are summarized in Section 5.

2. RTDB Characteristics and Requirements

The reasons why conventional database systems are not used in real-time applications include their poor performance and their lack of predictability. In conventional database systems, transaction processing requires access to a database stored on secondary storage; thus transaction response time is affected by disk access delays, which can be in the order of milliseconds. Although these databases are fast enough for traditional applications in which a response time of a few seconds is often acceptable, these systems may not be able to provide a response fast enough for high-performance real-time applications. One common approach to achieve high performance is to replace slow devices (e.g., a disk) by a high speed devices (e.g., a large main memory). Another alternative is to use special techniques to increase the degree of concurrency [Son88b].

Since real-time systems are often used in safety-critical applications, they must provide predictable performance. An unpredictable system can do more harm than good under abnormal conditions. There are many reasons why traditional database systems may have unpredictable performance. For example, to ensure the data consistency, traditional database systems often block certain transactions from reading or updating data if these data are locked by other transactions. Blocking will cause transactions to be delayed. Even worse, it is often difficult for a transaction to predict how long the delay will be since the blocking transactions themselves in turn may be blocked by other transactions. Consequently, the

response time for a transaction in conventional database systems is often unpredictable.

Moreover, databases in many real-time systems have the following unique problems:

(1) - Many data objects in a database correspond to active data objects in the real world. Their values may change by themselves, regardless of the database state and activities.

(2) A real-time database may never be completely *correct*. As soon as a real-world value is recorded in the database, it may already be out of date.

(3) Different data objects in a database may be recorded at different rates. Their values may not co-exist in the same real-world snapshot.

Due to these problems, RTDBS need special protocols to ensure that all transactions executed are *necessary* and *useful*. A transaction execution is necessary only if it can help other more critical transactions produce correct results (both in terms of time and value). A transaction execution is useful only if its result can be applied to the system without producing any adverse effect to the system mission. Toward these goals, an execution using some traditional protocol may not be acceptable to RTDBS, while RTDBS may accept some transaction executions using unconventional measures.

In the following, we compare real-time database systems with other types of databases such as CAD/CAM databases and business databases (Table 1). From these comparisons, we may have a better picture of the true requirements of real-time database systems.

Table 1. Database Characteristics

Applications	Data Characteristics	Transaction Characteristics	Performance Requirements
Real-Time Systems	Device inputs System and machine state System history/statistics Multi-media Temporal attributes Quality attributes	Periodic, short Event-driven Producer/consumer Frequent updates Decision making Associative (or no) search	Schedulability Hard deadline Graceful degradation System reliability Data availability Best effort
CAD/CAM	Design data Complex 3-D structure Hierarchical Persistent objects	Iterative, tentative Long transactions Sharing among groups Frequent updates	Version control Special concurrency Friendly user interfaces Flexible manipulation
Financial Systems	Large volume Simple objects Flat structure Numeric or text	Large table search Complex query Storage intensive Atomic actions	Data correctness Strict serializability System throughput High reliability

2.1. Data Characteristics

Since real-time systems are used to monitor and to control physical devices, they need to store a large amount of information about their environments. Such information includes input data from devices as well as system and machine states. In addition, many embedded systems must also store the system execution history for maintenance or error recovery purposes. Some systems may also keep track of some system statistics like average system load or average device temperature. Depending on the applications, real-time systems may have to handle multi-media information like audio (for sonar devices), graphics (for radar devices), and images (for robots). Since systems are constantly recording information, data must have their temporal attributes recorded. Also, some input devices may be subject to noise degradation and need to record the quality of the attributes along with the data.

Often a significant portion of a real-time database is highly perishable in the sense that it may contribute to a mission only if used in time. In addition to deadlines, therefore, other kinds of timing constraints could be associated with data in RTDBS. For example, each sensor input could be indexed by the time at which it was taken. Once entered into the database, data may become out-of-date if it is not updated within a certain period of time. To quantify this notion of "age", data may be associated with a *valid interval* [Liu88, SL92]. Data outside its valid interval does not represent the current state. What occurs when a transaction attempts to access data outside its valid interval depends on the semantics of data and the particular system requirements.

In comparison, data in CAD/CAM applications are mostly related to design information. They often have complex and hierarchical structures. Data in financial systems, on the other hand, are much simpler and flat in structure. However, financial systems usually have a large volume of data to be processed.

2.2. Transaction Characteristics

Transactions in real-time database systems can be categorized as *hard* and *soft* transactions. We define hard real-time transactions as those transactions whose timing constraints must be guaranteed. Missing the deadlines of this type of transactions may result in catastrophic consequences. In contrast, soft real-time transactions have timing constraints, but there may still be some justification in completing the transactions after its deadline. Catastrophic consequences do not result if soft real-time transactions miss their deadlines. Soft real-time transactions are scheduled taking their timing requirements into account, but they are not guaranteed to make their deadlines. There are many real-time systems that need database support for both types of transactions.

Many transactions in real-time database systems are used to record device readings or to handle system events. They are either periodic or event driven. Most of the transactions are short since they must be responsive to their environment. Transactions can be viewed either as producers or consumers for certain input data. For periodic transactions, most will perform updates in every period. The transactions triggered by events must often make decisions. Due to the real-time constraints, either no search or associative search mechanism is preferred.

In comparison, CAD/CAM transactions are more iterative in structure and longer in durations. They need to share information among group users. Also, many frequent updates will be conducted before a design is finalized. For financial applications, the queries requested could be quite complex and require long searches through large tables. Financial transactions therefore require fast high-volume storage devices to speed up their operations. To prevent data inconsistency, financial transactions usually are executed as atomic actions.

2.3. Performance Requirements

For CAD/CAM applications, systems must control the versions evolved during the design process. Due to the cooperation among team users, special mechanisms must be provided to facilitate concurrent access from different users. A friendly user interface and flexible manipulation primitives are definitely desirable. For financial applications, the most important performance criterion is the control of concurrent transactions in order to ensure that the effect of concurrent executions is equivalent to the effect those transactions would have had, if they had been run in some serial order. This well-known *serializability* goal in transaction processing is well established as an appropriate notion of correctness for conventional transaction scheduling. In contrast to real-time systems, conventional database systems do not emphasize the notion of timing constraints or deadlines for transactions. The performance goal is to reduce the response times of transactions by using a serialization order among conflicting transactions. For example, the most commonly used two-phase locking (2PL) protocol [Bern87] synchronizes concurrent data access of transactions by blocking and thus may violate the timing constraints of transactions.

The most important requirement for real-time applications is to provide a feasible schedule so that transactions can meet their hard deadlines. Systems also must degrade gracefully since their applications are often safety-critical. System reliability is definitely an important issue for these systems. To make fast and correct decisions, data availability is critical to system performance. Also, due to timing constraints, many decision-making processes can only use the best-effort approach; they must stop at the deadlines. In RTDBS, the timeliness of a transaction is usually combined with its criticality to calculate the priority of the transaction. Therefore, proper management of priorities and conflict resolution in real-time transaction scheduling is essential for insuring the predictability and responsiveness of RTDBS.

2.4. Issues on Predictability and Serializability

One of the most interesting challenges in RTDBS is the creation of a unified theory for real-time scheduling and concurrency control that maximizes both concurrency and resource utilization subject to three constraints: *data consistency*, *transaction correctness*, and *transaction deadlines* [Stan88]. While the theories of concurrency control in database systems and real-time task scheduling have both advanced, difficult problems remain in the interaction between concurrency control protocols and real-time scheduling algorithms. In database concurrency control, the objective is to provide a high degree of concurrency and thus faster average response time without violating data consistency. In real-time

scheduling, it is desirable to maximize resources usage, such as CPU utilization, subject to meeting timing constraints. If the system is not designed properly, it may be impossible to meet both objectives: a transaction may need to be blocked if it is in conflict with other transaction, yet it must be executed immediately to meet its deadline.

The first issue we want to address is the *predictability* of real-time database systems. As stated above, the goal of scheduling in RTDBS is to meet timing constraints. Many real-time task scheduling methods can be extended to real-time transaction scheduling, while concurrency control protocols are still used to maintain data consistency. The general approach is to utilize existing concurrency control protocols, especially two-phase locking, and to apply time-critical transaction scheduling methods that *favor* more urgent transactions [Son89]. Such approaches have the inherent disadvantage of being limited by the concurrency control protocol upon which they depend, since all existing concurrency control protocols synchronize concurrent data access of transactions by a combination of two measures: blocking and roll-backs of transactions. Both are barriers to meeting time-critical schedules. Several recent projects have tried to integrate real-time constraints with database technology to facilitate efficient and correct management of timing constraints in RTDBS. In Section 3, we will discuss some of the projects and compare the results.

The second issue that we will discuss is the *serializability* of RTDBS. Traditional concurrency control protocols induce a serialization order among conflicting transactions. Although in some applications weaker consistency is acceptable [Gar83], a general-purpose consistency criterion that is less stringent than serializability has been difficult to find. However, RTDBS may have a different notion of "correct execution" in transaction processing. Based on the argument that timing constraints may be more important than data consistency in RTDBS, attempts have been made to satisfy timing constraints by sacrificing database consistency temporarily to some degree [Lin89, Vrb88]. It is based on a new consistency model of real-time databases, in which maintaining *external data consistency* (values of data objects represent correct values of external world outside the database) has priority over maintaining *internal data consistency* (no data that violates consistency constraints). Moreover, a real-time transaction may include a *temporal consistency requirement* that specifies the validity of data values accessed by the transaction. While a deadline can be thought of as providing a time interval as a constraint in the future, temporal consistency specifies a temporal window as a constraint in the past. As long as the temporal consistency requirement of a transaction can be satisfied, the system may want to provide an answer using available (may not be serializable) information. In Section 4, we investigate several of the protocols which provide non-traditional (i.e. non-serializable) transaction schedules that are acceptable in RTDBS.

3. Serializable Execution of Real-Time Transactions

Conventional transaction processing requires controlling the concurrent execution of transactions to ensure serializability. However, the notion of serializability may be too restrictive for real-time transactions. In real-time database systems, guaranteeing temporal consistency requirements might be more critical than satisfying the conventional notion of serializability.

In this section, we review some of the proposed scheduling and concurrency control algorithms, based on serializability, for real-time transactions. To enforce serializability, they use conventional scheduling and concurrency control schemes such as two-phase locking (2PL), timestamp ordering (TO), and optimistic concurrency control (OCC) as their basis. They combine those conventional schemes with priority-based conflict resolution mechanism such as *priority abort* [Abb88, Abb92, Hua89, Hua91], *priority inheritance* [Sha91, Hua91], *priority wait* [Hua91, Har90, Har91], and *adjusting serialization order* [LS90, Son92]. Schemes that are based on the correctness criterion different from serializability will be discussed in Section 4.

3.1. Locking-based Conflict Resolution

Concurrency control protocols induce a serialization order among conflicting transactions. For a concurrency control protocol to accommodate timing constraints of transactions, the serialization order it produces should reflect the priority of transactions. However, this is often hindered by the past execution history of transactions. A higher priority transaction may have no way to precede a lower priority transaction in the serialization order due to previous conflicts. For example, let T_H and T_L be two transactions with T_H having a higher priority. If T_L writes a data object x before T_H reads it, then the serialization order between T_H and T_L is determined as $T_L \rightarrow T_H$. T_H can never precede T_L in the serialization order as long as both reside in the execution history. Most of the current (real-time) concurrency control protocols resolve this conflict either by blocking T_H until T_L releases the write lock or by aborting T_L in favor of the higher priority transaction T_H. Blocking of a higher priority transaction due to a lower priority transaction is contrary to the requirement of real-time scheduling. Aborting is also not desirable because it degrades the system performance and may lead to violations of timing constraints. Furthermore, some aborts can be wasteful when the transaction which caused the abort is aborted due to another conflict.

Abbott and Garcia-Molina have proposed a restart-based 2PL [Abb88, Abb92]. It incorporates priority information in lock setting so that transactions with higher priority will be given a preference. Whenever a higher priority transaction is in conflict with a lower priority transaction, the lower priority transaction will be aborted and restarted later on. One of the weaknesses of this scheme is the impact of restarts on scheduling other transactions to meet their timing constraints. Restarting a transaction could be very costly in terms of wasted resources, and a large number of restarts increase the workload of the system and may cause other transactions to miss their deadlines.

To reduce the number of restarts, the conditional restart protocol is proposed by the same authors. In this protocol, the lower priority transaction will have to be restarted only if the slack time of the higher priority transaction is smaller than the remaining execution time of the lower priority transaction that holds the lock. There are a few problems with this protocol. First, the effectiveness of this checking is greatly affected by the blocking probability of the lower priority transaction. Second, the scheduler must have information such as the execution time and slack time. In real-time database systems, such information is hard to get due to the dynamic nature of resource demands and the data-dependent execution path

of transactions. Furthermore, priority inversion and deadlock is still possible, although they have a lesser degree of impact.

Huang et al. developed and evaluated a group of protocols for real-time transactions [Hua89, Hua91]. Their study includes protocols for CPU scheduling, data conflict resolution, deadlock resolution, transaction wakeup, disk scheduling, and transaction restart. In terms of conflict resolution, they compared three approaches:

(1) priority inheritance which eliminates the problem of CPU blocking (not data blocking) and attempts to reduce the period of priority inversion by allowing the low priority transaction holding the lock to execute at the priority of the highest priority transaction waiting for the lock,

(2) priority abort w ch completely eliminates the problem of priority inversion as well as CPU blocking by aborting the low priority transaction,

(3) conditional priority inheritance which is a compromise between the two, taking the remaining execution time of the low priority transaction into consideration.

Their performance study is not based on simulation but on actual implementation of those protocols on a real-time database testbed called RT-CARAT. Their results indicate that CPU scheduling protocols have a significant impact on the performance of real-time databases. They also found that the overhead incurred in locking is non-negligible and hence must not be ignored in the analysis of real-time transaction processing. For conflict resolution, their performance results indicate that with respect to deadline guarantee ratio, the priority inheritance scheme does not work well, while conditional priority inheritance and priority abort schemes perform rather well for a wide range of system workloads. They clarified through experiments that the priority inheritance scheme is quite sensitive to the priority inheritance period. A long priority inheritance period will affect not only the blocked higher priority transactions but also other concurrent non-blocked higher priority transactions. The conditional priority inheritance scheme works well because of its reduced priority inheritance period.

3.2. Optimistic Approaches

Recently, real-time concurrency control protocols based on the optimistic approach have been proposed and studied [Har90, Hua91b, Son91]. Optimistic concurrency control (OCC) exploits the low probability of data conflicts [Kun81]. It is a non-blocking protocol: the OCC scheduler uses abort and restart to serialize concurrent data operations, thereby avoiding blocking. Subsequently, OCC is free from deadlock. In addition, it has a potential for a high degree of parallelism. These features of OCC make it promising particularly for real-time transaction processing. However, the abort-based conflict resolution of OCC has the problem of wasted resources and time.

In OCC, write requests issued by transactions are not immediately processed on data objects but are deferred until the transaction submits a commit request, at which time the transaction must go through the *validation phase*. Because write operations effectively occur at commit time, the serialization order selected by an OCC protocol is the order in which the transactions actually commit through the validation

phase. Transaction validation can be performed in one of two ways: *forward validation* or *backward validation*.

In OCC protocols that perform backward validation, the validating transaction either commits or aborts depending on whether it has conflicts with transactions that have already committed. Thus, this validation scheme does not allow us to take transaction characteristics into account. In forward validation [Hae84], however, either the validating transaction or conflicting ongoing transactions can be aborted to resolve conflicts. This validation scheme is advantageous in real-time database systems, because it may be preferable not to commit the validating transaction, depending on the timing characteristics of the validating transaction and the conflicting ongoing transactions. A number of real-time concurrency control methods based on OCC using forward validation scheme have been studied [Har90, Hua91b, Son92].

Haritsa et al. proposed a group of optimistic real-time concurrency control protocols, based on forward validation, and evaluated them on a simulation model [Har90]. When a conflict is detected during validation, the priorities of the conflicting transactions are examined, and their fate is determined according to the algorithm being used. If the validating transaction has a priority higher than all of the transactions with which it conflicts, the validating transaction will commit, and all the conflicting ones will abort. However, if some of the conflicting transactions have higher priority, the system can choose one of the following options:

(1) OPT-Sacrifice: if at least one of the conflicting transactions has higher priority, then the validating transaction is aborted.

(2) OPT-Commit: the validating transaction is always committed.

(3) OPT-Wait: it incorporates the *priority wait* mechanism such that the validating low priority transaction will wait for the completion of conflicting high priority transactions.

The Wait-50 algorithm is an interesting extension of Opt-Wait: it incorporates a *wait control* mechanism. In Wait-50, a simple "50 percent rule" is used, in which the validating transaction is made to wait while more than half of the conflicting transactions have higher priority. Once that state is reached, remaining conflicting transactions are aborted, irrespective of their priorities and the validating transaction is committed. The goal of the wait control mechanism is to detect when the beneficial effects of waiting, in terms of giving preference to higher priority transactions, are outweighted by its drawbacks, in terms of late restarts and an increased number of conflicts. In other words, it tries to avoid the loss of work already accomplished by the validating transaction. It is a compromising strategy in the sense that we can control the amount of waiting based on transaction conflict states. Their simulation study shows a significant performance gains from Wait-50 over other choices.

Haritsa et al. have also conducted a study on the relative performance of locking-based protocols and optimistic protocols, and concluded that OCC protocols outperform two-phase locking-based protocols over a wide range of system utilization. Huang et al. also conducted a similar performance study of real-time OCC protocols, but on a testbed system, not through simulation [Hua91b]. They examined the overall effects and the impact of the overheads involved in implementing real-time OCC on the testbed

system. Their experimental results contrast with the results in [Har90], showing that OCC may not always outperform a two-phase locking-based protocol which aborts the lower priority transaction when a conflict occurs. They pointed out that physical implementation schemes may have a significant impact on the performance of real-time OCC protocol.

The rationale for OCC is based on an "optimistic" assumption regarding run-time conflicts: if only few run-time conflicts are expected, we can assume that most execution is serializable [Bern87]. Therefore OCC simultaneously avoids blocking and restarts in the optimistic situations. Unfortunately, however, this optimistic assumption on transaction behavior may not always be true in real world situations. In a database system where run-time conflicts are not rare, OCC depends on transaction restarts to eliminate nonserializable executions. The adverse effect of transaction restarts for serialization is that resources and time are wasted. In OCC, because data conflicts are detected and resolved only during the validation phase, a transaction can end up aborting after having used most of the resources and time needed for its execution. When the transaction is restarted, previously performed work has to be redone. This problem of time and resource waste becomes even more serious in real-time transaction scheduling, because it reduces the chances of meeting transaction deadlines.

Another problem of OCC is that of unnecessary aborts. This problem is often caused by the imperfect validation tests used in OCC protocols. Many validation test schemes are based on the intersection of the read sets and write sets of transactions rather than on the actual execution order of transactions, since in general it is difficult to record and use entire execution history efficiently. Hence sometimes a validation process using read sets and write sets erroneously concludes that a nonserializable execution has occurred when it has not in actual execution. Such a conflict can be called a *virtual conflict*. A virtual conflict leads to one or more unnecessary transaction aborts. This problem of unnecessary aborts also results in waste of resource and time, and is serious in real-time transaction processing.

3.3. Dynamic Adjustment of Serializability

Son et al. proposed the concept of dynamic adjustment of serialization order to provide better service to high priority transactions [LS90, Son92]. Transactions write into the database only after they are committed. By using a priority-dependent locking protocol, the serialization order of active transactions is adjusted dynamically, making it possible for transactions with higher priorities to be executed first so that higher priority transactions are not blocked by uncommitted lower priority transactions, while lower priority transactions may not have to be aborted even in face of conflicting operations. The adjustment of the serialization order can be viewed as a mechanism to support time-critical scheduling. The objective of this protocol is to avoid unnecessary blocking and aborting.

The protocol is similar to optimistic concurrency control (OCC) in the sense that each transaction has three phases, but unlike the optimistic method, there is no validation phase. This protocol's three phases are read, wait, and write. The read phase is similar to that of OCC wherein a transaction reads from the database and writes to its local workspace. In this phase, however, conflicts are also resolved by using the transactions priority. While other optimistic real-time concurrency control protocols resolve

conflicts in the validation phase, this protocol resolves them in the read phase. In the wait phase, a transaction waits for its chance to commit. Finally, in the write phase, updates are made permanent to the database. The simulation study in [Son92c] indicates that this protocol offers a significant performance improvement over 2PL with priority abort (also called *high priority scheme* in [Abb88]). One of the main reasons for the improved performance is the reduced number of "useless restarts" and "unnecessary aborts".

4. Non-Serializable Real-Time Transactions

As we have discussed earlier, serializability may not be necessary for real-time transactions. To facilitate more timely executions which meet their deadlines, we may extend the definition of correctness in database transactions. Since real-time systems are used to respond to external stimuli (e.g. in combat systems) or to control physical devices (e.g. in auto-pilot systems), a timely and useful result is much more desirable than a serializable but out-of-date response. As long as the result of a transaction is consistent with the situations of the real world, whether or not the database is internally consistent may not be important to the application. Depending on the semantics and requirements of operations, a real-time system may apply different protocols under different situations.

In this section, we review several techniques for generating non-serializable real-time database schedules. All techniques utilize some semantic information on the temporal or dependency relationship between transactions or data objects. With this extra information, the system may produce a set of schedules that are non-serializable but acceptable to the specific applications.

4.1. External and Temporal Consistencies

Many RTDBS are used to monitor and control physical devices and large complex systems. Since the real world is always changing, it is up to the systems to ensure that their database subsystems are always consistent with the real world. Ideally, a system should guarantee that the database always contains good approximate values of their real-world counterparts. However, this may be too expensive to implement for some applications since there may be too many values to be updated in the database. Another, less expensive, solution is to make sure that all data read by a real-time transaction are close approximates of their current real-world values. The latter approach may be easier to satisfy since only the subset of the database currently used must be guaranteed to have up-to-date values.

Another important issue for real-time transactions is that of temporal consistency. Temporal consistency is the constraint that all data constitute a real-world snapshot (i.e. they correspond to the real-world facts of approximately the same time). If a transaction uses some new fact, mixed with old facts, the transaction may have an erroneous picture about the real world and thus make wrong decisions.

A transaction T_k is a sequence of distinct operations, $a_k^1, a_k^2, \cdots$. Each operation of a transaction accesses only one object. An object, however, may be used in more than one operation in a transaction. A transaction can thus be defined by a sequence of (a_k^i, o_k^i) pairs. A *history H* of a set of transactions T is thus a sequence of operations, (a_k^i, o_k^i), from many transactions. Two histories are *equivalent* if they have

the same effect on the values of database and transaction results. A serializable history is a history which is equivalent to a serial history (i.e. transactions are executed sequentially).

To reason about the desirable history for real-time databases, we define a timestamp for each of the transaction operations. The timestamp of each operation $s(a_k^i)$ in a history is defined to be the clock time when the operation is performed. The timestamp of the object version $s(o_k^i)$ is the time when the version is created.

Given a history of transaction executions, the *external consistency* requirement can be defined by the following equation as in [LJ91]:

$$\forall i,\ |s(a_k^i) - s(o_k^i)| \le \varepsilon_k^i$$

The equation specifies that, for each operation of a real-time transaction T_k, the data used by the operation must be within the valid lifespan ε_k^i of the data. ε_k^i is dependent on the nature of the operation a_k^i. Some operation may require its data to be very consistent with the real world and therefore have a very small ε_k^i value. Others may have no concern for the validity of their data and thus have $\varepsilon_k^i = \infty$. In many practical applications we may have one single value for all operations in a transaction. In other words,

$$\forall i,\ \varepsilon_k^i = \varepsilon_k = V$$

To check for the *temporal consistency* for a transaction, we need to compare the timestamps for all objects read by the transaction. In other words, transaction T_k may require that the timestamps of all objects it reads have a difference not larger than δ_k:

$$\forall i\, j,\ |s(o_k^i) - s(o_k^j)| \le \delta_k$$

Sometimes, only the data in a data set have temporal consistency requirements. For example, the three dimensional attributes of an aircraft location must be temporally consistent whenever they are used in a computation. In that case, we can define a temporal consistency requirement for a data set S in terms of ξ_S:

$$\forall i\, j,\ |s(o_k^i) - s(o_k^j)| \le \xi_S \ \text{ where } \{o_k^i, o_k^j\} \subset S$$

The idea of temporally consistent data set can be compared to the *Atomic Data Sets* (ADS) proposed by Rajkumar. In his thesis [Raj89], Rajkumar proposed an approach to decompose a database into disjoint ADS, and use the modular concurrency control protocol [SLJ88] for real-time database concurrency control. The consistency of each ADS can be maintained independent of the other ADS's. A *setwise two phase locking protocol* is then used to make sure that transactions are run serializably with respect to each of the atomic data sets. However, no concept of temporal consistency is defined in ADS. We believe that the two concepts are compatible and that some protocol can be implemented incorporating both of them.

An extensive set of simulations have been performed in [SL92] to study the performance of various concurrency control algorithms in maintaining the temporal consistency of data in hard real-time systems. A multiversion database model is assumed in the study and the transactions are assumed to be mostly

periodic. The study compares the two-phase locking and the optimistic concurrency control algorithms, and finds that the optimistic algorithm is poorer in maintaining temporal consistency.

4.2. The Compatibility Table

Based on the concept of internal and external consistencies, we can divide the actions in a real-time transaction into two parts: those actions in the *E–part* to enter external events in the database (i.e. to maintain external consistency) and those in the *I–part* to maintain internal consistency. Typically, a transaction may start with actions in the E-part and conclude with actions in the I-part.

In some real-time applications, a transaction cannot afford to wait for the database to regain internal consistency if the transaction has a stringent deadline requirement. In such cases, the database may allow the internal consistency to be ignored temporarily in order to return a result before the transaction's deadline. However, in real-time systems, such transaction executions must still maintain the external consistency. One solution is to guarantee only the part of the transaction which maintains the external consistency. The rest of the transaction may be executed at a lower priority after the result is produced before the deadline.

A similar situation occurs when a transaction B is blocked waiting on an active transaction A to finish. Although A has no deadline constraint, B must be finished before deadline. In this case, B can interrupt A as long as the E-part of A has finished. After B is executed, A may resume its I-part to recover the internal consistency. Thus for B's execution, internal consistency is not guaranteed, but external consistency is maintained. Since internal consistency usually has no time constraint, they may be regained after the external deadline is met.

With the division of the I-part and E-part in each transaction, a transaction compatibility table (TCT) can be defined in a system. During run-time, when real-time transactions are scheduled, the table is inspected to see if they need to wait for transactions arrived earlier. A transaction requiring an externally consistent data set must wait until all updates by its predecessor transactions are finished. A transaction is a *predecessor* transaction of T if it updates some data value that will be used by T. The predecessor transactions for T is denoted as $pre(T)$. To decide whether a transaction T_1 should be executed after T_2, the entry of TCT(T_1, T_2) is examined. The entry has four possible values as follows:

(1) $T_2 \in pre(T_1)$. T_1's execution is dependent on T_2's execution. So if T_2 is ahead of T_1 in the scheduler queue, T_2 must always be executed before T_1.

(2) $T_2 \in pre(T_1)$ but T_2 contains an I-part which can delayed. T_1 may preempt T_2 when T_2 reaches an externally consistent breakpoint (i.e. external data have been entered).

(3) $T_2 \in pre(T_1)$ but T_2's I-part can be skipped if T_1 has executed. Only the E-part of T_2 must be finished before T_1's execution. The I-part of T_2 will not be executed at all.

(4) T_2 is not in $pre(T_1)$. It is acceptable to execute T_1 before T_2 even if T_2 is in front of T_1 in the scheduler queue.

It should be clear that TCT is not symmetric. Using the semantic information in a TCT, a scheduler may achieve better performance by rearranging the transaction schedule. Suppose a transaction T is ready to be executed, and there are several other transactions waiting to be executed before T. The scheduler can compute the expected completion time for T if all ready transactions are executed in the order of their arrival. If T can meet its deadline in this schedule, no adjustment is required and T will be placed at the end of the schedule. If the expected completion time for T is later than its deadline, the scheduler may adjust its starting time by using the TCT information.

One of the issues in using the TCT approach is the size of the table. However, since most of the transactions are probably incompatible, TCT is a sparse matrix in most cases. There are many efficient data structures designed for sparse matrices so the size of the TCT should not be a problem. A more unfamiliar issue is to divide each transaction into I-part and E-part. This can be solved by employing some mechanical method in the compiler to analyze the data flow and separate I-part from E-part automatically. The most tedious work involved in implementing TCT is in deciding the value for each entry. This may require careful analysis of the semantics. More research is still needed to make the approach easily usable.

4.3. Epsilon Serializability and Similarity

Epsilon serializability (ESR) [Pu91] is a generalization of serializability (SR) that explicitly allows limited amount of inconsistency in transaction processing. ESR enhances the degree of concurrency since some non-SR executions are allowed. It allows users to bound the amount of temporary inconsistency. A transaction with ESR as its correctness criterion is called an epsilon-transaction (ET). An ET is a query ET if it consists only of reads. An ET containing at least one write is an update ET. Query ETs may see an inconsistent data state produced by update ETs. An update transaction may export some inconsistency when it updates a data object while query ETs are accessing the same data object.

ESR defines correctness for both consistent states and inconsistent states. In the case of consistent states, ESR becomes SR. In addition, ESR associates an amount of inconsistency with each inconsistent state, defined by its *derivation* (or a *distance*) from a consistent state. To an application designer, this implies that each query ET has an *import-limit*, which specifies the maximum amount of inconsistency that can be imported by it. Similarly, an update ET has an *export-limit* that specifies the maximum amount of inconsistency that can be exported by it. The database system ensures that these limits are not exceeded during the execution of ETs.

ESR has several important applications in real-time database systems. A concrete example of ESR application is replication control in distributed real-time databases. It offers the possibility of maintaining mutual consistency of replicated data asynchronously. A distributed real-time database which supports ESR permits temporary and limited differences among data object replicas: these replicas are required to converge to the standard one-copy serializability (1SR) as soon as all the update messages arrive and are processed. A recent simulation study shows a significant improvement in terms of system responsiveness can be achieved by using ESR in a distributed real-time database system as measured by the number of

transactions that meet their deadlines [Son92].

Another interesting application of ESR in real-time database systems is to use it as a value-based correctness requirement. As long as the changes made to the value of a data object remain within a specified limit, the system can allow on arbitrary order in accessing the data object by concurrent transactions. For example, if an application can tolerate an imprecision upto 5 meters in distance in computing the position of a moving object, a transaction can access the data object without considering the access order of other conflicting transactions, if the inconsistency limit is ensured not to be violated.

A related approach to execute transactions without serializability is to allow similar data to be updated or read in any order. If there are two updates that record similar values in a database, it makes no difference to the future read operations which value is recorded first. This is the concept of *similarity* proposed in [KM92]. Since a real-time database models an external environment that changes continuously, the value of an object in the database can only be *similar* to its physical counterpart. Therefore, for each data object, we can define a region of values that are similar to a specific value. Moreover, two database states are similar if the corresponding values of every data object in the two states are similar.

With the above definition, two *views* of a transaction are said to be similar iff every read event in both views uses similar values with respect to the transaction. A schedule is *final-state similar* to another schedule if

(1) they are over the same set of transactions,

(2) for any initial state, they transform similar initial database states into similar output database states.

Thus a system can produce a schedule which is similar to a serializable schedule as long as the two final states are similar and the views of transactions in the two schedules are similar. Scheduling algorithms using this technique are being investigated in [KM93].

5. Conclusions

The design of RTDBS is still a wide-open area. Meeting the requirements of RTDBS will require a balanced and coordinated effort between concurrency control and transaction scheduling. In this paper, we have reviewed the issues and studied several approaches. The first approach is to combine existing concurrency control protocols with real-time scheduling algorithms. To meet more deadlines, protocols can be modified to favor more urgent transactions. The other approach in designing RTDBS is to explore the non-serializable semantics in real-time applications. Since RTDBS needs to maintain external and temporal consistencies, and real-world data usually have continuously evolving values, we can employ non-traditional measures to meet the deadline of real-time transactions.

RTDBS of tomorrow will be large and complex. They will be distributed, operate in an adaptive manner in a highly dynamic environment, exhibit intelligent behavior, and may have catastrophic consequences if certain logical or timing constraints of transactions are not met. In this paper we tried to answer questions raised by by some of the new characteristics. Meeting the challenges from all of the

characteristics would require more extensive and coordinated research efforts in many of the topics listed below:

- development of modeling techniques for distributed real-time transactions and databases to specify timing properties and temporal consistency in an unambiguous manner. Validity of external data consistency and relationships between consistency constraints and timing constraints need to be easily and clearly specified.
- development of priority-based scheduling protocols and concurrency control protocols that can, in an integrated and dynamic fashion, manage transactions with precedence, resources (including communication resources and I/O devices), and timing constraints. In particular, resource allocation policies and distributed transaction management protocols must be integrated.
- new models and protocols for database fault tolerance under real-time constraints. Since recovery by "undoing" operations may not be applicable in many circumstances, a form of forward recovery may be necessary. Moreover, systems may need to provide uninterruptible minimum services and continue to function during recovery.
- new architecture support for fault-tolerance, for efficient data management, and for time-constrained communication. Important issues in new architecture include interconnection topology, interprocess communications, and support of fault-tolerant database operations. It is essential to have hardware support for fast error detection, reconfiguration and recovery. In addition, new architectures may support real-time scheduling algorithms.

REFERENCES

[Abb88] Abbott, R. and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Study," *VLDB Conference*, Sept. 1988, pp 1-12.

[Abb92] Abbott, R. and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," ACM Trans. on Database Systems, vol. 17, no. 3, pp 513-560, Sept. 1992.

[Bem87] Bernstein, P., V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[Buc89] Buchmann, A. et al., "Time-Critical Database Scheduling: A Framework for Integrating Real-Time Scheduling and Concurrency Control," *5th Data Engineering Conference*, Feb. 1989.

[Gar83] Garcia-Molina, H., "Using Semantic Knowledge for Transaction Processing in a Distributed Database," *ACM Trans. on Database Syst.*, vol. 8, no. 2, pp 186-213, June 1983.

[Hae84] Haerder, T., "Observations on Optimistic Concurrency Control Schemes," *Information Systems*, vol. 9, no. 2, June 1984.

[Har90] Haritsa, J., M. Carey, and M. Livny, "Dynamic Real-Time Optimistic Concurrency Control," *Real-Time Systems Symposium*, Orlando, Florida, Dec. 1990.

[Har91] Haritsa, J., M. Livny, and M. Carey, "Earliest Deadline Scheduling for Real-Time Database Systems," *Real-time Systems Symposium*, pp 232-242, Dec. 1991.

[Hua89] Huang, J., J. Stankovic, D. Towsley, and K. Ramamritham, "Experimental Evaluation of Real-Time Transaction Processing," *Real-time Systems Symposium*, Dec. 1989.

[Hua91] Huang, J., J. Stankovic, K. Ramamritham, and D. Towsley, "On Using Priority Inheritance in Real-Time Databases," *Real-time Systems Symposium*, pp 210-221, Dec. 1991.

[Hua91b] Huang, J., J. Stankovic, K. Ramamritham, and D. Towsley, "Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes," *VLDB Conference*, Sept. 1991.

[Kun81] Kung, H. and J. Robinson, "On Optimistic Methods for Concurrency Control," *ACM Trans. on Database Syst.*, vol. 6, no. 2, pp 213-226, June 1981.

[Kor90] Korth, H., "Triggered Real-Time Databases with Consistency Constraints," *16th VLDB Conference*, Brisbane, Australia, Aug. 1990.

[KM92] T.W. Kuo and A. Mok, "Application Semantics and Concurrency Control of Real-Time Data-Intensive Applications," *Proc. IEEE 13th Real-Time Systems Symposium*, Phoenix, AZ, pp. 35-45, Dec 1992.

[KM93] T.W. Kuo and A. Mok, "Real-Time Transaction Scheduling," manuscript under preparation.

[Lin89] K. J. Lin, "Consistency issues in real-time database systems," *Proc. 22nd Hawaii Intl. Conf. System Sciences*, Hawaii, pp. 654-661, Jan. 1989.

[LJ91] K.J. Lin, F. Jahanian, A. Jhingran and C. D. Locke, "A Model of Hard Real-Time Transaction Systems," Technical Report, IBM, 1991.

[LS90] Lin, Y. and S. H. Son, "Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order," *11th IEEE Real-Time Systems Symposium*, Orlando, Florida, Dec. 1990.

[LL73] Liu, C. L. and Layland, J. W., "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *JACM 20*, January, 1973, pp. 46-61.

[Liu88] Liu, J. W. S., K. J. Lin, and X. Song, "Scheduling Hard Real-Time Transactions," *5th IEEE Workshop on Real-Time Operating Systems and Software*, May 1988, pp 112-116.

[Mok83] Mok, A.K., "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment", Ph.D. Thesis, Massachusetts Institute of Technology, May 1983.

[Pu91] Pu, C. and A. Leff, "Replica Control in Distributed Systems: An Asynchronous Approach," *ACM IGMOD Conference*, May 1991.

[Raj89] Rajkumar, R., "Task Synchronization in Real-Time Systems," *Ph.D. Dissertation*, Carnegie-Mellon University, August 1989.

[SG90] L. Sha and J.B. Goodenough, "Real-time scheduling theory and Ada," *IEEE Computer*, Vol. 23, No. 4, pp. 53-62, Apr. 1990.

[SL92] X. Song and J.W.S. Liu, "How Well can Data Temporal Consistency be Maintained?" *Proc. IEEE Symposium Computer-Aided Control System Design*, Napa, CA, pp. 276-284, March 1992.

[SLJ88] Sha, L., J. Lehoczky, and E. D. Jensen, "Modular Concurrency Control and Failure Recovery," *IEEE Trans. Computers*, Vol. 37, pp. 146-159, Feb 1988.

[SRL88] Sha, L., R. Rajkumar, and J. Lehoczky, "Concurrency Control for Distributed Real-Time Databases," *ACM SIGMOD Record 17*, 1, March 1988, pp 82-98.

[SRL90] Sha, L., Rajkumar, R. and Lehoczky, J. P., "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", *IEEE Trans. Computers*, Vol. 39, No. 9, pp. 1175-1185, Sep. 1990.

[Sha91] Sha, L., R. Rajkumar, S. H. Son, and C. Chang, "A Real-Time Locking Protocol," *IEEE Transactions on Computers*, vol. 40, no. 7, July 1991, pp 793-800.

[Son88] Son, S. H., "Semantic Information and Consistency in Distributed Real-Time Systems," *Information and Software Technology*, Vol. 30, Sept. 1988, pp 443-449.

[Son88b] Son, S. H., guest editor, *ACM SIGMOD Record 17*, 1, Special Issue on Real-Time Database Systems, March 1988.

[Son89] Son, S. H., "On Priority-Based Synchronization Protocols for Distributed Real-Time Database Systems," *IFAC/IFIP Workshop on Distributed Databases in Real-Time Control*, Budapest, Hungary, Oct. 1989, pp 67-72.

[Son90] Son, S. H., "Scheduling Real-Time Transactions," *Euromicro Workshop on Real-Time Systems*, Horsholm, Denmark, June 1990, pp 25-32.

[Son91] Son, S. H., R. Cook, J. Lee, and H. Oh, "New Paradigms for Real-Time Database Systems," in *Real-Time Programming*, K. Ramamritham and W. Halang (Editors), Pergamon Press, 1991.

[Son92] Son, S. H., J. Lee, and Y. Lin, "Hybrid Protocols using Dynamic Adjustment of Serialization Order for Real-Time Concurrency Control," *Journal of Real-Time Systems*, vol. 4, Sept. 1992, pp 269-276.

[Son92b] Son, S. H. and S. Koloumbis, "Replication Control for Distributed Real-Time Database Systems," *12th International Conference on Distributed Computing Systems*, Yokohama, Japan, June 1992, pp 144-151.

[Son92c] Son, S. H., S. Park, and Y. Lin, "An Integrated Real-Time Locking Protocol," *Eighth IEEE International Conference on Data Engineering*, Phoenix, Arizona, February 1992, pp 527-534.

[Son93] Son, S. H., J. Lee, and H. Kang, "Approaches to Design of Real-Time Database Systems," *Database Systems for Next-Generation Applications - Principles and Practice*, W. Kim, Y. Kambayashi, and I. Paik (eds.), World Scientific Publishing, 1993, pp 120-131.

[Song90] Song, X. and J. Liu, "Performance of Multiversion Concurrency Control Algorithms in Maintaining Temporal Consistency", *COMPSAC '90*, pp 132-139, October 1990.

[Stan88] Stankovic, J., "Misconceptions about Real-Time Computing," *IEEE Computer* Vol. 21, No. 10, October 1988, pp 10-19.

[Vrb88] Vrbsky, S. and K. J. Lin, "Recovering Imprecise Transactions with Real-Time Constraints," *IEEE Symp. Reliable Distributed Systems*, Oct. 1988, pp 185-193.

Divide & Conquer Strategies and Underlying Lossless Principles

Harold Szu, Edgar Cohen, and John Wingate
Naval Surface Warfare Center Dahlgren Division,
Silver Spring/White Oak, MD 20309-5000

Abstract

The mathematical principle of global optimization is formulated for massively parallel and distributed processors (MP &DP) by means of divide-and-conquer strategies. The lossless principle which is analogous to the incoherent phenomenon in physics is expressed in terms of a vector velocity V that is derived from the least mean square (LMS) kinetic energy $E = |V|^2/2$. We illustrate the nonlinearity that the sum of the best Traveling Salesman Problem (TSP) solutions in subregions is not necessarily globally the best because of the boundary resultant vectors: $V = A + B$ have a cross interaction terms $(A,B) \neq 0$. In fact, a nearest neighbor connection at the boundary cities represents a longer overall distance than the next nearest neighbor connection between boundary cities. Then, a theorem of orthogonal division error (ODE) for lossless divide-and-conquer (D &C) is proved, and the orthogonal projection is constructed for solving TSP explicitly.

Keywords: Nonconvex Optimization, Simulated Annealing, TSP, Lossless Divide-and-Conquer, Boundary Resultant Vectors, Orthogonal Projections, Recursive Algorithm

1. Introduction

The hypothesis that a Divide & Conquer (D&C) Optimization Strategy shall work for massively parallel & distributed processors (MP&DP) depends critically upon the existence of a lossless mathematical principle for an Instantaneous & Distributed Criterion (I&DC) allowing all divisions/processors to make local decisions which contribute positively to the global optimization procedure. In other words, there should be no requirement during execution for communication with the central processor which would constantly assess tradeoffs among the decisions made by the local units.

This phenomenon has been referred to (by Szu in the 1987 Second Supercomputing Conference at Boston [1]) as the Reporter bottleneck, namely, "Who should do what, where, when, why, and how----6 W's speed bottleneck." The first bottleneck is about 10^9 operations per sec (ops) due to serial machines. (According to Einstein, the speed of light is 3×10^{10} cm/sec, which, for a 30 cm processor length, can only be repeated 10^9 times). Depending on MP &DP paradigms, the second bottleneck was estimated as 10^{12} ops (about thousand times

faster than the first), for most parallel digital computers are operated under a lock-step and clock-cycle mode. The fact was benchmarked as follows: For both Transputers and Hypercubes, the communication overhead costs retarded the speedup factor which was revealed itself when plotted against the number of processors [2]. The tradeoff between the communication cost and the actual execution time will have a diminishing return as the number of processors increases, see Fig. 1.

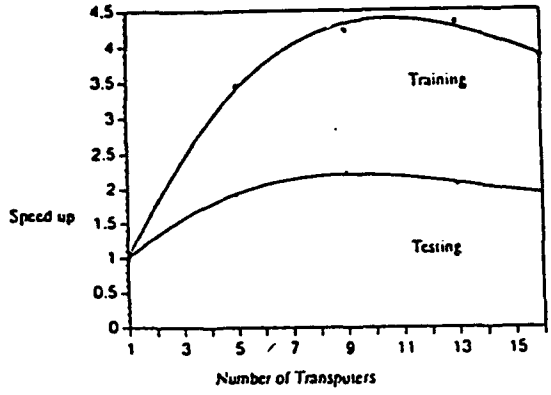


FIGURE 4. Speed up for Toroidal Transputer Network

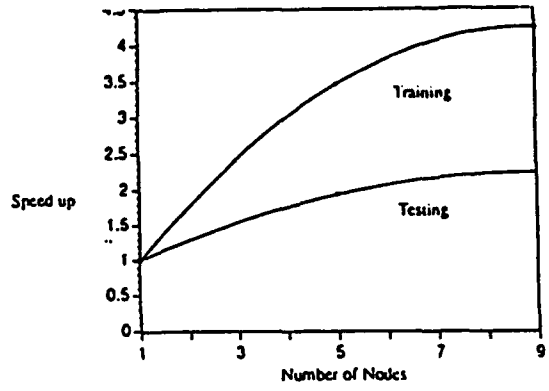


FIGURE 5. Speed up for Intel's Hypercule iPSC/2

However, before we present our solution in this case, we want to motivate the importance of global optimization constrained by a minimum communication, especially in the context when one has the advantage of a using a large parallel computer architecture. For example, ARPA has produced a class of Touchstone with upto 532 processor nodes. It belongs to the class of Multiple Instruction Multiple Data (MIMD) processors under a lock-step and clock cycle operation. Other style computers exist, e.g., the sixth gen artificial neural networks (ANN) of ten billion processors working asynchronously without clock cycle, nor lock steps. A brain-style computes at the speed of Mega-Cray $10^6 x 10^9 ops = 10^{15} ops$ (estimated for 10^{10} neurons x 10^4 synaptic memory capacity x 10 ops each). We donot feel the need of a clock cycle for the lock step communication and execution, because, if we hear tick-tock in our brain as we think, we got to see a psychiatry. Then, what is the underlying mathematics (of such a massively parallel Brain-style computing) that seems to break the barrier of the second bottleneck by three orders of magnitude?

We believe that such a principle, if exists, must help minimize the communication need in a so-to-speak lossless divide-and-conquer strategy. At this point, such a strategy might seem to be solely created by the future generations of computers, but in fact it is mathematically profound and underlies almost all real world constrained optimizations-----management, scheduling, resource allocation, inventory, logistics, global optimization and the military focus on distributed warfare, command & communication[3].

In this paper, a theorem of a lossless D&C strategy is given in Sect. 2 for the LMS global optimization for the first time, and TSP example in Sect. 3.

2. Theorem of Lossless Divide-and-Conquer

The least mean square (LMS) kinetic energy E is defined

$$E = (1/2)\langle (V,V) \rangle \equiv (1/2)\langle |V|^2 \rangle \quad (1)$$

where the angular brackets $\langle\ \rangle$ denote statistical ensemble average, the round bracket (,) the inner product in the Hilbert space. Note that in the continuum medium the vector velocity V is defined to be proportional to the negative of the gradient descent force F direction – grad E = F ≡ dV/dt.

While such a vector V is by Hamiltonian definition a locally conservative quantity, the kinetic energy is (intrinsically global) a scalar quantity. In our new methodology it is crucial to adopt vector V as the linear distributable quantity.

In order to implement properly a divide-and-conquer scheme, one must be able to utilize a distributed criterion. Mathematically speaking, this means that ideally one wants to decompose the LMS problem into two LMS problems such that the division is lossless. If this divide-and-conquer is possible recursively, there must exist two orthogonal projection operators P and Q, such that

$$P + Q = I, \quad (2)$$

the identity operator. (It is not trivial and not always possible to construct P and Q as they must be self-adjoint linear operators.) By such a decomposition of the problem, one can thus eliminate the cross-correlation between the two parts as follows:

$$V = IV = PV + QV \equiv A + B, \quad (3)$$

$$E = \langle (A + B , A+B) \rangle /2 \quad (4)$$

Theorem of Orthogonal Division Errors (ODE)

The divide-and-conquer in two subregions becomes lossless,

$$\langle |V|^2 \rangle = \langle |A|^2 \rangle + \langle |B|^2 \rangle + 2\langle (A,B) \rangle \quad (5)$$

when the the boundary resultant vectors: A and B of two subregions representing

the cross talk contribution become vanishes.

Proof:

The cross terms <(A , B)> vanishes in two possible cases (1) the deterministic orthogonality,

$$(A , B) = 0, \tag{6}$$

(2) random phase approximation of which the statistical averaged to become zero:

$$\langle (A,B) \rangle \equiv 0 \tag{7}$$

Since

$$\text{Optimize} \langle |V|^2 \rangle = \text{Optimize} \langle |A|^2 \rangle + \text{Optimize} \langle |B|^2 \rangle + 2\ \text{Optimize} \langle (A,B) \rangle, \tag{8}$$

then by construction Eqs(6,7) we have

$$\text{Optimize} \langle |V|^2 \rangle = \text{Optimize} \langle |A|^2 \rangle + \text{Optimize} \langle |B|^2 \rangle \tag{9}$$

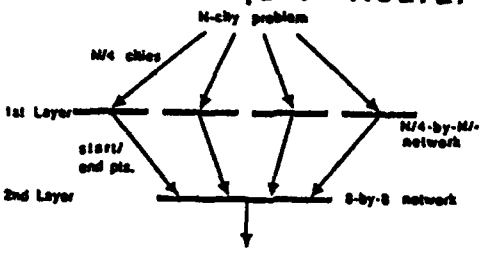
Since energy by definition is real and positive, the optima over the subregions guarantee the optimum of the entire region. Furthermore, the orthogonal decomposition minimizes the communication needs during LMS executions in each subregion. QED

Furthermore, the concept of boundary resultant vector permit an algorithmically recursive implementation as follows. Given V= A+B, each A and B can be furthermore decomposed into a+a′, and b+b′, etc. In this fashion, one can in principle estimate the boundary loss in the divide-and-conquer strategy when the decomposition is not longer orthogonal.

3. Lossless Divide and Conquer of TSP

A case in point is that of the traveling salesmen problem in which one might ask: Is it possible to divide the original set of cities into two parts such that a solution to each part would be useful in obtaining a solution to the entire problem? Suppose that our purpose is to find an optimal tour once and only once through a set of nodes (called cities) in any clockwise sense that the sum of the squares of the distances between nodes is minimal. In Fig. 2 we have divided TSP into 4 quadrants and used a modified Hopfield-Tank Artificial Neural Network to find a local optimum solution for each quadrant. Then, we tried to patch together at the boundary, we discovered that the cut and splice at the nearest neighbor cities denoted by C was not shorter than another cut-and-splice at the next nearest cities denoted by D. This is completely against ones intuition. It points out the nonlinear nature that the total is more than the sum of its parts.

Solving Large-Scale Optimization Problems by Divide-and-Conquer Neural Networks



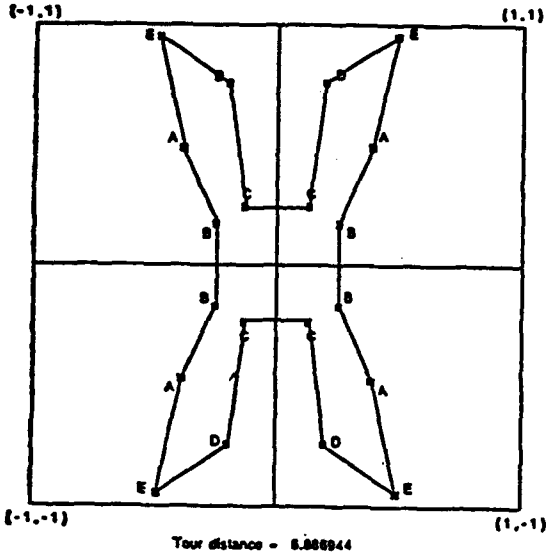


Figure 3. A local optimum tour based on the Divide-and -Conquer Algorithm.

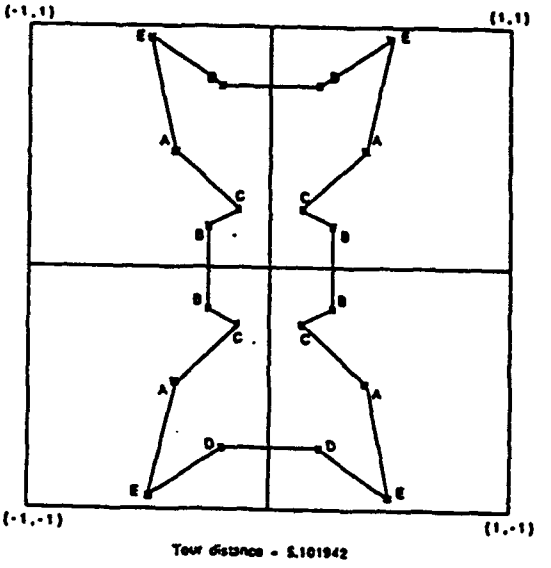
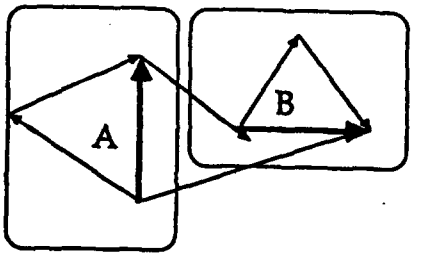


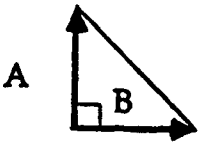
Figure 5. The optimum tour for the 20-city TSP.

Lossless Divide-and-Conquer TSP

Orthogonal Division Error Theorem

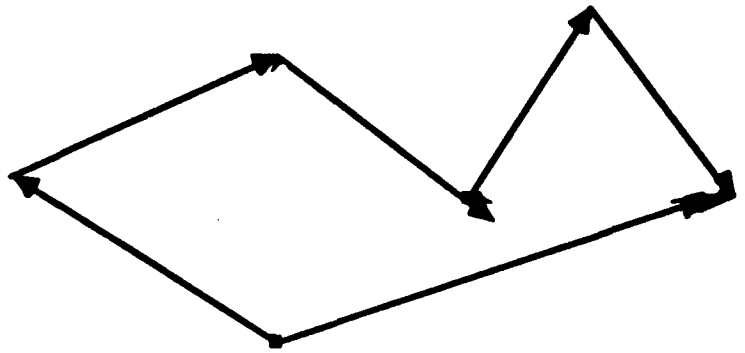
$$\min |A + B|^2 = \min |A|^2 + \min |B|^2 + \min 2(A,B)$$





Pythagoren Theroem

Then a proper division of the original set of nodes into subsets satisfies the orthogonality requirement Eq(6).



The global minimum is obviously obtained in this six cities along the contour around both regions. Several comments are given as follows.

(1) When the boundary resultant vectors A and B are not only orthogonal but also touch each other, by Pythagoren law, A and B are losslessly replaced by the cut-and-splice boundary contour pointing from the head and bottom of the arrow vector A to the head of B.

(2) When the boundary displacement vectors is not orthogonal, then the cross terms shouldbe also minimized. But the communication cost becomes important, and could be compounded in a recursive revision of either A or B in its own subregion. In fact, the antiparallelism $(A,B) \leq 0$ is sometimes preferred for the global minimization.

(3) In other words, the boundary resultant vectors A and B are not orthogonal division error (ODE) theorem for lossless D & C strategy is necessary but not sufficient. If min $|A|^2$ were not the best, there is no way to be sure that the total min $|A+B|^2$ will be the best, and the theorem is only a strategy to minimize the boundary correlation and therefore the communication cost.

(4) Once a proper division is secured, the next problem to be addressed is that of finding optimal tours through the two subsets (induced by the projections P and Q). This is addressed by the simulated annealing algorithm in Sect.4.

4. Simulated Annealing

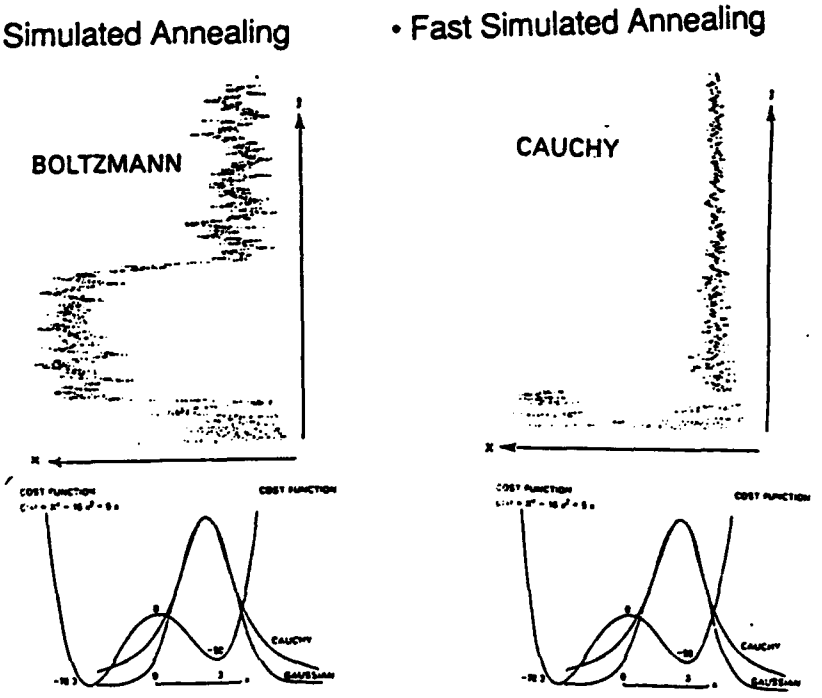
The concept of simulated annealing stems from the pioneering work of Stuart and Donald Geman[5] in 1984, and to work of Kirkpatrick, Gelatt, and Vecchi [6], published in 1983. The seeds of the endeavors of these scientists were in turn

due to the pioneering effort of Nicholas Metropolis, Arianna Rosenbluth, Marshall Rosenbluth, Augusta Teller, and Edward Teller[7] in 1953. Thus, the fundamental methodology is at this point in time, forty years old, but the details of its systematic theory have been addressed only relatively recently. The process is called simulated annealing because its purpose is to emulate a well-known phenomenon encountered in condensed matter physics based on statistical mechanics principles. That specific purpose is the discovery of ground states of systems composed of large numbers of atoms (typically of the order of 10^{23} per cubic centimeter).

Simulated annealing as a mathematical concept is of interest in that it represents a systematic approach to the solution of a large class of nonconvex optimization problems. Whereas most optimization techniques for deterministic problems utilize some iterative, deterministic mechanism or strategy, simulated annealing employs a more global concept which is closely tied to probability theory. Fundamental to this approach are two notions: state generation and state acceptance. Note that classical methods invoke only the first part, namely, state generation and that, therefore, in the case of nonconvex problems, they often yield local rather than global optima. The key to the success of simulated annealing as an optimization principle lies in the state acceptance and, in particular, to the proper coupling of generation and acceptance. Once that is accomplished, one obtains a theory of convergence for global optima, and the task is therefore that of enhancing the rate of convergence by appropriately combining certain generation and acceptance probability distributions. A rigorous convergence theory on multidimensional lattices has been developed as is evidenced by two papers which appeared in the Journal of Applied Probability [8][9]. It is possible to utilize this methodology on continuous variable problems simply by dividing the underlying space into regions of small size and then construing these subregions as entities to which the generation and acceptance laws apply.

As originally contemplated, simulated annealing as a mathematical concept was devised as a sequential algorithm useful for optimizing some energy or cost functions on a multidimensional lattice. As it was observed that the method was "slow to converge" recent work has been concentrated upon expediting the process by utilizing different distributions. For example, thermodynamics dictates use of a Boltzmann law for both state generation and acceptance, and that works properly in conjunction with a temperature schedule which is inversely proportional to the logarithm of time. To enhance the rate of convergence, Szu[12] has recommended using a Cauchy distribution for state generation together with a temperature schedule inversely proportional to time, since the variance of a Cauchy random variable is infinite [13]. Thus, intuition suggests that various parts of the landscape would be visited more rapidly, since a Cauchy law permits "random (Le'vy) flights [14]", in addition to (Wiener) random walks. It turns out that one must then be careful to use an acceptance law compatible with the Cauchy generation law. A

recent discovery is that a Boltzmann law is not the proper one to choose, as then the overall Markov process ceases to be ergodic. One needs to use either an acceptance law which is temperature independent or one based perhaps on a modification of the Cauchy distribution [15].



With the advent of modern digital computers and, in particular, with the production of massively parallel and distributed computers (MP & DP), there has been a shift in emphasis toward that of designing optimization algorithms which can take advantage of this capability. In particular, one would like to capitalize upon the use of such a parallel architecture by dividing the workload among the different processors. In this way perhaps the speed of the optimization technique would not be so critical, since the method would be employed by any given processor on only a small part of the underlying space.

There are indeed two types of optimization problems to be addressed by a multiprocessor architecture: (1) those which, relative to the optimization criterion, can be naturally divided into subproblems whose optimal states are directly related in some manner to the global optimal state. In other words, the global optimum is some function of the optima for the subproblems and (2) those for which a global optimum is not achievable in this way, but, nevertheless, there may be some function of the subproblem optima which yields a satisfactory, though not strictly global, optimal state (a so-called suboptimal state). Under the second possibility, another issue arises, namely, that, by changing the objective function itself, one might be able to convert a problem of type (2) into one of type (1). A primary focus of

our research is to discover the proper objective functions, if they exist, which are naturally related to the globally optimal states in the sense that decisions made locally by the individual elements of a massively parallel architecture are sufficient to obtain the global optimum, thus obviating the need for communication with any central processing facility. The use of force rather than energy as an objective function could be useful here, since force is fundamentally a local concept.

There has been a fair amount of work dating from the 1970's devoted to the partitioning of directed graphs. Such work may be directly relevant to parallel simulated annealing. For example, it should be possible to utilize some of the heuristic procedures given in this literature, together with simulated annealing, to devise "good solutions" to optimization problems. Thus, efficient hybrid approaches would be developed which would expedite the search for satisfactory solutions to NP-hard problems.

Let us illustrate the hybrid approach by appealing to concepts extracted from papers of the type just mentioned. The first paper to be mentioned is one from Kernighan and Lín which actually appeared in 1970 [10]. The main problem addressed in this paper was the following: Partition the nodes of an undirected graph with costs on its edges into subsets of given sizes so as to minimize the sum of the costs on all edges cut. Two applications of the methodology developed in this paper are: (1) Planning of circuit boards and (2) Computer paging properties. We proceed to some implementation details. Let G be a graph of n nodes of sizes (weights) w_i, $1 \le i \le n$, and p a positive number such that $0 < w_i \le p$ for all i. Suppose that $C = (c_{ij})$, $1 \le i,j \le n$, is the cost matrix (weighted connectivity matrix). Now let us define a k-way partition of G. One has k subsets V_i, $1 \le i \le k$, of vertices of G such that

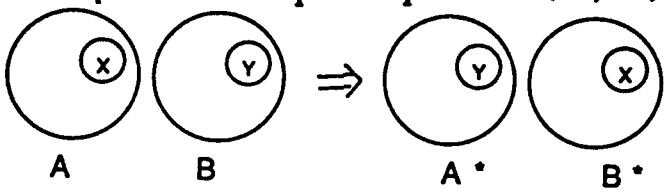
$V_i \cap V_j = 0$, $i \ne j$, and such that $\bigcup V_i = G$. Furthermore, defining $|V_i|$ to be the number of vertices in Vi, an admissible partition is one for which $|V_i| \le p$ for all i. The cost of a partition is just S over all unordered pairs (i,j) with $i \in V_i$, $j \in V_l$, and $V_k \cap V_l = 0$. The authors note that minimizing external cost is equivalent to maximizing internal cost. Furthermore, supposing that kp =n and that one has the task of partition G into k subsets of size p, one find that the total number of classes to be considered is (superscript T denotes the transpose):

$$(1/k!)\,(n\ p)^T(n\text{-}p\ p)^T \ldots (2p\ p)^T$$

For n=40 and p =10 (k=4), the result exceeds 10^{20} cases ! Clearly, one should, in this framework, generally contemplate heuristic solutions. To do so, Kernighan and Lin first consider two-way uniform partitions, wherein the problem is to find a minimal-cost partition of a given graph of 2n vertices into two sets of n vertices

each. In mathematical terms this may be phrased as follows:

Let S be a set of 2n points, $C=(c_{ij})$, $1 \le i,j \le 2n$. Assume that C is symmetric and that $c_{ii}=0$ for i. The quantity c_{ij} is unrestricted in sign. One wants to minimize the external cost $T= \Sigma_{AxB} c_{ab}$, $A \cup B = S$, $A \cap B = 0$, $|A| = |B| = n$. The essence of the method is to start with an arbitrary partition (A,B) of the set of nodes Σc_{ij} and to try to decrease the initial external cost T by a series of interchanges of subsets of A & B. Kernighan and Lin note that a minimum cost 2-way partition is derivable from (A,B) by extracting a certain subset X from A and a certain subset Y from B and then interchanging the two to produce an optimal partition (A*, B*). Diagrammatically this is:

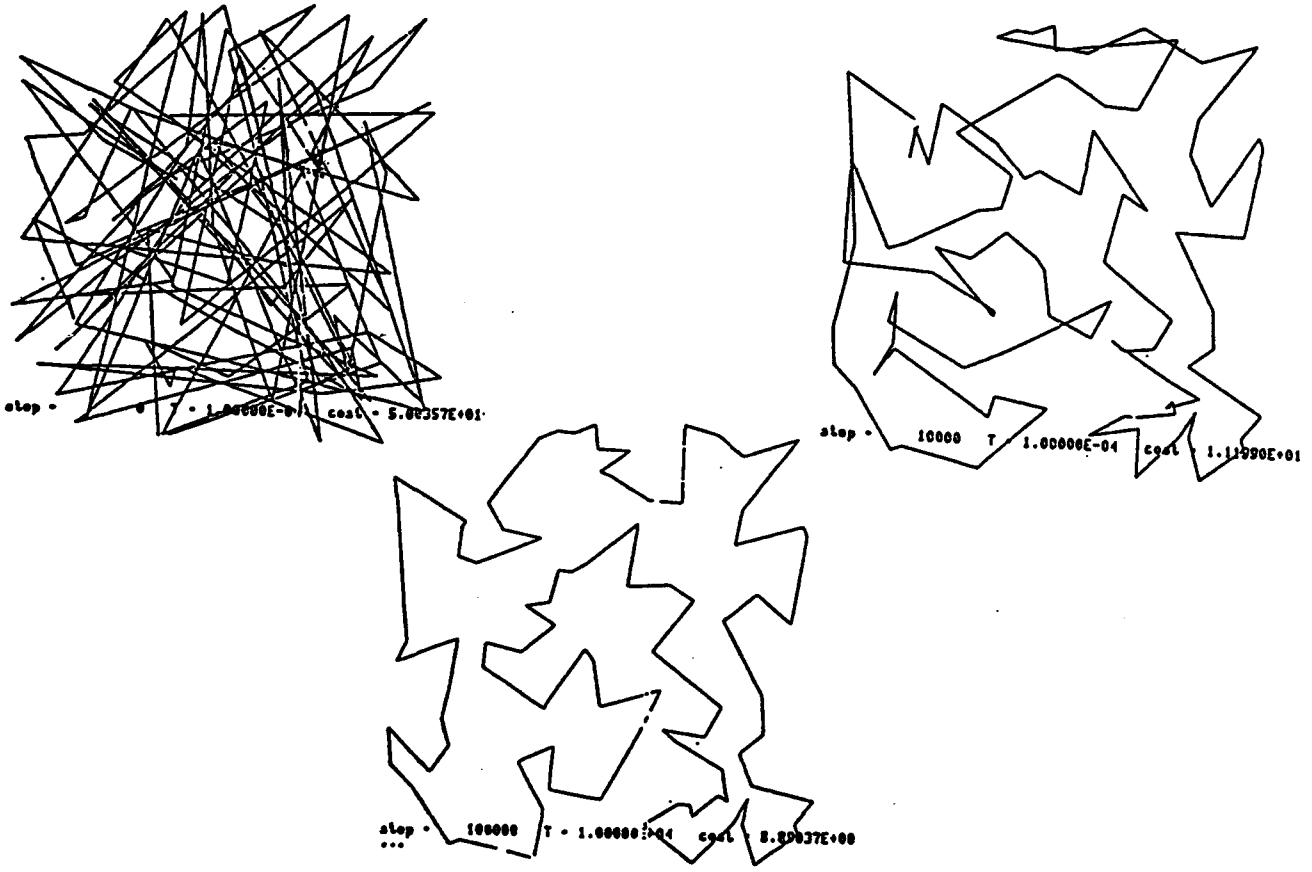


Therefore, $A^* = (A \sim X) \cup Y$, $B^* = (B \sim Y) \cup X$, where $|X| = |Y| \le n/2$. They propose an optimization algorithm for accomplishing this in a systematic manner without having to consider all possible pairs (X,Y) directly. This whole process (a kind of divide-and-conquer approach) may clearly be repeated in order to secure a k-way partition of the original graph G. However, there is an attractive hybrid approach which would utilize simulated annealing. As noted before, minimizing external cost for any k-way partition is equivalent to maximizing the total internal cost of the k subgraphs. This remark is valid, because the total cost for the original graph is constant. Therefore, one may employ the heuristic mechanism of Kernighan and Lin for i steps of their procedure, afterwards appealing to simulated annealing to complete the whole process. The philosophy here is that, once the number of nodes has been satisfactorily reduced, simulated annealing may be a useful tool with regard to both its accuracy and speed, especially when fast simulated annealing is employed. Another viewpoint may also be adduced at this point when problems such as TSP (the traveling salesman problem) are considered. It seems reasonable that good, though clearly not optimal, solutions should be obtained through use of k-way minimal cut partitions, since in TSP one seeks a minimal cost tour. Therefore once such a k-way partition is obtained via the K-L procedure, it would be reasonable to invoke "parallel simulated annealing" on the k subgraphs and thus fuse the results to secure a solution to TSP.

With regard to the TSP in particular, we would finally like to mention some results of an old paper by Richard Karp [11], in which a probabilistic analysis of partitioning algorithms for TSP was conducted. Using a simple partition scheme to divide a rectangle enclosing the original cities into 2k subrectangles each containing at most t cities, where t was given a priori, he showed the following:

There exists a family of algorithms with the property that , for every $\varepsilon > 0$, there is an algorithm $A(\varepsilon)$ in the family such that (a) $A(\varepsilon)$ runs in time $C(\varepsilon)\, n + O(n \log n)$; (b) with probability 1, $A(\varepsilon)$ produces a tour costing not more than $(1+\varepsilon)$ times the cost of an optimal tour. Consistent with the philosophy presented here, his idea was to partition the original region X (a rectangular region enclosing all cities) into "small" subregions, each of which contains about t cities. Then an optimal tour was to be constructed within each subregion (using a computer program TOUR), and the subtours were to be combined to yield a tour through all the cities. Again one can envision the possibility of using parallel simulated annealing in conjunction with the Karp algorithms.

A simple heuristic rule of Lin is based on the triangle inequality of Pythagoren law that $A + B \geq C$. Wherever there is a cross of a tour path, one applies the inequality to uncross the path. This is illustrated with 100 cities over a unit square with a random tour path giving a total distance of about 50, and reduced to 11.1 when some crosses are untangled. when all are eliminated, one obtained 8.89.



It is interesting to apply the algorithm of the lossless Divide-and-Conquer strategy recursively to obtain a global minimum for 1000 cities. This is an ongoing effort.

5. Conclusion

As always it seems that all truth is simple in historical hind sight. There is no exception to such a lossless Divide-and-conquer strtategy. Nevertheless, we have never come across before. It is possible for us to discover the ODE because of two basic realizations: (1) forces, such as resultant dispalcement vectors $V = A + B$ in TSP, are local quantities; but the energy, such as the squared distance kinetic enegy $E=(1/2)|V|^2$, is global and scaler; and (2) the local vector quantity is much easilier to be distributed with much less communication costs. We shall make several intersting philosophical comments before our closing remarks.

(1) Phase transition: This Divide-and-Conquer strategy when is pushed to the extreme to the microscopic world is not unlike the physical annealing phenomena in a phase transition. As a working simulated annealing model of molecular computing for the global minimum crystaline ice state, we must adopt the force (Vander Waal's Coulomb force) that requires minimum or no communication need from the central processor (Mother Nature) giving a local acceptance criteria----"against peer pressure (force)", rather than climbing energy landscape (energy), at a higher temperature than the transition temperature, in order to make distributed decisions to avoid a local minimum. This fact of local force rather than global energy eventually endow the system an ability of finding the global energy state through a self-organized criticality.

(2) Incoherent Sum: The lossless principle is consistent with our common sense that IRS is to make US rich in its global optimization fashion while each individual citizen wishes to be locally optimized to be rich as well. The theroem says that's possible only when no conflict exists, and hence minimum communication is needed. Consequently, for a mutual dependent sociology, such a complete alignment of national and individual interests is unlikely, as would be predicted by the difficulty to fulfil the condition of lossless D & C stategy: $V = \Sigma_i V_i$ which guarantees individual optimization giving the global optimization,

$$\text{Optimize} \langle |V|^2 \rangle = \Sigma_i \text{ Optimize} \langle |V_i|^2 \rangle,$$

because each term is real and positive, (incoherent intensity sum for thoudsand points of light), when the lossless orthogonal division error (ODE) is satisfied.

(3) NP-Complete: To elaborate further the degree of difficulty of finding a general solution of lossless D & C strategy, we mention that the whole class of computationally intractable problems, e.g., the NP-complete problems (nondeterministic polynomial time) such as the Traveling Salesman Problem(TSP) or NP-hard problems such as the four-color mapping problem, would be theoretically solved if one were successful in finding a deterministic procedure for a lossless D&C optimization strategy. To emphasize the mathematical significance of this lossless D&C ODE strategy, we wish to address the general challenge as the 11th

problem of Hilbert beyond the celebrated ten problems, or the von Neumann second bottleneck problem, or the Fermat's last theorem (as gave to Mersenne in 1643 that no integers--x,y,z exist, such that $x^2+y^2 = z^2$ satisfies Pythagoren law and z is a square number and (x+y) is too), of which the boundary resultant vectors A and B is perhaps a special case. We believe, whatever it is referred to, it remains as one of the central challenges of 20th Century computer-civilization, and in the 21 st Century the computerization.

In summary, we have found a mathematical principle for a lossless Divide-and-Conquer strategy by minimizing the communication need. Also, in this paper, we have found a nontrivial practical example TSP to illustrate the losses principle.

Acknowledgement: Support of ONR/ONT program: Engineering of Complex Systems is acknowledged.

References

[1] Szu, H. ," Sixth Generation Computing Architectures,"in K.H. Zhao et al. , eds, Learning and Recognition: A Modern Approach, Singapore: World Scientific, 59-65, 1988.

[2] Szu,H. ,Yeh, C. ,Rogers,G. , Jenkins, M., and Farsaie, A. , "Speed up Performances on MIMD Machines," Int'l Joint Conf, Neural Networks, IJCNN-92, Baltimore, pp. III-742, III-747, 1992.

[3] Layman, G. E. , and Egan, J. T., "Distributed System Architecture for U.S. Navy Battle Group Command Support System,"In 25 Annual Tech. Symposium, Wash. D.C. Chapter of ACM, Gaithersburg MD June 12, 1986, pp. 103-111.

[4] Foo, S. and Szu, H., "Solving Large-Scale Optimization problems by Divide-and-Conquer Neural Networks," Int'l Joint Conf, Neural Networks, IJCNN-89, Wash DC, pp. I-507, I-511, 1989.

[5] Geman, S. and Geman, D. , "Stochastic Relaxation, Gibbs Distribution, and The Bayesian Restoration of Images," IEEE Trans. Pattern Analysis and Machine Intelligence, Vol.6, pp.721-741, 1984.

[6] Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. , "Optimization by Simulated Annealing," Science Vol. 220, pp.671-680, 1983.

[7] Metropolis, N, Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., and Teller, E., "Equations of State Calculations by Fast Computing Machines," J. Chem. Phys. Vol. 21, pp.1087-1091.

[8] Mitra, D., Romeo, F., and Sangiovanni-Vincentelli, "Convergence and Finite Time Behavior of Simulated Annealing," Adv. Applied prob. Vol. 18, pp.747-771, 1986.

[9] Anily, S., and Federgruen, A., "Simulated Annealing Methods with General Acceptance Probabilities," J. Applied Prob. Vol. 24, pp.657-667, 1987.

[10] Kernighan, B., and Lin, S., "An efficient heuristic procedure for partitioning graphs," Bell System technical Journal, Vol. 49, pp. 291-307. 1970.

[11] Karp, R., "Probabilistic Analysis of Partitioning Algorithms for the traveling-salesman problems in the plane, " Math. Oper. Res. Vol. 2, pp.209-224, 1977.
[12] Szu, H., "Fast Simulated Annealing," Snowbird Utah Conf. Am Inst.Phys. (Denker, Ed.)Vol. .pp. , 1987;
[13] Szu, H. and Hartley, R., "Nonconvex Optimization," Proc. IEEE, Vol. , pp. 1987;
[14] Szu, H. and Hartley, R.,"Fast Simulated Annealing," Phys. Letters, Vol. pp. 1987.
[15] Takefuji, Y. and Szu, H., "Design of Parallel Distributed Cauchy machines,"Int'l Joint Conf, Neural Networks, IJCNN-89, Wash DC, pp. I-529, I-532, 1989.
[16] Szu, H, "Colord Noise Annealing benchmark by Exhaustive Solution of TSP, " Int'l Joint Conf, Neural Networks, IJCNN-90, Wash DC, pp. I-317, I-3201, 1990.

A Fault Injection Simulation Testbed for Analyzing Fault Tolerance Protocols

William F. Dudzik
Advanced System Technologies, Inc
5113 Leesburg Pike, Suite 514
Falls Church, VA 22041
703 - 845-0040

ABSTRACT

This paper describes the role of a simulation testbed in analyzing the non-steady state behavior of a fault tolerance protocol underlying the design of the Federal Aviation Administration's (FAA) next generation air traffic control system. The modeled protocol, the group membership protocol, is designed to ensure the consistency of state information among processors that support fault recovery through hardware and software redundancy. The modeling objective was to test the robustness of the protocol in the presence of faults and forward detected problems to the developers for resolution; modeled fault scenarios focused on performance faults attributable to late or lost messages and timers that did not expire on time or at all. The model was implemented using GPSS/vi™ and successfully identified fault scenarios in which protocol behavior was deemed unacceptable, thereby resulting in modifications to the protocol.

Keywords: Fault Tolerance, Group Membership Protocol, Fault Injection, Simulation

1. Introduction

This paper describes the combined use of discrete-event simulation and fault tree analysis in evaluating a distributed protocol, the group membership (GM) protocol, when injected with simulated faults. The GM algorithm is an essential element of the Federal Aviation Administration's (FAA) Advanced Automation System (AAS) program for modernizing the air traffic control (ATC) system. A key requirement of the AAS is high availability. In AAS, downtime (the time critical services are not available) is limited to only a few seconds per year. If the GM protocol fails to function correctly and consistently, then the availability requirements for the AAS can not be met.

As an essential building block of the AAS fault tolerance scheme, the GM algorithm is designed to ensure that each member of a group of processors has the same "view" of the state of all the processors in the group (i.e., working or failed). For example, if a hardware or software element (server) providing a specific service fails, then that event will be detected by all processors so that appropriate recovery actions can be initiated. The state consistency problem is a classical problem in distributed computing whose implementation in AAS must satisfy stringent real time needs.

While it is possible to analyze the behavior of fault tolerance protocols using timelines for simple applications, this approach becomes infeasible for more complex systems containing multiple processors and large numbers of concurrent events. Another approach one could pursue is extensive laboratory testing. Although limited testing in the laboratory has been conducted for the GM protocol, laboratory time is scarce and it is often difficult and time consuming to instrument failure scenarios, for example, those requiring multiple and nearly simultaneous faults. Consequently, a well designed and accurate model of the protocol offers the advantages of assessing many fault scenarios with minimal use of laboratory resources. Regardless, the objective of our analysis effort was to identify fault scenarios which resulted in GM protocol behavior that violated the software developer's design guidelines.

The remainder of this paper is organized as follows. Section 2 outlines the operation of the GM protocol. Our GPSS/vi [1, 2] simulation model of the protocol is described in Section 3. Section 4 summarizes an

example of a protocol deficiency that was identified using the simulation. Finally, our conclusions appear in Sections 5.

2. The Group Membership Protocol

We begin by providing some background information for the GM protocol and then describe its operation in detail.

2.1 Background

A fundamental concept for ensuring high availability of a computing service is the replication of state information on separate processors [3]. Upon recognizing the failure of a peer server on one processor, the surviving servers on the remaining processors have enough state information to resume the work of their failed peer. In order for this scheme to work in practice, the replicated servers must achieve agreement on the global state in the presence of random communications delays, component (hardware and software) faults, and server joins (i.e., the addition of new servers). The purpose of the GM algorithm is to achieve this agreement. The theoretical work underlying the development of the GM protocol was performed by Flaviu Christian while at IBM's Almaden Research Laboratory in San Jose, California [4].

In the AAS design, sets (typically, 2-4) of homogeneous processors (designated groups) provide the hardware redundancy necessary for high availability air traffic applications (Figure 1).

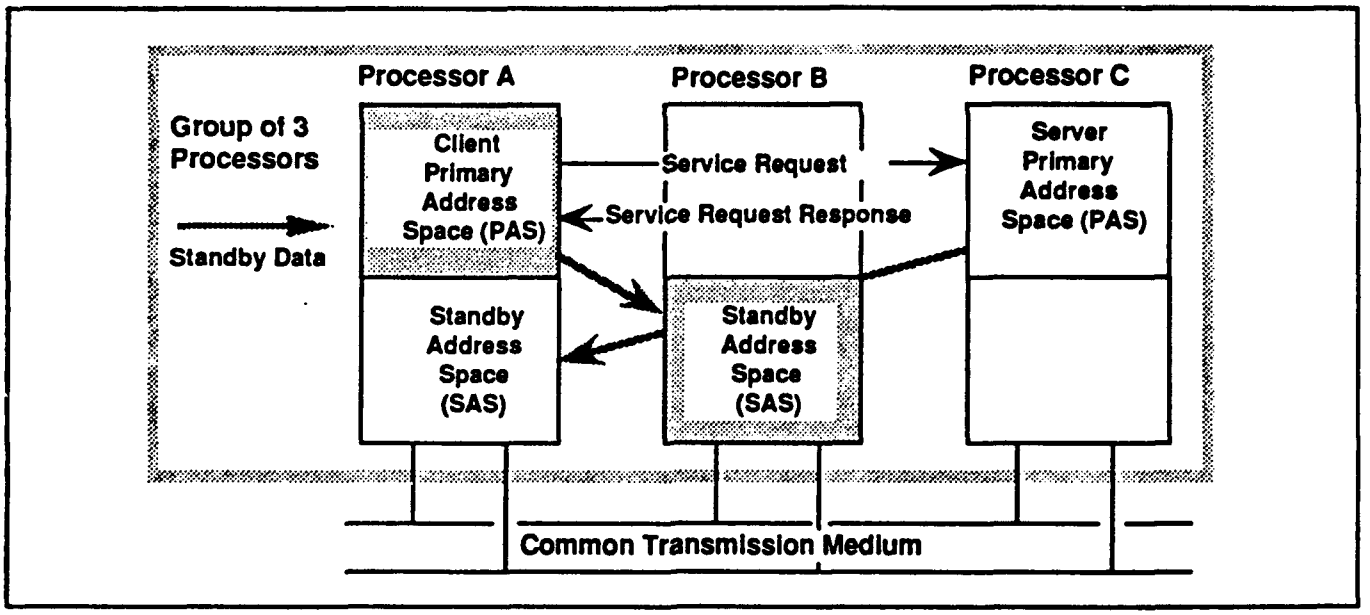


Figure 1: AAS Hardware Redundancy Groups

Transmission of information among processors is accomplished through a common transmission medium. Both clients which request service and servers that provide service are replicated in designated partitions of processor memory called address spaces. The primary copy of client or server software resides in the primary address space (PAS) on one processor while the standby or backup copy (SAS) resides on a separate processor. In Figure 1, the group consists of three processors A, B, and C with client PASs residing on A and C[1], respectively. The corresponding SASs are replicated on B and A. Clients communicate with servers via service requests and responses are returned in service request responses. The gray colored arrows from PAS to SAS denote the periodic flow of status information

[1] The PASs on A and C provide different services.

required to ensure that the SAS is sufficiently current to assume the primary processing responsibilities in the event of PAS failure. If the client PAS on A or A itself fails, for example, then the PAS processing will "switchover" to the client SAS on B.

The GM protocol executes concurrently in each group member. It is designed to detect changes in group composition, propagate those changes, and ensure that the resulting updates are consistent among all members even in the presence of faults. Since servers are supported by hardware resources, processor failures imply server loss but the converse is generally not true. In terms of this distinction, the GM protocol is concerned with the availability of the processors that support those servers. Each group member maintains a set of state information termed the membership view which contains the identification (ID) number of each processor it believes is in the group and the time that its membership view last changed. Generally, the membership view changes whenever processors join or depart the group. A processor may join a group during initial group formation or after being repaired; a processor departs if commanded or if it has failed.

Regardless of the cause, membership changes detected by one member are propagated on the network to the remaining group members via group commands and incorporated into the views of all members at an agreed upon future time. The effect of processing a group command is that all group members update their views at (nearly) the same time[1] and function as if they were a single logical processor.

2.2 Protocol Description

The GM algorithm consists of two processes. The first process (Figure 2a), termed Steady State, detects the failure of existing group members. Steady State uses periodic timers, called Roll Call (RC) and Validation (VAL), which are synchronized[2] among group processors. When the RC timer expires, the VAL timer is started and each group member sends an Accept Roll Call (ARC) message to the other members of the group. The ARC message contains the membership view of its author and signifies that the sender is still working. Upon receipt of an ARC message, the recipient notes that the ARC author has reported. When the VAL timer expires, each processor evaluates its membership and marks as failed those group members which have not reported. The RC timer is then restarted and the cycle continues.

The absence of an ARC is the mechanism by which one processor infers the failure of another. Assuming a group of four processors (A, B, C, D), the first validation time (1.0) in Figure 2a shows the case where all processors (only A's membership view shown) have received ARC messages from the other three and, hence, have the same membership view. Each group member now waits for the next expiration of its RC timer. The second validation event shows the case where processors A, C, and D (only A shown) have received all expected ARCs, but processor B has not received an ARC from processor C (for example, due to a performance fault) before validation time; as a result, B deletes C from its view (now ABD) and broadcasts a Process Membership Check (PMC) message.

The PMC message is broadcast to all group members whenever an existing group member updates its view and indicates that there is a discrepancy among membership views that must be resolved. The PMC message contains the membership view (membership list plus last change time) of its author. The author's view is then compared to the receiver's view; if the views are the same no action is taken. If they differ, we must decide which view is better.

---

1 Note that the "atomic effect" of group commands assumes that the clocks of all group members are synchronized to a specified tolerance.

2 A separate timer synchronization protocol is used to ensure that the clocks for each processor in the group are in approximate agreement.

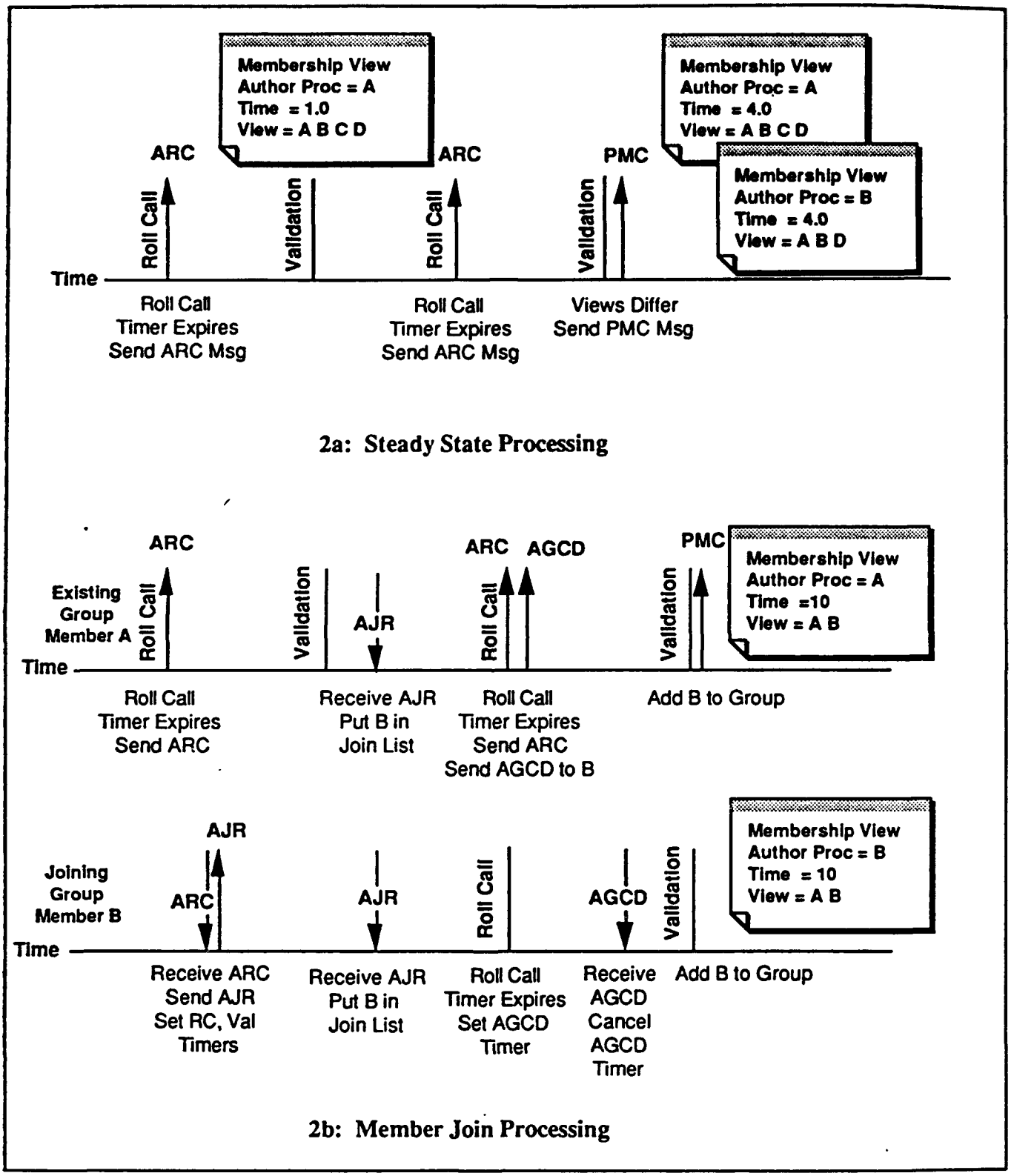


Figure 2: Group Membership Protocol Timeline

View X is defined as better than view Y if one of two conditions holds: 1) X's membership change time is more recent than Y's or 2) if their change times are identical, X's membership list is "greater" than Y's

when the lists of processor IDs are compared position by position. Zero (indicating absence of a processor) is considered to be greater than any non-blank entry. For example, let processors A, B, C, and D have IDs 1, 2, 3, and 4, respectively. View 234 is better than 134 since 2 > 1 and 1230 is better than 1234 since 0 > 4.

To attain view consistency, worse views "conform" to better views as follows. Given that X's view is better than Y's, first determine if X's view is a proper subset of Y's. If yes, then the processor with view Y marks as failed those processors with IDs belonging to the view difference[1]. In Figure 2a, B's view (ABD) is a subset of A's view (ABCD) and, hence, processor A deletes C (view difference of ABCD and ABD). Like A, processor D removes C from its view. Note that to conform to the received better view (ABD), C must shut itself down. The result now is that A, B, and D have updated and consistent views (viz.,ABD). If X is not a subset of Y, then the processor with the worse view must fail itself. It marks itself as failed and broadcasts an Accept Member Depart (AMD) group message. All processors receiving the AMD message update their views at the same time to reflect the failure of the AMD's author.

The second process comprising the GM algorithm allows processors to join a group. There are many special cases. In this paper, we consider the simple case of a processor attempting to join an existing group. The basic notion is that before a processor can join a group, it must synchronize its RC and VAL timers with those of the existing group members and receive a copy of the membership views from the group members.

For the sake of simplicity, assume that the existing group consists of a single member A which Processor B wishes to join. As illustrated in Figure 2b, when Processor B receives an ARC from A it uses the message's time stamp to synchronize its own RC and VAL timers and broadcasts an Accept Join Request (AJR) message to the group. Upon receiving the AJR, both A and B add B to their join list. At the next RC time, B sets its Accept Group Configuration Database (AGCD) timer indicating that it expects to receive a copy of the membership view from A while A transmits an AGCD message to B (A knows the identity of B from the AJR message that it received.). The AGCD message contains the current membership view from the perspective of its author. B cancels its AGCD timer upon reception of the AGCD message and updates its membership view (previously empty) to include A. At this point, both A and B have the same view (A). B is then added to the membership views of both A and B at validation time yielding a common view of AB. A PMC message is then broadcast by A.

Depending on the circumstances, several rounds of PMC commands may be necessary to establish consistent views among group members. Since inconsistencies usually result in removal of processors and the group has finite size, group equilibrium is achieved quickly. Note that the primary role of timers within the protocol is passive; if problems are not detected before their expiration, they serve as a "last line of defense". From a real-time perspective, the values of the RC and VAL timers are envisioned to be on the order of two and one seconds, respectively.

3. Analyzing the GM Protocol Using GPSS/vi

The approach for analyzing the GM algorithm was straightforward. The first step in the analysis was to functionally simulate the algorithm using the interactive simulation environment of the GPSS/vi modeling system. The next step was to develop a systematic fault tree. Initially, our analysis focused on performance faults affecting the messages and timers underlying the algorithm. We assumed that such faults would be caused either by delays in the communications network or, more likely, since the network itself had high redundancy, due to contention delays within processors attached to the network. Faults affecting a single message or timer were considered first, followed by multiple faults of the same type (for example, ARC messages received late [after validation time] at both processors A and B) and finally, selected combinations of faults involving distinct message and timer types. Since the

---

[1] View difference is the set of processors that are not in both views.

combinations of multiple faults was practically unlimited, our initial efforts focused on algorithm correctness for basic single and simple multiple faults.

The GM algorithm model was verified by replicating the results of three fault scenarios that had successfully executed in the laboratory environment. Matches between lab and model results provided evidence that the model correctly represented the algorithm. Matches in this context do not mean that model and laboratory results were sufficiently close in value as in traditional simulation verification exercises but instead that the membership views of each processor as projected by the model exactly matched the corresponding view observed in the laboratory test.

Given a candidate set of fault scenarios, the model was modified to reflect the specific fault(s) being evaluated and then executed. The effects of lost or late messages (and timers) were modeled by using suitable delay values to represent message transmissions or periodic timers. Only a few discrete transmission delay values were of interest in the modeling runs, since an algorithm fault is not triggered unless a message is received after an arbitrary but fixed delay T. Messages arriving before time T are "on time." Messages arriving after time T are "late." The effect of an on time message is independent of its exact arrival time. Similarly, the effect of a late message does not vary with the degree of lateness. In effect, sampling delay times was not necessary which simplified the analysis.

Key model outputs included the membership views of each group member and a detailed trace which annotated the specific model execution and proved indispensable in verifying that fault scenarios had been implemented correctly. Model results were then evaluated to detect instances in which 1) no group member survived, 2) the protocol shut down multiple good (fault free) processors in order to preserve consistency, and 3) performance could be improved by eliminating redundant messages. Relative to item 1, a fundamental design guideline was that at least one processor should always survive. Item 2 refers to those cases where the protocol behaves correctly but in a less than optimal (marginal) manner. Results indicating potential shortcomings with the protocol and candidate fixes were forwarded to the protocol developers for resolution.

4. Detecting a Fault in the GM Protocol

We now describe a specific fault scenario in which model results led to a modification of the GM protocol. Assume that processors A, B, and C with IDs 1, 2, and 3, respectively, belong to a group and that Processor D with the largest ID of 4 is attempting to join the group. Further assume that the AGCD messages from group members A, B, and C are received late by D (i.e., received after validation time). The timeline analysis in Figure 3 was reconstructed from the model run. The following discussion shows that a perfectly functioning three processor group is totally disabled when a fourth processor attempts to join. Although the GM algorithm meets its stated requirements by maintaining a consistent membership view, the end result is far from optimal.

As previously described, the joining processor D broadcasts an AJR message to the group members after synchronizing its RC and VAL timers. At the first roll call after transmitting the AJR message (labeled 1 in Figure 3a), D belongs to the join lists of A, B, C, and D. At validation time, A, B, and C add D to their view which is now ABCD, update the membership change time, and issue PMC messages. Meanwhile, the AGCD timer (scheduled to expire at validation time) for D expires and D believes it is the first processor in the group to be initialized (since no AGCD messages have arrived). D adds itself to the group (which previously was empty), records the membership change time, and has view D; D does not issue a PMC, since it was not an existing group member. D ignores the PMC messages from A, B, and C, since the received views are not better than its own (D>ABCD). At the next roll call time (labelled 2), each processor broadcasts an ARC message and receives at least one view that differs from its own, resulting in another round of PMC messages. A, B, and C each transmit one such PMC while D generates three PMCs, one for each received view that differs from its own. The membership change time is the same (last validation) for all processors.

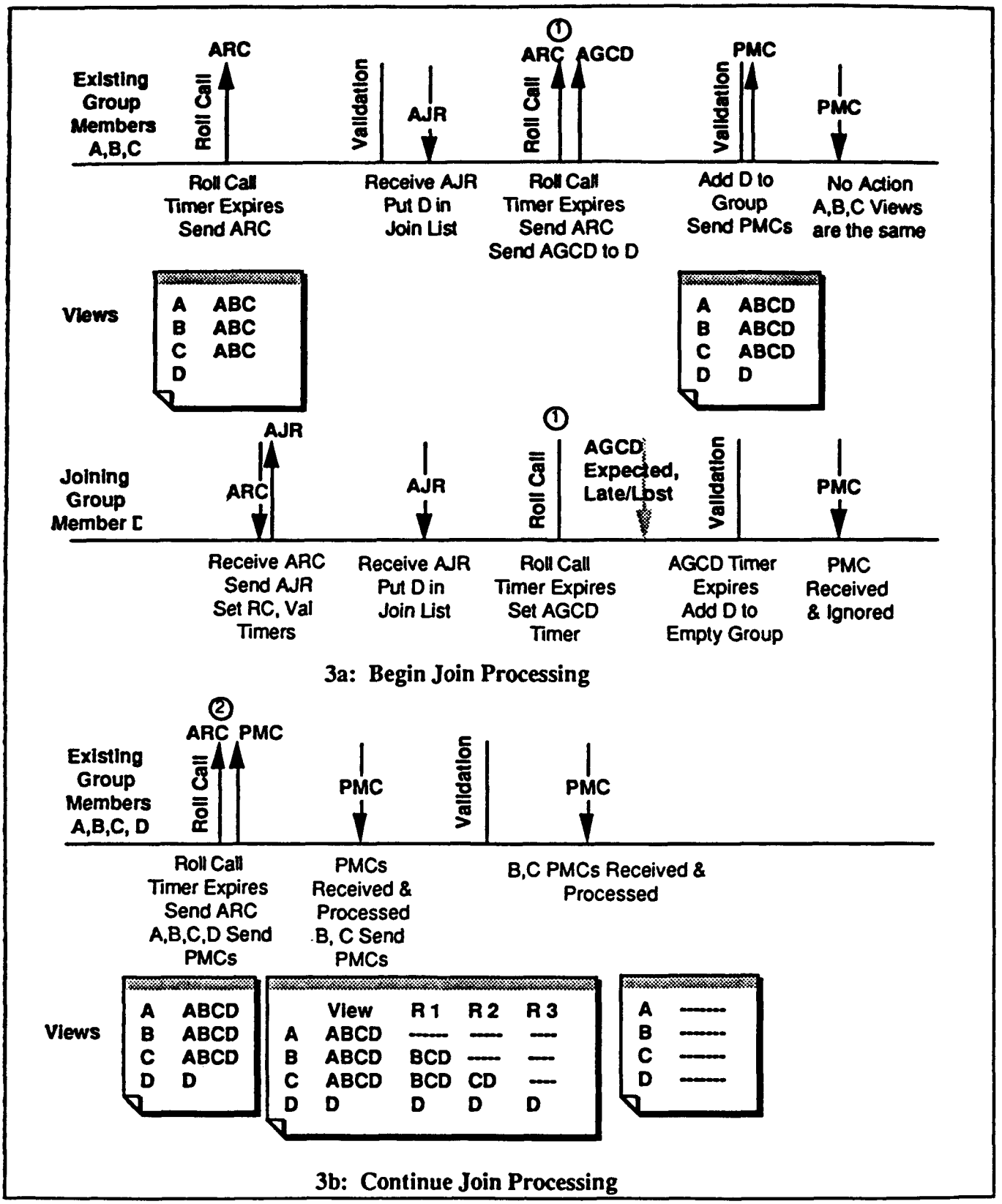


Figure 3: Model Result Identifies Protocol Deficiency

D again ignores the PMCs from A, B, and C, since D's view is still better than the other processors' views. Upon receiving D's view which is better than their own (D>ABCD) and a subset of their own, A, B, and C must fail the processors belonging to the set difference ABCD-D = ABC. According to the protocol, processor failure includes the following three steps: 1) remove the failed processor from your membership view, 2) update your membership change time, and 3) issue a PMC message. This process is repeated for each processor to be failed. Processor A fails the first member in the set difference, namely itself, and shuts itself down. B fails processor A, updates the membership change time, and issues a PMC with view BCD. B then shuts down the second processor in the set difference, namely itself, and ceases processing. Similarly, C issues PMCs after marking A and B down with views BCD and CD, respectively, and then fails itself during the third pass through the shutdown processor logic. These incremental steps are labeled R1, R2, and R3 in the figure. Because the PMCs issued by B and C have more recent change times than processor D, D will shutdown after the next round of PMC messages is processed. The final result is that no group member survives the join process.

This scenario demonstrates that a processor in the act of failing should not issue PMC messages. It is interesting to note that if the group consists of only a single member (a scenario previously tested in he laboratory), then the final result is that the joining processor survives. The results of this scenario together with others highlights what is a common experience; converting an algorithm which works in principle to one which functions as real software operating on real hardware often requires substantial effort.

5. Conclusions

We believe that this type of modeling constitutes a valuable application of simulation. The model was instrumental in identifying cases where protocol behavior was unacceptable (no processor survived certain faults) or where behavior was correct but undesirable, for example, under heavy workload conditions where loss of a good processor could be critical. These cases were forwarded to the developer for evaluation. Their analysis concurred with the model's results and they modified the protocol to correct the implementation errors. These corrections were incorporated into the normal incremental software build cycle and thus prevented problems during the system test phase. Modeling of the GM protocol also heightened our awareness of the difficulties in implementing protocols devised in a laboratory environment and not yet extensively tested in the field.

ACKNOWLEDGEMENTS

The development of the model of the GM protocol was performed under contract to the Volpe National Transportation System Center (VNTSC) in support of the Federal Aviation Administration. We appreciate the reviews and comments from Mr. Paul Connolly of VNTSC and Mr. Ted Page of the FAA's Advanced Automation Program Office.

REFERENCES

1. Ball, Duane: GPSS/vi, Proceedings of Winter Simulation Conference, December 14-16, 1992.

2. Schriber, Thomas J: Simulation Using GPSS, John Wiley and Sons, 1974.

3. Christian, Flaviu: Understanding Fault-Tolerant Distributed Systems, Communications of the ACM, February, 1991, p. 56-78.

4. Christian, Flaviu: Reaching Agreement on Processor Group Membership in Synchronous Distributed Systems, IBM Internal Report, October 1989.

Effectively Using the UNIX make Utility for Permanent and Temporary Changes

David H. Jennings
John J. Reilly
Naval Surface Warfare Center / Dahlgren Division
Strategic and Space Systems Department (K)
Submarine Launched Ballistic Missile
Software Development Division (K50)
9 April 1993

1.0 Introduction

The Systems Simulation Branch (K51) is responsible for the design and programming of large software models, which are typically composed of hundreds of C and Fortran source files. The source, header, and executable files are made available to users in a project directory, which is defined under a common branch directory. During project development and maintenance the developer changes at least one file under his account, while the remaining files are used from the project directory. These changes are referred to as *temporary* changes because the changes are not being applied to the project files. Once the changes are tested and approved, they are ready for incorporation into the formal project. In this case, the updated files are placed back in the project directory and *permanent* changes are made. The changes are *permanent* because they are incorporated into the released project.

The branch chose to use the UNIX[1] Source Code Control System (**SCCS**) for configuration management of the source files and the UNIX **make** utility to build the project files. The UNIX **make** utility provides a powerful tool to create and update object, binary, and library files. **make** accomplishes this by comparing the dates of the prerequisite files with the target. If any prerequisite file is found to be more recent than the target, commands are executed to update the target.

The **make** utility must be given a series of dependencies to determine if a target is out of date. Any file which is dependent on another must be explained via either an implicit rule (.c.o: means a.o is dependent on a.c) or a dependency statement (a.o: a.h means a.o is dependent on a.h). The latter case is important since the dependencies of all header files must be explicitly stated. These dependencies are usually given in a file named *makefile*.

However, **make** is difficult to use, especially given the number of sources and the permanent / temporary changes: permanent changes use all sources, headers and objects in a known directory structure, while temporary changes combine some files in the project space with some files in the user's space. Additionally, temporary changes to a header file may affect files which the user is not currently editing, thus these files must be retrieved, using **SCCS**, into the user's space and re-compiled.

In order to solve these problems and to insulate the developers from learning **make**'s nuances, K51 developed a set of tools (scripts) to automatically generate makefiles capable of handling the temporary / permanent change problem. These easy-to-use tools provide a powerful, consistent makefile that supports the branch's development environment. This paper describes the work performed by personnel in the Systems Simulation Branch (K51) to enhance the use of **make** utility and find solutions to the above problems. Hence it is referred to as the K51 Makefile.

---

[1]UNIX is a registered trademark of AT&T.

2.0 Capabilities

The K51 Makefile was developed to give users an easy interface into a complex UNIX utility. It also does the following:

- creates a consistent makefile for all projects.
- allows target/dependency files to reside in different directories.
- allows permanent changes to be made by updating project files.
- allows temporary changes to be made in a current working directory (cwd), automatically compiling and/or linking the necessary files from project space.
- allows library files to be linked from other projects/models.
- allows simple, consistent targets (temp, perm, etc.) for all projects.
- allows the capability of updating a driver file and library file within the same project.
- allows the capability of creating a driver file *on the fly* in a cwd, even if one does not exist in project space.
- displays a concise help screen when make is entered without a target name.
- supports projects with FORTRAN, C, or a combination of source files.
- allows overriding of makefile macros from either the make command line or by UNIX environment variables.

All makefiles and scripts used by the K51 Makefile can execute in either a Korn or Bourne Shell.

3.0 Makefile Generation

3.1 genmake Command

A utility named **genmake** was developed to generate or update a project makefile. The generated makefile is divided into three sections:

- general macros including those defining paths to files, temporary directory, project name, target name, project source files, and project driver files.
- language macros (C, FORTRAN) and miscellaneous macros.
- command which includes a file containing all targets. This file also includes a file of all needed rules.

3.2 Source/Driver Files

The major purpose of **genmake** is to add a user's project file names to a makefile template.

Based on the value of the input parameter *srctype* (default is *project*) and *gentype* (default is *add*), file names (fnames ...) are added to/deleted from the appropriate macro in the makefile as a function of the file extension. The K51 Makefile supports the following extensions:

- .c C language
- .f FORTRAN with no C Preprocessor directives
- .for FORTRAN with no C Preprocessor directives
- .F FORTRAN with C Preprocessor directives
- .pf FORTRAN with C Preprocessor directives

When files are added the entire list of files is sorted if the input parameter *sort* is set to *yes* (the default).

If the input parameter *srctype* is set to *driver* then the files are added/deleted from the **DRSRC** macro. It is assumed that one or more of these files are combined to produce a driver for the project's library file (.a file) either in project space, in cwd, or both.

4.0 make Command

After generating a makefile, targets can be generated or updated using the **make** command. The general form of the command used with the K51 Makefile is

make [makeflags] [target] [macro definitions]

Many flags are available; however, only a few are most commonly used:

- -e allows environment variable definitions to override the makefile's macro definitions.
- -f specifies description file (default is makefile or Makefile).
- -n displays commands but does not execute them.
- -p prints the complete set of macro definitions.

The target has been defined for all K51 makefiles to include

- **perm**
- **temp**
- **drivert**

Each of the above is described in the sections that follow.

4.1 make perm Command

The purpose of the **make perm** command is to update the project files. When the user executes the **make perm** command in the project directory that contains makefile, a UNIX shell script (**permscpt**) is invoked with all needed values input via positional parameters.

Project dependency information is defined first in the file makefile.dep, which is included in the user's makefile. This is the heart of the K51 Makefile, and the information in this file must be correct for both permanent and temporary changes. The K51 Makefile can accept the project and driver source files either under the control of the Source Code Control System (SCCS) or as ASCII text files. The dependency file (makefile.dep) is composed of five parts:

- header information.
- source dependency information.
- source object files list.
- driver dependency information (optional).
- driver object files list (optional).

The header information contains target, project, date, and user information. To create file dependencies all source files must be processed to determine all references to header (.h) files.

After the dependencies are found, they are processed based on the prerequisite file's extension via an **awk** script and placed in the dependency file. The full path to the object files are then listed under the macro **OBJABS**.

If the project contains a driver, driver file dependency information is appended to makefile.dep. They are computed like the source file generation dependencies, except the object files are listed under the macro **OBJDRV**.

After the dependencies are computed, the object files and target are defined. The K51 Makefile supports the generation of both a binary target (all object files are linked to form a executable image) and a library target (all object files are placed in a random library). If the project contains a driver which calls functions from a library, it is created/modified after the target is updated.

4.2 make temp Command

The purpose of **make temp** is to update local, temporary files without changing the project files. The project's makefile must be present in the cwd to make the **temp** target. When the user executes the **make temp** command, a UNIX shell script (**tempscpt**) is invoked with all the needed values input via positional parameters.

Since it is possible that project files would be changed if the user executes **make** with a makefile.dep file from the project directory, a test is made to prevent this. Also, the user has the ability to force compilation of files other than those determined to be out of date. This is useful when changes to compilation or preprocessor flags are desired.

Dependencies are computed in the cwd and placed in the local makefile.dep file. This contains the same information as the project makefile.dep, except targets are defined to produce local versions of the project object/executable/library files.

Next all header files in the cwd are examined to determine if any project source files are dependent on them. It is crucial for this to be done so that any change to a local header file will

force compilation of a project source file upon which it depends. The project's makefile.dep file is used to determine this.

For all needed source files, a copy of the file is placed in a temporary directory and the C Preprocessor is used to determine all dependencies. Finally, the object file list of those in the cwd is appended to the end of makefile.dep, under the macro **OBJABS**. If a binary target is generated, the full path to the object files that are not generated in the cwd are listed, too. This is not needed for library targets, since the project library is linked instead.

If the project contains a driver, driver file dependency information is appended to the makefile.dep in the cwd.

After dependencies are computed, the object files and target are defined. The K51 Makefile supports the generation of either a binary or library target in the cwd. The target is dependent on all source object files (either in the cwd or project space) and the optional, external libraries. If the base name of the external library is in the cwd, the library in the cwd is used in lieu of the external library. If any of the object files are older than the corresponding source files (either in cwd or project space), then the source file is compiled to produce the object file. If any of the object files or the external libraries are newer than the target, the target is made by linking the object files with the external libraries, or by combining the object files to form a library in the cwd. If the project contains a driver which calls functions from a library, it is created/modified after the target is updated.

4.3 make drivert Command

The purpose of this target is to create an executable driver in the cwd. Dependencies do not need to be defined since it does not use the makefile.dep file. The user must have the appropriate driver source or object files in the cwd before making this target.

This target is useful for users who wish to create drivers *on the fly*, using the project's library file. Different drivers can be created for different purposes, without changing the project's files.

5.0 Summary

The K51 Makefile was designed to enhance the effectiveness of using the basic UNIX **make** utility. It meets the needs of programmers who wish to maintain project files, test temporary changes without changing project files, and use the powerful **make** capabilities without the inconvenience of having to deal with creating and updating the makefile. The **genmake** utility automates adding and deleting source file names from the makefile.dep file. However, the user is still able to edit his makefile and modify previously defined macros. The burden of dealing with the makefile's syntax and keeping the dependencies up to date are

hidden from the user. As long as the macro containing the source file names is kept up to date (using **genmake**), files upon which the source files are dependent are automatically generated and included in the makefile. By entering simple commands (e.g., make perm or make temp), the user can quickly update his project files or test new files with his project without affecting the project files.

Appendix A References

1. Cummings, M., Jennings, D., NAVSWCDD, *K51 Directory and File Naming Conventions*, 27 May 1992.

2. Jennings, D., NAVSWCDD, *K51 Makefile User Guide*, 11 March 1993.

3. Jennings, D., NAVSWCDD, *Make Workshop Notes*, 30 November 1992.

4. Reilly, J., NAVSWCDD, *K51 Makefile Template User's Guide*, 27 February 1992.

Utilization Bounds for Tasksets with Known Periods

Dong-Won Park Swaminathan Natarajan Arkady Kanevsky

Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
(409) 845-8287, swami@cs.tamu.edu

Abstract

Fixed priority scheduling is commonly used to ensure that periodic tasks will be able to meet hard real-time deadlines. There are two major approaches by which we can guarantee that a given taskset can be scheduled according to some fixed priority assignment: utilization bound checks, which check the total expected processor utilization, and do not require detailed information about the taskset; and exact schedulability checks, which use detailed information. In this work, we present a technique for determining period-specific utilization bounds, which use task period information, generally available at design time, but not task computation time information, which is hard to determine accurately. The technique we use for determining the bound is an innovative approach which makes use of linear programming.

1 Introduction

Scheduling hard real-time periodic tasks using system utilization bounds was first developed by Liu and Layland [8]. This technique has gained popularity as an approach to designing predictable real-time systems. They analyzed the *rate-monotonic* scheduling algorithm, in which higher priorities are assigned to tasks with shorter periods. They showed that for this algorithm, all tasks were guaranteed to meet their deadlines provided their combined utilization of the processor did not exceed a certain level (the *worst case utilization bound*). They also showed that this algorithm is optimal among all preemptive fixed priority assignment algorithms for scheduling periodic tasks, for the case where task deadlines are coincident with the end of a task's period.

Subsequent work on this topic has brought up several interesting points:

- Lehoczky, Sha and Ding suggest that the average case behavior of this algorithm is substantially better than the worst case behavior[6]. They showed that the behavior of this algorithm is strongly dependent upon the relative values of the periods of the tasks comprising the task set. This suggests that if we take advantage of information about task periods, we could obtain a taskset-specific utilization bound which would be much higher than the general bound derived by Liu and Layland.
- Leung and Whitehead introduced a new fixed priority scheduling algorithm, the *deadline-monotonic algorithm*, in which higher priorities are assigned to tasks with shorter deadlines[7]. They proved

that the deadline-monotonic algorithm is optimal for the case where tasks have deadlines that are at or before the end of their periods. This suggests that in order to deal with more general situations, it would be useful to be able to derive utilization bounds for priority assignments other than just the rate-monotone priority assignments.

In this work, we develop a technique to determine the utilization bound for a specific task set, where we know the period and deadline of each of the tasks, but we may not know the task computation time. This is the most common situation, since task periods and deadlines depend on the characteristics of the application and are usually fixed at design-time, whereas computation times are very difficult to determine, even after the actual application code has been written. Our technique is applicable to any arbitrary fixed priority assignment, and to situations where some tasks must be complete before the end of their periods, in which case the Liu and Layland bound is not applicable.

This problem fills an important gap between the worst case bound which does not take any task information into account, and exact schedulability tests [6], which require complete information about the task set. This problem is an important one in the design of real-time systems. In general, the arrival rate of each periodic task is fixed at design time. However, it is difficult to determine the computation time of tasks: the computation to be performed may vary from one arrival to the next (it may be *data-dependent*); various system features, such as interrupts, cache memory, virtual memory, I/O, message transmission over networks, resource availability may all cause variations in the execution time of tasks. This difficulty is usually overcome by determining an upper bound on task computation times, and scheduling for the worst-case situation when each task requires this maximum time. However, it may not even be possible to determine a useful worst-case bound in many applications, such as radar tracking, where the computation time depends directly on the number of objects being tracked. Therefore, it would be very useful to determine a utilization bound which takes into account the specific task periods, but does not make assumptions about task computation times. Also, a utilization bound has the advantage over an exact schedulability criterion that it can be used to perform simple schedulability checks at run-time, when tasks may have transient overloads.

This paper is organized as follows: Section 2 develops our technique for determining optimal utilization bounds for tasksets with known periods. Section 3 discusses some of the ways in which our results can be used. In section 4, we present some examples and simulation results to show utilization bounds obtained by our technique. We conclude the paper in section 5.

2 Period-specific Utilization Bound

In this section, we develop a technique to derive the utilization bound for a taskset in which the periods of all the tasks are known, under a scheduling algorithm based on fixed task priorities.

2.1 System Model and Objective

We consider a set of periodic tasks with each task having a deadline before or at the end of its period. We consider preemptive fixed priority assignment where each task receives a unique priority, and higher priority tasks can preempt any lower priority task. Priority assignment is done at design time before any tasks are scheduled. We assume that task periods are known at design time, however task computation times are not known.

Let $\tau_1, \tau_2, \ldots, \tau_n$ be the taskset sorted in the decreasing order of the priorities to be scheduled by preemptive fixed priority assignment. Let $T_1, T_2, \ldots, T_n$ be the periods of the tasks of the taskset, and let $D_1, D_2, \ldots, D_n$ be the deadlines of the corresponding tasks, where each $D_i \leq T_i$. Let B_m be the minimum utilization bound that guarantees the schedulability of τ_m, and let $C_1, C_2, \ldots, C_m$ be a set of positive computation times of tasks $\tau_1, \ldots, \tau_m$ that achieves that bound. We will only analyze the case when all tasks are initialized at the same time, since Lehoczky proved that this is a critical instance of the taskset. That is the worst case scenario when the demand for computation time is the largest.

Our objective is to determine the exact Period-Specific Utilization Bound (PSUB), such that all tasks are guaranteed to meet their deadlines if their combined utilization is less than or equal to this bound, and for any utilization greater than this bound there exist a possible set of computation times for which the taskset is not feasible.

2.2 Determining the Period-Specific Utilization Bound

We propose a technique based on linear programming to determine period-specific utilization bound.

lemma 1 *There is no idle processor time prior to the first deadline of task τ_m.*

Proof: Assume that there is a δ idle time prior to the first deadline of the task τ_m. Then even with the increase of computation time C_m by δ task τ_m will still meet its deadline, but the utilization bound B_m increases. Hence, this violates the assumption that the set of computation times achieves the minimum utilization bound B_m. Since the first deadline of τ_m is the critical instance of the task τ_m, the lemma follows.

□

Next we show that all tasks with higher priority that arrive before the deadline of task τ_m must complete their execution before the deadline of task τ_m in order for B_m to be minimal.

lemma 2 *All tasks with higher priority that arrive before the deadline of task τ_m must be finished before the deadline of task τ_m.*

Proof: Assume that a request of some task τ_i with priority higher than τ_m (so $i < m$) finishes after the deadline of τ_m of its critical instance. Let δ is the amount of computation time task τ_i is executed after the deadline D_m of task τ_m. There are two possible cases.

Case 1: If $T_i \geq D_m$, then since the task τ_i is still executing after the deadline of task τ_m, there is no time for lower priority task τ_m to execute at all. Hence, task τ_m does not has any computation time or the taskset is nonschedulable. In both cases we have contradictions to the assumptions.

Case 2: If $T_i < D_m$, then we create a different set of computation times that reduces utilization B_m but is still schedulable. We simply reduce the execution time of τ_i by δ and increase the execution time of τ_m by the amount of time freed up by task τ_i. So new computation times C_i' and C_m' of tasks τ_i and τ_m becomes,

$$C_i' = C_i - \delta$$

$$C'_m = C_m + \delta \times \lfloor \frac{D_m}{T_i} \rfloor.$$

This transformation preserves the schedulability of the system. Hence, the new utilization of the system $B'_m = B_m + \frac{\delta}{T_i \times T_m}(T_i \times \lfloor \frac{D_m}{T_i} \rfloor - T_m)$. Since the term in the last parentheses is not positive, new utilization can only be smaller or equal to the original one. This violates the assumption that B_m is minimal and the lemma follows.

□

Corollary 1 $\sum_{i=1}^{m} C_i \times \lceil \frac{D_m}{T_i} \rceil = D_m$.

As the consequence of the above we have the following theorem.

Theorem 1 *The utilization bound B_m for the schedulable task τ_m can be achieved by the following procedure:*

Minimize

$$B_m = \sum_{i=1}^{m} \frac{C_i}{T_i}$$

subject to

$$\text{for } 1 \le j \le m-1 \text{ and } 1 \le l \le \lfloor \frac{D_m}{T_j} \rfloor, \ \sum_{i=1}^{m} C_i \times \lceil \frac{l \times T_j}{T_i} \rceil \ge l \times T_j,$$

and

$$\sum_{i=1}^{m} C_i \times \lceil \frac{D_m}{T_i} \rceil = D_m,$$

and

$$\text{for } 1 \le i \le m, \ C_i > 0$$

Proof: The first set of equations expresses the constraint that there should be no idle time in the system, as proved in lemma 1. The second equation expresses the constraint of lemma 2, that there should be no overflow at the time D_m. The third equation simply constrains execution times to be positive. Thus this system of inequalities determines the worst-case combination of computation times which meets these constraints, and has the lowest combined utilization. □

Notes:

- We do not need any additional constraint to express the idea that τ_m meets its deadline; it is automatically ensured by lemma 2.
- Though these inequalities seem complicated, in fact they are identical, but opposite to the inequalities in the exact schedulability condition developed by the Lehoczky, Sha and Ding [5] i.e. the exact schedulability criterion has $\le$ where we have $\ge$ symbols. This is because we are expressing the idea that the task arrivals will at least consume all the available time (no idle time), whereas they are ensuring that all the arrivals will complete within that time, ensuring scheduling feasibility.

- This bound merely ensures that τ_m will meet its deadline as long as the system utilization does not exceed this bound. It does not give similar assurances for the other tasks. We obtain this assurance by iterating this procedure over all τ_i.

Theorem 2 *The period-specific utilization bound PSUB for the system is the smallest B_i for each task τ_i such that*

$$PSUB = \min_{m=1}^{n} B_m$$

Proof: Since $PSUB \leq B_i$ for all $1 \leq i \leq n$, all tasks τ_i are guaranteed to meet their deadlines. The bound is tight, since there exists some task τ_k for $1 \leq k \leq n$ such that $PSUB = B_k$. By theorem 1, this task τ_k does not have any idle time. □

Hence, we can find the PSUB by solving n linear programming problems to determine the utilization bound B_m for each task.

2.3 Discussion

The technique we have outlined above is significant because it enables us to obtain a utilization bound test for any arbitrary fixed priority assignment algorithm. Moreover, for the special case of the rate-monotonic scheduling, it gives us a higher (tighter) bound for specific tasksets than the more general Liu and Layland bound.

In addition to this, there is another, broader significance to this technique. To date, analytical results about the rate-monotonic and other fixed priority scheduling algorithms have followed the techniques introduced by Liu and Layland, and developed further by Lehoczky, Sha and the Carnegie-Mellon group, of identifying and analyzing specific worst-case situations. A significant aspect of our result is that it opens up an alternative door to analyzing the behavior of the system, based on using optimization techniques to identify worst-case behavior. The general approach of setting up constraint equations to model system parameters, and using optimization to derive boundary conditions, is a highly extendible one, and we are already currently using it to solve other similar problems of interest. We are very optimistic about its potential as an alternative analytical tool.

There are several different algorithms for solving linear programming [2, 4, 3]. The best of them gives polynomial algorithms such as [4, 3]. However, the potentially high time complexity of linear programming is not a problem for our application, since we are determining utilization bounds off-line, at system design time.

3 Applications of the result

There are many different ways in which this result can be applied:

- For rate-monotonic scheduling, if we have a taskset with known periods, we can derive a utilization bound and use it to check for scheduling feasibility, even if exact computation time bounds are not available. This technique can also handle some situations where tasks must be completed before the end of their period, in which case the Liu and Layland bound is not applicable.

- Occasionally, we may wish to assign non-rate-monotonic priorities to some tasks; for example, we may wish to raise the priority of some low-frequency, high-criticality task to ensure that it will not miss the deadline even in overload situations. We can use this technique to obtain utilization bounds and guarantee schedulability in these situations.

- System designers may have flexibility in choosing task periods. For many applications, such as monitoring and data acquisition, the requirement is simply to "sample at least every 25 seconds" etc. In these cases, system designers may be able to adjust task frequencies in such a way that the utilization bound is increased (select harmonic frequencies). For example, if other tasks in the system have a period of 12, choosing a period of 24 may lead to a much higher utilization bound than choosing period to be 25. Thus, contrary to intuition, we may be able to get better schedulability while also increasing system performance. We develop this theme further in examples in the next section, and analytical results on this topic can be found in [9].

- In some cases, we may have some (incomplete) information about computation times. For example, we may know the minimum computation time for some of the tasks, and perhaps exact computation times for some others [1]. By modifying the constraint equations appropriately, it is possible to derive progressively better bounds as more information is known about the system, thereby increasing the range of systems which can be guaranteed.

- In addition to these design-time applications of the bound, it can also be used to check schedulability at run-time. Sometimes, the computation times of tasks may be known only at task arrival time when the input data size is known (e.g. radar tracking, where the processing time may depend on number of targets). Utilization bounds can be used to check schedulability by keeping track of the combined processor utilization of all the tasks in the system.

- The individual task utilization bounds can themselves be useful to detect which tasks may overrun in particular overload situations. Careful design can also ensure that critical tasks are less likely to miss their deadlines, by adjusting priorities so that their task utilization bounds are not the critical ones.

4 Examples and simulation

We present some examples and the result of simulation in this section.

Table 1 shows the concept of individual task utilization bounds. Task 2 has a bound of 83.33%. This bound occurs when the computation times of task 1 and task 2 are 100 and 200 respectively. Similarly, we show the bounds for tasks 3 and 4, and the corresponding worst-case computation times. Of course, the computation times of lower priority tasks are irrelevant in determining the bound for higher priority tasks. The table illustrates that task 4 is safe as long as the system utilization is below 98.37%. The task 2 may be threatened if utilization of τ_1 and τ_2 exceeds 83.33%. Also, task τ_3 may be threatened if combined utilization of τ_1, τ_2 and τ_3 exceeds 83.07%. Thus, all tasks are guaranteed to meet their deadlines if their combined processor utilization is equal to or less than 83.07%.

Table 2 shows the results of some simulation studies for determining period-specific utilization bounds for randomly generated tasksets. Task periods were uniformly distributed between 20 and 2400. The table shows the minimum, average, and maximum PSUB for the several tasksets generated. It should be noted that uniform distributions are not likely to produce tasksets with harmonic frequencies, particularly

for large tasksets. In contrast, in real systems, frequencies are multiples of some basic clock cycle, hence they do tend to be harmonic and clustered.

Task	Period	C_{B_2}	B_2	C_{B_3}	B_3	C_{B_4}	B_4
τ_1	300	100		5		5	
τ_2	400	200	83.33%	200		ϵ	
τ_3	605	any		190	*83.07%*	580	
τ_4	1190	any		any		10	98.37%

Table 1. An example illustrating Optimal B_τ and their C_τ in the critical zone

Number of Tasks	$n(2^{1/n}-1)$	*minimum* PSUB	*average* PSUB	*maximum* PSUB
3	77.98%	78.69%	87.47%	99.31%
5	74.35%	77.15%	82.70%	94.57%
10	71.77%	73.79%	77.63%	87.22%
20	70.53%	71.85%	73.91%	77.01%
50	69.80%	70.49%	71.16%	71.78%

Table 2. Simulation Result : Period Range 20 - 2400

5 Conclusion

Our technique fills an important gap between the worst-case schedulability test which does not take any task information into account, and exact schedulability tests, which require complete information about the taskset. System designers can use this technique to obtain a better check of scheduling feasibility. An additional major benefit of our technique is that it opens up an alternative approach based on linear programming to analyzing the behavior of fixed-priority scheduling. We are optimistic about the potential of this approach because of its easy extensibility to a variety of scheduling problems.

6 Acknowledgement

The authors wish to express their sincere appreciation to Dr. Wei Zhao for his comments throughout this work.

References

[1] T. P. Baker and A. Shaw. The cyclic executive model and ADA. *Real-Time Systems*, 1:7–25, 1989.

[2] G. B. Dantzing. *Linear Programming amd Extensions.* Princeton University Press, 1963.

[3] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[4] L. G. Khachian. A polynomial algorithm in linear programming. *Soviet Math. Dokl.*, 20:191–194, 1979.

[5] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Proc. IEEE. Real-Time Systems Symposium*, pages 201–209, 1990.

[6] J. P. Lehoczky, L. Sha, and Y. Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. *Proc. IEEE. Real-Time Systems Symposium*, pages 166–171, 1989.

[7] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[8] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20:46–61, 1973.

[9] R. Menon, M. J. Kim, A. Kanevsky, and S. Natarajan. Improving safety margins in rate-monotone scheduling. *Navy Complex Systems Engineering Synthesis and Assessment Technology Workshop*, pages 371–390, 1992.

A Stochastic Control Approach to Combined Task–Message Scheduling in Distributed Real-Time Systems*

Dar T. Peng† and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

Abstract

Using a stochastic control approach, we address the combined problem of scheduling both periodic tasks and inter-task messages in distributed real-time systems.

First, the concurrent execution of tasks and processing of messages are modeled as a sequence of continuous-time Markov chains. Then, the combined task and message scheduling problem (TMSP) is formulated as a Markov decision process to minimise the expected number of tasks missing deadlines. Both centralised and decentralised solutions to the TMSP are derived using the dynamic programming technique.

For the centralised case the global, up-to-date information on the execution of tasks and processing of messages at each computing node (CN) is assumed available to all other CNs so that an optimal scheduling decision can be made. For the decentralised case, however, each CN makes scheduling decisions using only its own local information and other CNs' information which are periodically broadcast. Illustrative examples are presented and optimal broadcast frequencies are determined.

Index Terms — Centralised/decentralised task scheduling, task deadlines, Markov decision process, one-step delayed sharing information pattern, probability state, stochastic control, dynamic programming.

*This work has been supported in part by the Office of Naval Research under Grant No. N00014-92-J-1080. Any opinions, findings, and recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the funding agency.

†Current address: Microelectronics and Technology Center, AlliedSignal Aerospace Company, 9140 Old Annapolis Road, Columbia, MD 21045. E-mail: dtp@batc.allied.com

1 Introduction

In a real-time system, the normal workload is composed of a set of *periodic* tasks, which is known *a priori* and usually pre-assigned to the computing nodes (CNs) of the distributed system for execution. Generally, these tasks communicate with one another to accomplish the overall mission, and inter-task communications introduce precedence relations among the corresponding parts, called *activities*, of the communicating tasks. Owing to its data-dependent conditional branches and loops, an activity usually takes a random amount of time to complete.

The main objective of this paper is to formulate and solve the problem of scheduling *both* periodic tasks and inter-task messages in a distributed real-time system such that the long-term expected number of periodic tasks missing deadlines is minimized.

Our combined task and message scheduling problem (TMSP) deals with each CN's decision on the execution of its periodic tasks as follows:

D1. Which of ready activities in a CN should be executed next if the number of free processors at the CN is less than that of ready activities?

D2. Which of the messages arrived at a CN must be processed next?

D3. While waiting for a specific message to arrive, the CN either continues to wait for the message or abandons the waiting and executes, instead, a certain *default activity.*

D1 and D2 are typical problems addressed in the scheduling domain, while D3 needs further elaboration. As mentioned before, periodic tasks communicate with one another for synchronization and information exchange. The communicating partners involved are usually blocked (e.g., [14]) until the communication is completed, meaning that no activities requiring the information are allowed to continue. This blocking communication scheme could result in a situation where one or more of the communicating partners are delayed indefinitely (until the end of their period) waiting for a message which may never arrive. This could be due to, for instance, the failure of the sending CN or a communication link or termination of the replying task. To ensure the timely completion of each task, there must be a provision for a communicating partner to carry on its execution even in the absence of the requested information. Of course, to compensate for the missing information, some form of *default activity* should be invoked. This default activity introduces an extra

computation load to the corresponding CN. It is D3 that deals with the decision on whether to wait for an outstanding message or to immediately execute the default activity and use the execution results instead.

Notice that D3 can equally well be interpreted as a CN's decision between two tasks. That is, while executing a task, called a *primary*, the CN either continues to execute the primary or abandons the primary and executes, instead, a certain *replacement* task. For instance, we may use the fifth control law as the primary and the third control law, which is less accurate but needs less time to complete, as its replacement. D1-D3 are actually three dependent parts of a single problem, since neither of them can be solved without considering the others.

For a set of independent periodic tasks each with fixed execution times, the *rate monotonic* scheduling algorithm [7] has been rigorously studied [15, 8]. Liu *et al.* [9] considered using *imprecise* computation results as long as the *mandatory* part of the task meets its deadline. To the best of our knowledge, however, the TMSP, which includes *dependent* periodic tasks and *random* task execution times, has not been addressed in the literature, perhaps because of the difficulty associated with it [12].

Based on the continuous-time Markov chain (CTMC) model of task executions, we will first transform the TMSP into a Markov decision process (MDP) [13]. The MDP is then solved with the dynamic programming (DP) technique [2]. As described in [11], the CTMC model is built typically via the construction of a generalized stochastic Petri Net (GSPN) [10].

The rest of the paper is organized as follows. In Section 2, we show how the notion of default activity is modeled in the GSPN, and then describe briefly the method of [11] to build the underlined CTMC model. The centralized TMSP is formulated and solved in Section 3. In Section 4, we solve the decentralized TMSP for which each CN periodically broadcasts its local state to other CNs. Using this solution, we also identify the optimal frequency of state broadcasts. For both centralized and decentralized TMSPs, the computational complexities of the proposed solution algorithms are also addressed briefly in Sections 3 and 4. Finally, concluding remarks are made in Section 5.

2 The System CTMC Model

Since the state-space approach is to be used to the TMSP, we need to show first how the system CTMC model is constructed. We begin with the GSPN

model of the default activity. Then, a typical method of [11] to construct the CTMC model is briefly described.

2.1 GSPN Model of Default Activity

A default activity is extra work that a CN can choose to perform instead of waiting for an outstanding message. To ensure not to waste the computing resource in a CN, default activities are implemented as follows. While waiting for an outstanding message x, the CN will execute all ready activities including the default activity R_x provided that there are enough free processors in the CN. Otherwise, the CN may or may not choose to execute R_x. Then, the CN's next action will depend on the following two cases:

C1. R_x is started and completed before x arrives. The CN uses the results of R_x and proceeds as if the precedence constraints imposed by x were met. No message is sent back to unblock the communicating partner from which x was originated.

C2. x arrives before completing (starting) R_x. The CN stops executing (starting) R_x, immediately processes x and, if needed, sends a message back to unblock its communicating partner.

The GSPN model of a default activity is shown in Fig. 1, where the typical communication primitives SEND-RECEIVE-REPLY (Fig. 1(a)) and QUERY-RESPONSE (Fig. 1(b)) have been used. Notice first that a "control place" has been created in the sending (querying) side to assure the CN be unblocked by either receiving r (t) or completing R_r (R_t), but not both. Second, unlike SEND-RECEIVE-REPLY, the task being queried is not blocked by the QUERY request (Fig. 1(b)). Third, if the receiving CN in Fig. 1(a) gives up on waiting for message s and completes the default activity R_s, then the sending CN will eventually be forced not to wait for the associated reply message r, which will never arrive, by executing the default activity R_r. On the other hand, if message s arrives before R_s is completed then, from C2, the receiving CN must switch immediately to processing s provided there is a free processor.

2.2 Constructing the System CTMC Model

Consider a distributed real-time system, where the tasks have been pre-assigned among the set $\{N_k : k = 1, 2, \cdots m\}$ of m CNs. Since each task repeats itself at regular intervals, it is sufficient to solve the TMSP within the *planning cycle*

$I = [0, L)$ only, where L, the length of I, is the least common multiple of all task periods. For simplicity, we assume that each task is invoked simultaneously at the beginning of I and must be completed by its next invocation time; otherwise, it will simply be discarded. For completeness, a typical method [11] to construct the system CTMC model is briefly reviewed in what follows.

The CTMC model to be built is described by its state space and state transitions. Each state represents a particular stage of the concurrent executions of tasks on *all* CNs, whereas a transition represents either a task invocation (*time-driven* transition) or the completion of an activity (*event-driven* transition). Consequently, the resulting CTMC model is composed of a sequence of l CTMCs, where l is the number of distinct task invocation instants in I. Within each individual CTMC, the system evolution is determined only by the event-driven transitions representing the completions of activities.

The CTMC modeling procedures are summarized as:

- To alleviate the problem of state space explosion, contiguous stretches of executable codes are combined, whenever possible, to build the smallest number of activities while preserving all precedence constraints among tasks and the expected task execution times.

- The GSPN is used to model the concurrent execution of activities and their precedence constraints. The resulting GSPN model is then transformed into a sequence of CTMCs by performing reachability analysis on the GSPN, and replacing each firing delay (event-driven transition) with an appropriate exponentially distributed random variable.

Suppose the tasks are invoked at time instants $0 = w_1 < w_2 < \cdots < w_l < L$. The resultant CTMC model can thus be described formally as $\{(S_u, \Lambda_u, \Theta_u) : u = 1, 2, \cdots, l\}$, where S_u is the set of states reachable within time interval $I_u = [w_u, w_{u+1})$, and $\Lambda_u : S_u \times S_u \rightarrow T$ is the event-driven transition function between any two states in S_u, where T is the set of all event-driven transitions. We use a_{ij} to denote the activity representing the transition from state s_i to s_j in S_u, and use $\mu_{i,j}$ to represent the *transition rate* (i.e., the execution rate) of a_{ij}. Finally, $\Theta_u : S_u \rightarrow S_{u+1}$ is the time-driven transition function, which specifies for each state in S_u a particular state in S_{u+1} the system will be in when a task is invoked at w_{u+1}.

For example, consider the simplified task system in Fig. 2 to be used throughout the rest of the paper. It shows a GSPN model where three periodic tasks T_1, T_2 and T_3 are pre-assigned to N_1, N_2 and N_3, respectively.

T_1, T_2 and T_3 with periods 5, 10 and 5 are invoked twice, once and twice, respectively, within the planning cycle $I = [0, 10)$. T_1 queries T_2 for information, whereas T_3 communicates with T_2 for synchronization. Suppose default activities, labeled as a_{14} and a_{15} in Fig. 2, are provided only for T_1 related to T_2's response; no default activities are provided for synchronization between T_2 and T_3. Branch conditions follow activities a_8 and a_{11}. To guarantee the successful synchronization between T_2 and T_3, we assume these two branches are identical.

At the beginning of I, a *token* is generated in the place ϕ_1 (thus, ϕ_2, ϕ_4 and ϕ_5), ϕ_{19} (ϕ_{20} and ϕ_{22}) and ϕ_{27}. At any time $t \in (0, 5)$, states evolve based on event-driven transition firings only. At $t = 5$, when T_1 and T_3 are invoked again, the *marking* of the GSPN is determined as follows: 1) a token is generated in ϕ_{10} (thus, ϕ_{11}, ϕ_{13} and ϕ_{14}) and ϕ_{32}, 2) all tokens in ϕ_1–ϕ_9 and ϕ_{27}–ϕ_{31} are removed to discard the unfinished first invocations of T_1 and T_3, and 3) the tokens in ϕ_{19}–ϕ_{26} are still determined by event-driven transition firings. At $t \in (5, 10)$, the state evolution is again determined by event-driven transition firings until $t = 10$, when the system repeats itself for the next planning cycle. Let $\mu_1 = \mu_3 = \mu_4 = \mu_6 = \mu_{15} = 2$, $\mu_2 = \mu_5 = 4$, $\mu_7 = \mu_{10} = \mu_{11} = \mu_{12} = \mu_{13} = \mu_{14} = 1$, $\mu_8 = 6$, $\mu_9 = 4$, and branching probabilities $p = 2/3$, where μ_j is the execution (i.e., transition) rate of a generic activity a_j. After performing reachability analysis on the GSPN, a total of 90 (108) states are generated within $I_1 = [0, 5)$ ($I_2 = [5, 10)$). Appendix A lists all the states where N_2 needs to make a scheduling decision because more than one scheduling option is available.

From the brief descriptions above, it can be seen that the CTMC model fully describes the behavior of the system under all scheduling decisions of the CNs, each of which may even have any number of processors. Based on this model, it is our purpose to derive the optimal scheduling decision for each CN such that the expected number of tasks (invocations) missing deadlines is minimized.

In the centralized case to be addressed in the next section, we assume that each *global* system state $s_i \in S = \bigcup_{u=1}^{l} S_u$ is available to all CNs. To facilitate the analysis of the decentralized TMSP in Section 4, the local state available to the individual CN needs to be identified first. This local state is embedded in the global state s_i and can be defined by N_k's local state space S_u^k within I_u. S_u^k is constructed by first identifying those GSPN places associated with N_k only and then selecting the markings of such places from each state in S_u. An element in S_u^k thus represents the information available to N_k only. For

example, if s_i denotes the global state "a_{14} has been completed on N_1, a_7 has been completed on N_2, but delay a_{10} has not been completed", then N_1's local state corresponding to s_i denotes "a_{14} has been completed on N_1" only. For more details, the readers are referred to [11].

3 Optimal Centralized TMSP

By discretizing the planning cycle I into a large number of small intervals, the TMSP becomes a problem of solving the Markov decision process (MDP) embedded in the system CTMC model. The technique of dynamic programming (DP) is to be used to solve the MDP. Although the DP is a well-known technique, the main issues of solving the MDP with the technique lies in the correct identification of the following characterizing elements of the MDP: *decision set, one-step transition probability,* and *one-step cost* so that *functional equations* can be established for an optimal solution (see, e.g., [2]). Assume the planning cycle I has been divided into $V \gg 1$ equally spaced intervals each of length $h = L/V$, and each task invocation always occurs at an *epoch* defined as the left boundary of an interval.

3.1 Decision Set

The decision set D_i for state $s_i \in S$ is the set of all scheduling options available when the system is in s_i. D_i is determined by combining (over all CNs) the set of scheduling options available to each individual CN. For example, if N_1 has 2 scheduling options and N_2 has 3 in s_i, then D_i contains a total of $2 \times 3 = 6$ scheduling options. A *policy* π specifies which activities to choose for each CN in each state $s_i \in S$ at each epoch v, $0 \leq v \leq V-1$. The *policy space* Π is the set of all such π's. For convenience, the set of activities chosen by all CNs under π in state s_i at epoch v is denoted by $\Omega_i^\pi(v)$.

3.2 One-Step Transition Probability

The epochs serve as the *stages* of the corresponding DP network [2], i.e., transitions occur only between states of adjacent epochs. Let $P_{ij}^\pi(v)$ denote the one-step transition probability from state s_i at epoch $v \in I_u$ to s_j at epoch $v+1$ under policy π. Also, let Z_i be the total transition rates of all transitions in $\Omega_i^\pi(v)$, and A_i be the set of all destination states of transitions in $\Omega_i^\pi(v)$, i.e., $A_i = \{s_j : \Lambda_u(s_i, s_j) \in \Omega_i^\pi(v)\}$. Depending on whether or not a task is invoked at epoch $v+1$, $P_{ij}^\pi(v)$ is determined as follows.

P1: $w_{u+1} > (v+1)h$ (epoch when no task is invoked):

$$P_{ij}^{\pi}(v) = \begin{cases} \mu_{ij}\, h & \text{if } s_j \in A_i \text{ and } j \neq i \\ 1 - Z_i\, h + \mu_{ii}\, h & \text{if } j = i \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

because the execution time of each activity is assumed to be exponentially distributed. ($\mu_{i,j}$ is the transition rate from s_i to s_j.)

P2: $w_{u+1} = (v+1)h$ (epoch when at least a task is invoked):
The transition probability in this case is similar to that of P1 except that the time-driven transition function Θ_u is fired. Specifically,

$$P_{ij}^{\pi}(v) = \sum_{s_x \in B_j} P_{ix}^{\pi}(v), \quad s_j \in S_{u+1}, \tag{2}$$

where $B_j = \{s_x : \Theta_u(s_x) = s_j\}$, and $P_{ix}^{\pi}(v)$ is the transition probability obtained from P1 above.

3.3 One-Step Cost

To derive the one-step cost, we need to identify the cost associated with each state s_i first. To do this, the notions of *goal state* and its *marked set* are introduced as follows. In the CTMC model, concurrent task executions can be viewed as the movement of tokens in the corresponding GSPN. A place in the GSPN is said to be marked if it has a token in it, implying the completion of all activities preceding the place. In other words, a deadline in the task system is essentially associated with the marking of a place. A place is time-critical if a deadline is associated with it. A state s_i containing a non-empty set M_i of marked time-critical places is then called a goal state with the marked set M_i. Since a task is considered completed only after all its activities are finished, a time-critical place is located only at the conclusion of the task containing the place.

Let Ψ_u and $\Phi_i = \Psi_u - M_i$, $1 \leq u \leq l$, denote, respectively, the set of all those places of s_i that should be marked and that have not been marked at the next task invocation time w_{u+1}. Since whether or not a task misses its deadline is not known until its next invocation, it is natural to set the cost function — the expected number of tasks missing deadlines when the system is in s_i at epoch v — of the underlying MDP as:

$$c^{\pi}(v, s_i) = \begin{cases} 0 & \frac{w_u}{h} \leq v < \frac{w_{u+1}}{h} - 1 \\ \sum_j P_{ij}^{\pi}(v)|\Phi_j| & v = \frac{w_{u+1}}{h} - 1, \end{cases} \tag{3}$$

where $P_{ij}^{\pi}(v)$ is the one-step transition probability prior to the time-driven transition function Θ_u and $|\Phi_j|$ the number of time-critical places of s_j that should be, but is not, marked (i.e., missing deadlines) at w_{u+1}.

3.4 Functional Equations and Optimal Policy

Under the assumption that each CN has perfect observation of all s_i's, one can construct and solve, backward recursively, the following functional equations for an optimal policy π^*:

$$U_V(s_i) = 0, \tag{4}$$

$$U_v(s_i) = \min_{\pi \in \Pi} \{c^{\pi}(v, s_i) + \sum_j P_{ij}^{\pi}(v) U_{v+1}(s_j)\}, \ 0 \leq v < V, \tag{5}$$

where V is the last epoch of I and $U_v(s_i)$ the *cost-to-go* [2] or the total expected number of tasks missing deadlines from epoch v to V under an optimal scheduling policy. Note that the minimum expected number J^* of tasks missing deadlines achieved under an optimal policy π^* is computed as $J^* = U_0(s_o)$, where s_o is the unique starting state of the CTMC model of the task system.

Consider again the task system of Fig. 2. Fig. 3 shows the decisions of N_2 under the optimal policy π^*, which is derived by dividing the planning cycle $I = [0, 10)$ into $V = 1,000$ intervals each of length $h = 0.01$. For example, as the system is in state $s_3 = (2, 4, 6, 20, 22, 27) \in S_1$ (Appendix A) at time $t \in I_1 = [0, 5)$, the optimal decision for N_2 (Fig. 3(a)) is to execute a_7 if $t \leq 0.89$, and to execute a_2 (i.e., respond to T_1's query) if $t > 0.89$. Suppose the system is in s_3 at $t < 0.89$, when N_1 executes default activity $R_1 = a_{14}$, N_2 executes a_7 and N_3 sends a message to N_2. Assume further that a_{14} is finished at the next epoch $t + h$ and thus brings the system to state $s_6 = (6, 9, 20, 22, 27)$. Then, as shown in Fig. 3(a), the optimal policy for N_2 is still to execute a_7. However, if the message from N_3 arrives at N_2 before completing a_7 or a_{14} (i.e., bringing the system to $s_{11} = (2, 4, 6, 20, 22, 28)$), then N_2 should instead reply to N_3 (i.e., execute a_8) immediately. The optimal policy in any other state and at any other epoch can be obtained similarly from Fig. 3. Also, in s_4, s_9, s_{12} and s_{18} within I_2, a_5 is always executed before a_7 (Fig. 3(b)). This is obvious, since at these states, the token is in ϕ_{22}, rather than in ϕ_{24}, meaning that T_2 is stuck at ϕ_{22} waiting hopelessly for a message that will never arrive. Hence, it is useless for N_2 to execute a_7 since T_2 will not be completed in time anyway.

In Fig. 3, several anomalies on the optimal decisions occur near the last epoch of $[0, 5)$ and $[5, 10)$. For example, when the system is in s_6 at epoch

499, the optimal decision of N_2 as shown in Fig. 3(a) is a_2 rather than a_7. This is because, from our approximation of continuous time with discrete epochs, a_2 is a decision just as good as a_7 at the last epoch of [0, 5). As V gets larger and larger, the derived π^* will become closer and closer to the true optimum.

The optimal cost for this example is $J^* = 1.3045$, which is rather high. Further experiments show that this is because of the absence of default activities for N_2 and the tightness of task deadlines. For example, J^* immediately drops to as small as 0.00056 when each task deadline is extended to 10 times of the original deadline. Also, from Eqs. (4) and (5), the computational complexity of the above solution method is $O(V\,|S_u|\,|D|)$, where V is the total number of epochs, $|S_u|$ the average number of states in a typical period I_u, and $|D|$ the average number of available scheduling decisions in each state.

4 Optimal Decentralized TMSP

Since the up-to-date global information is not generally available to each CN, it is important to derive an optimal *decentralized* policy such that the expected number of tasks missing deadlines is again minimized. Under this policy, each CN makes its own part of the scheduling decision using information available to the CN only. The information available to a CN consists of the current execution stage of tasks on itself (i.e., local state) and possibly out-dated information on the other CNs. This out-dated information is the local state which is broadcast periodically from each of the other CNs.

Given a state broadcast frequency, system performance depends heavily on how well each CN uses the broadcast information together with its own local state to make scheduling decisions. This problem belongs to the class of *dynamic team decision problems* [4], and is very difficult to solve except those with *one-step delay sharing* (1-SDS) information pattern [1, 5, 16]. In our decentralized TMSP, a variation of 1-SDS problems, the following dynamic information is available to each CN at time t: its own local state at time t, and the local states of all other CNs at time $t-1$.

Ideally, the more frequently does each CN collect the other CNs' local state information, the better scheduling decisions it will make. However, more frequent state broadcasts induce higher overheads to normal inter-task communications, and thus degrade the overall system performance. We shall determine the optimal frequency of broadcasts after the optimal decentralized policy for each CN has been derived.

In the following subsections, we first compute the delays both in state broadcasts and in normal inter-task communications due to the introduction of state broadcasts. Then, as in the centralized TMSP, we identify the important characterizing elements of the MDP such that the decentralized TMSP can again be solved with the DP technique. Unlike its centralized counterpart however, the elements for the decentralized TMSP include: the *probability state* of the DP network, *set of admissible action rules, probability state and its one-step transition probability, one-step cost,* and the *functional equations.*

4.1 Delays in State Broadcasts and Inter-Task Communications

Using the CTMC model and the periodic state broadcasts, the (communication) subsystem can be approximated by two single-server queues in parallel: an M/M/1 queue for inter-task communications and a D/D/1 queue for state broadcasts. Message communication delays thus depend on what portion of the subsystem's capacity is allocated to each of these two queues.

Let λ_x and b_x, respectively, be the known *arrival rate* and average service need (e.g., in number of bits) of inter-task messages. Let λ_y and b_y be the given number of synchronous state broadcasts per unit time and the service need for each broadcast, respectively (to be elaborated further below). Then, the portion of the subsystem's capacity allocated to serving inter-task messages can be approximated by $\xi_x = (\lambda_x b_x)/(\lambda_x b_x + \lambda_y b_y)$ and that for state broadcasts becomes $\xi_y = 1 - \xi_x = (\lambda_y b_y)/(\lambda_x b_x + \lambda_y b_y)$.

Recall that, in the CTMC model without state broadcasts, the delay of an inter-task message j was represented by an exponentially distributed random variable with rate μ_j. So, the average *sojourn* (system) time $\bar{S}$ of messages in the subsystem can be approximated by $\bar{S} = \frac{1}{n}\sum_j 1/\mu_j$, where n is the total number of inter-task messages within I. It follows from [6] that the *service rate* σ_x of the original M/M/1 queue is $\sigma_x = \lambda_x + 1/\bar{S}$. Hence, the "adjusted" service rate σ'_x of inter-task messages with broadcast messages considered becomes $\sigma'_x = \xi_x \sigma_x = \xi_x(\lambda_x + 1/\bar{S})$, and the average sojourn time $\bar{S}'$ of the new M/M/1 queue for the inter-task messages turns out

$$\bar{S}' = \frac{1}{\sigma'_x - \lambda_x} = \frac{1}{\xi_x(\lambda_x + 1/\bar{S}) - \lambda_x} = \frac{\bar{S}}{\xi_x - \xi_y \lambda_x \bar{S}}. \qquad (6)$$

Notice that $0 < \xi_x - \xi_y \lambda_x \bar{S} \leq 1$; otherwise, if $\xi_x - \xi_y \lambda_x \bar{S} = 0$ then the total traffic density of the M/M/1 queue will saturate the subsystem. Given $\bar{S}'$ as

above, the "adjusted" transition rate μ'_j of inter-task messages j then becomes

$$0 < \mu'_j = \mu_j \frac{\bar{S}}{\bar{S}'} = \mu_j \left\{ \xi_x - \xi_y \lambda_x \bar{S} \right\} \leq \mu_j \tag{7}$$

as a result of introducing state broadcasts.

Since the subsystem's capacity is in fact allocated indistinguishably among all the messages, the average service time W_x (W_y) of each inter-task message (state broadcast) at the M/M/1 (D/D/1) queue must be $W_x = \alpha b_x/\xi_x$ ($W_y = \alpha b_y/\xi_y$), where α is a fixed constant. It follows that $W_x/W_y = (b_x/\xi_x)(\xi_y/b_y)$ and thus the relation $\lambda_x W_x = \lambda_y W_y$ holds. Hence, the state broadcast delay (sojourn time or service time in this case) W_y can be expressed [6] as:

$$W_y = \frac{\lambda_x}{\lambda_y} W_x = \frac{\lambda_x}{\lambda_y} \frac{1}{\sigma'_x} = \frac{\lambda_x}{\lambda_y} \frac{1}{\xi_x(\lambda_x + 1/\bar{S})}. \tag{8}$$

In the above derivation, while each inter-task message is treated as a single arrival at the M/M/1 queue, all the synchronous state broadcasts at a given epoch are treated as a single arrival at the D/D/1 queue. This is not unreasonable because state information is assumed to be simultaneously broadcast at a given epoch by all CNs. Also, the inter-broadcast interval $1/\lambda_y$ must always be larger than W_y to prevent the broadcast messages from saturating the subsystem. Therefore, in the following discussions, we assume that each CN broadcasts state information only after the previous broadcast has been received by all other CNs. (This can be accomplished, for example, with the method in [3].) In terms of the theory of stochastic control, this assumption allows the separation of estimation from control [16], which is essential for the applicability of the DP technique.

In what follows, we assume that each CN broadcasts its state for B times within $I = [0, L)$ with inter-broadcast interval $g = L/B \geq W_y$. That is, the first broadcast is made at $t = 0$, the last at $t = (B-1)g$, and all messages broadcast at t will be received at $t + W_y \leq t + g$.

4.2 Probability States of the DP Network

Since the current global state is unavailable, we need to use other information as the "state" in our DP network. For each epoch v in $I_u = [w_u, w_{u+1})$ define a *probability state* as the vector $\rho \triangleq (p_1, p_2, \cdots, p_{|S_u|})$, where p_i is the marginal probability that the system is in s_i at epoch v, and S_u the global state space in I_u. Obviously, there are infinite number of probability states at each epoch v. However, since (i) both $|S_u|$ and the number of available scheduling options

in each state are finite and (ii) W_y time unit old global state is available to all CNs once every g time units, only a finite subset of probability states will be generated. These probability states serve as the set of "nodes" in the DP network on which the DP technique is applied for an optimal solution.

4.3 Set of Admissible Action Rules

An *action rule* is a rule of action for a *policy* at an epoch, and a policy is the collection of action rules at all epochs. The set of action rules that N_k can use within I_u is the Cartesian product of the sets of scheduling options available to N_k when N_k is in each of its local states within I_u. For example, if the total number of local states that N_2 can be in within I_1 is, say, 3 and N_2 has 2 scheduling options for each of these 3 states, then the total number of action rules that N_2 can possibly take within I_1 is $2^3 = 8$. One of such action rules would be: "If N_2 is in the first, second and third local states, then it executes activities a_2, a_7 and a_8, respectively." The entire set Γ_u of action rules for all CNs within I_u is the Cartesian product of the sets of action rules for all m CNs.

Since some components of a probability state ρ in I_u may be zero, only a subset of all action rules are *admissible*. To determine this set of admissible action rules for a given ρ, we first identify the set of all local states of N_k within I_u, each of which has a non-zero probability according to ρ. Second, identify the set of action rules for each of such local states for N_k. Third, construct the set of admissible action rules for each N_k at ρ. Finally, the set of all admissible action rules $\Gamma_u(\rho)$ is then constructed as the Cartesian product of the sets of admissible action rules over all CNs. Obviously, only $\Gamma_u(\rho)$ needs to be considered in solving the TMSP at ρ.

4.4 Probability State and One-Step Transition Probability

For the centralized TMSP, the one-step transition probability is, as described before, one essential element in its DP formulation. In the decentralized case however, except at the epochs where broadcasts are received, it is the probability state, rather than the probability of jumping into a state, that is essential to the DP formulation. This is because the system always jumps from one probability state to another (of the next epoch) with probability 1. Suppose the system is in probability state ρ at epoch $v \in I_u$. The probability states

generated at epoch $v+1$ from ρ at v and their one-step transition probabilities are determined depending on whether or not epoch $v+1$ is a broadcast receipt epoch.

A: $v+1$ is not a message receipt epoch:

From the definition of an admissible action rule $\gamma \in \Gamma_u(\rho)$, each γ determines a unique scheduling decision for each CN and each *global* state $s_i \in S_u$ with $p_i \neq 0$. Given that the system is in s_i at epoch v, one can use Eq. (1) or (2) to determine the probability $P_{ij}^{\gamma}(v)$ that the system will move to s_j at epoch $v+1$ if γ is used. Given the marginal probability p_i of s_i at epoch v, one can then compute the marginal probability p'_j of s_j at epoch $v+1$ as: $p'_j = \sum_{i=1}^{|S_u|} p_i P_{ij}^{\gamma}(v)$. Therefore, the unique probability state $\rho' \triangleq (p'_1, p'_2, ..., p'_{|S_u|})$ generated at epoch $v+1$ from ρ at epoch v is determined by:

$$\rho' = \rho \mathbf{P}^{\gamma}(v), \tag{9}$$

where $\mathbf{P}^{\gamma}(v)$ is the one-step *transition matrix* whose elements are the one-step transition probabilities $P_{ij}^{\gamma}(v)$'s determined in Eq. (1) or (2). For notational simplicity, we write $\rho' = \mathbf{P}^{\gamma}(\rho)$ to denote that ρ' is the unique probability state from ρ under γ.

Since one ρ' is generated from ρ for each $\gamma \in \Gamma_u(\rho)$, a set of $|\Gamma_u(\rho)|$ branches of *rhps* is formed in the DP network at epoch $v+1$ from ρ. Looking one epoch backward, the ρ itself is nothing but one branch of another set generated from a certain probability state at epoch $v-1$, and so on (Fig. 4). Given a generic ρ at epoch v between two consecutive state receipt epochs, this relationship continues to hold through a particular probability state, written as $X(\rho)$, at the epoch when states were broadcast, and finally rooted at a unique probability state at an epoch when the last state broadcasts were received. It is important to point out that each element of $X(\rho)$ represents the *prior probability* of a global state at the state broadcast epoch, and the particular global state realized at that epoch cannot be known until W_y time units later. $X(\rho)$ plays an important role in determining the one-step transition probability as will be described in the following case.

B: $v+1$ is a message receipt epoch:

Since W_y time units old global state information is now available to each CN at epoch $v+1$, the probability state at epoch $v+1$ is not generated from ρ at epoch v by Eq. (9). Rather, from the idea of the 1-SDS information

pattern, a total of $|S_u|\,|\Gamma_u|^s$ probability states can be generated directly at each of such epochs in I_u, where Γ_u is the set of all action rules in I_u and $s = W_y/h > 1$. Specifically, each of these probability states corresponds to: (i) the actual global state the system was in W_y time units ago, and (ii) the specific sequence of action rules adopted from the state broadcast epoch to epoch $v+1$. In other words, each of these probability states can be identified as an $(s+1)$-tuple $(s_i, \gamma'(0), \gamma'(1), \cdots, \gamma'(s-1))$, where s_i is the actual global state the system was in W_y time units ago, and $\gamma'(e)$, $e = 0, 1, \cdots, s-1$, represents the action rule taken at the e-th epoch since the last state broadcast.

Similarly, one may correspond each probability state ρ at epoch v, when the state broadcasts are not available yet, to an s-tuple $(X(\rho), \gamma(0), \gamma(1), \cdots, \gamma(s-2))$, where $X(\rho)$ is the root of ρ at the state broadcast epoch, and $\gamma(e)$ the action rule taken at the e-th epoch since the state broadcast. Notice that while $\gamma(e)$'s represent the action rules which "brought" the system to probability state ρ at epoch v, $\gamma'(e)$'s are those brought the system to ρ' at epoch $v+1$. In what follows, the one-step transition probability from ρ to ρ' is determined, depending on whether or not $\gamma(e) = \gamma'(e)$, $\forall\, e = 0, 1, 2, \cdots, s-2$.

Given ρ and ρ' at epochs v and $v+1$, respectively, denote as $\gamma(s-1)$ the action rule adopted at epoch v. Then, the one-step transition probability from ρ to ρ' can be determined easily as:

$$Q_{\rho\rho'}^{\gamma(s-1)} = \begin{cases} q_i & \text{if } \gamma(e) = \gamma'(e),\ \forall\, e = 0, 1, \cdots, s-1, \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

where q_i is the i-th component of $X(\rho)$ representing the marginal probability of the system being in s_i, which is the first component in the $(s+1)$-tuple representation of ρ'.

4.5 One-Step Cost

The one-step cost $d^\gamma(v, \rho)$ at epoch v for probability state ρ under action rule γ can be easily determined using the one-step cost $c^\pi(v, s_i)$ obtained from Eq. (3) for the centralized TMSP. Specifically, we have

$$d^\gamma(v, \rho) = \sum_{i=1}^{|S_u|} p_i c^\pi(v, s_i), \tag{11}$$

where π is the centralized policy corresponding to the decentralized action rule γ when the system is in s_i, and p_i the i-th component of ρ.

4.6 Functional Equations and Optimal Policy

Unlike the centralized case for which only one pass is needed, three passes are required to derive the optimal decentralized policy. The first pass is to generate all the probability states of the DP network. The second pass solves the functional equations, backward recursively, to find the optimal action rule at *each* probability state ρ and each epoch v. Since ρ is not observable, a third pass is needed to identify the optimal action rule for each epoch. In what follows, the last two passes are described; the first pass has already been presented in Section 4.4.

A. Functional Equations (the second pass)

Similarly to the centralized case, the functional equations $U_v(\rho)$ is the cost-to-go representing the number of task missing deadlines as the system is in ρ at epoch v. $U_v(\rho)$ can be determined easily by using backward recursion as follows.

$$U_V(\rho) = 0. \tag{12}$$

If $v+1$ is not a message receipt epoch, then

$$U_v(\rho) = \min_{\gamma \in \Gamma_v(\rho)} (d^{\gamma}(v, \rho) + U_{v+1}(\rho')). \tag{13}$$

On the other hand, if $v+1$ is a message receipt epoch then

$$U_v(\rho) = \min_{\gamma(s-1) \in \Gamma_v(\rho)} \left\{ d^{\gamma(s-1)}(v, \rho) + \sum_{\rho'} Q_{\rho\rho'}^{\gamma(s-1)} U_{v+1}(\rho') \right\}, \tag{14}$$

where $Q_{\rho\rho'}^{\gamma(s-1)}$ is the one-step transition probability from ρ to ρ' derived in Eq. (10). Notice that the minimum expected number, J^*, of tasks missing deadlines, which is achieved under the optimal decentralized policy γ^*, is equal to $U_0(\rho_0)$, where ρ_0 is the probability state representation of the unique starting state s_o of the task system.

B. Optimal Policy (the third pass)

Let $\bar{\gamma}_v(\rho)$ be the optimal action rule for ρ at epoch v derived in the second pass above. Then, the optimal decentralized action rule γ_v^* we are interested in can be identified as the $\bar{\gamma}_v(\rho)$ of one particular ρ to be explained below. Notice that the probability state ρ at epoch v is not observable (and thus not appears as part of the notation of γ_v^*) and is uniquely determined by the action rule used and, for broadcast receipt epochs, the out-dated state

broadcast information as well. This means that the nodes (probability states) of the DP network visited by the optimal action rule γ_v^* form a path of V nodes each at a separate epoch from 0 to $V-1$. Therefore, the optimal action rule γ_v^* at epoch v can be identified following the nodes ρ_v^* to be visited on the path as follows. We consider each of the three intervals: (1) from epoch 0 to the first message receipt epoch r_1, (2) from the i-th to the $(i+1)$-th message receipt epoch, $1 \leq i \leq B-1$, and (3) from the B-th (last) message receipt epoch r_B to the end of the planning cycle. Consider first the interval between epoch 0 and r_1. Obviously, ρ_0^* can be initialized as

$$\rho_0^* = \rho_0. \tag{15}$$

Then, forward recursively,

$$\gamma_v^* = \bar{\gamma}_v(\rho_v^*), \quad 0 \leq v < r_1\,, \tag{16}$$

and

$$\rho_{v+1}^* = \mathrm{P}^{\bar{\gamma}_v}(\rho_v^*), \quad 0 \leq v < r_1\,, \tag{17}$$

where $\mathrm{P}^{\gamma}(\rho)$ denote the unique probability state from ρ under γ as described in Eq. (9).

Next, consider the interval from r_i to r_{i+1}, $1 \leq i \leq B-1$. From the results of the second pass and using the global state contained in the received broadcasts, a unique probability state ρ_{r_i} at epoch r_i is determined. Similar to Eqs. (15)-(17), we have

$$\rho_{r_i}^* = \rho_{v_i}, \tag{18}$$

$$\gamma_v^* = \bar{\gamma}_v(\rho_v^*), \quad r_i \leq v < r_{i+1}\,, \tag{19}$$

and

$$\rho_{v+1}^* = \mathrm{P}^{\bar{\gamma}_v}(\rho_v^*), \quad r_i \leq v < r_{i+1}, \tag{20}$$

At epoch r_{i+1}, when state broadcasts are received again and used, a unique probability state $\rho_{r_{i+1}}$ can be identified by using the results of the second pass as well as the information contained in the broadcasts received. This process repeats itself for all such intervals between epochs r_i and r_{i+1} until the final message receipt epoch r_B.

Finally, within the interval from epochs r_B to the last epoch V, one can derive:

$$\rho_{r_B}^* = \rho_{r_B}, \tag{21}$$

$$\gamma_v^* = \bar{\gamma}_v(\rho_v^*), \quad r_B \leq v < V, \tag{22}$$

and

$$\rho^*_{v+1} = \mathbf{P}^{\gamma^*_v}(\rho^*_v), \quad r_B \leq v < V. \tag{23}$$

Obviously, $\gamma^* \triangleq [\gamma^*_0, \gamma^*_1, \cdots, \gamma^*_v, \cdots, \gamma^*_{V-1}]$ is the optimal policy derived for the decentralized TMSP with periodic state broadcasts.

In summary, three passes are needed to solve the decentralized TMSP:

P1. Generate the DP network (forward recursively) as described in Sections 4.2-4.4.

P2. Solve the functional equations, Eqs. (12)-(14), backward recursively using the one-step cost derived in Eq. (11) for the DP network generated.

P3. Identify the optimal decentralized scheduling rules forward recursively from the solutions obtained in P2 using Eqs. (15)-(23).

As an example, consider again the task system in Fig. 2. Within $I = [0, 10)$, suppose inter-task messages a_1, a_3, a_4, a_6, a_{12} and a_{13} each occurs only once, while a_{10} and a_{11} each occurs twice, resulting in a total of 10 messages. Recall that $\mu_1 = \mu_3 = \mu_4 = \mu_6 = 2$ and $\mu_{10} = \mu_{11} = \mu_{12} = \mu_{13} = 1$. The average sojourn time $\bar{S}$ of these inter-task messages within the communication subsystem is $\bar{S} = \frac{1}{10}\sum_i \frac{1}{\mu_i} = 4/5$. Since $\lambda_x = 10/L = 1$, the service rate of the M/M/1 queue $\mu_x = \lambda_x + 1/\bar{S} = 9/4$. Let the arrival rate of state broadcasts $\lambda_y = 2$ and the service need $b_y = 0.2\,b_x$, where b_x is the service need for each inter-task message. Then, the portion of capacity allocated to the inter-task messages is $\xi_x = b_x/(b_x + 0.4b_x) = 5/7$ and the adjusted service rate $\mu'_x = \xi_x\mu_x = (5/7)(9/4) = 45/28$. From Eq. (6), $\bar{S}' = 1/(\mu'_x - \lambda_x) = 28/17 > 4/5 = \bar{S}$, and $\bar{S}/\bar{S}' = 17/35$. From Eq. (8), the delay in broadcasting states becomes $W_y = \lambda_x W_x/\lambda_y = (1/2)(28/45) = 14/45$ since $W_x = 1/\mu'_x = 28/45$. Notice that the inter-broadcast interval (0.5) is larger than W_y satisfying the requirement that states are broadcast only after the previously broadcast states have been received. Also, to avoid saturating the communication subsystem, the maximum of λ_y with $b_y = 0.2\,b_x$ occurs only when $\bar{S}' = \infty$, i.e., $\mu'_x = \xi_x\mu_x = \lambda_x$ or $\xi_x = \xi_y\lambda_x\bar{S}$ (Eqs. (6) and (7)). This occurs at $\lambda_y = 25/4$ when $\xi_x = 4/9$, where $W_x = W_y = \infty$.

To ease the computational difficulty imposed by the DP algorithm, the following approximations are made: (i) the planning cycle is discretized into 200 intervals, (ii) W_y is discretized into only two stages, each of which contains several intervals, and (iii) the marginal probability of each state is discretized into 10 different intervals. Applying this approximation to the decentralized

TMSP with $b_y = 0.2\,b_x$ and $\lambda_y = 0.4$, 0.8, 1.0, 2.0, 3.0 and 4.0, we obtained $J^* = 2.309$, 1.577, 1.445, 1.446, 1.489 and 1.543, respectively, showing that $\lambda_y = 1$ is the best among the five broadcast frequencies. To show the fact that the optimal frequency depends on b_y, the same algorithm is applied again to the cases with $b_y = 0.5\,b_x$ and $\lambda_y = 0.2$, 0.4, 0.8, 1.0 and 2.0. The best broadcast frequency again turned out to be $\lambda_y = 1.0$, but with the corresponding $J^* = 1.544 > 1.445$. However, as shown in Fig. 5, the true optimal broadcast frequency of the former should be greater than that of the latter case. These results are not surprising since the communication subsystem now needs to allocate more capacity to deliver the same state information, and thus, degrades the normal inter-task communications.

4.7 Computational Cost of the Solution Algorithm

Similar to any DP-based algorithm, the computational complexity of the proposed solution algorithm comes mainly from the total number of probability states in the underlying DP network. Let $|P|$ be the number of intervals into which the probability spectrum [0, 1] is discretized and remember that $|S_u|$ is the average number of global system states in each interval I_u. Then, the total number of probability states generated will be $\Delta = V|P|^{|S_u|}$, where V is the number of epochs in a planning cycle. Remember that the average number of action rules available to each system state is $|D|$, the computational complexities of the first (generating the DP network) and the second (solving the functional equations) passes are both $O(\Delta|D|) = O(V|P|^{|S_u|}|D|)$, whereas that of the third pass (searching for the optimal policy) is $O(V)$. This makes the computational complexity of the overall algorithm $O(V|P|^{|S_u|}|D|)$. It is not surprising that because of the difficulty of the decentralized TMSP, the resulting solution is not a polynomial algorithm. (In fact, all DP-based algorithms have exponential complexity.) Notice that the computational complexity of our solution to the centralized TMSP is $O(V|S_u||D|)$ as pointed out before, which is much simpler than that of its decentralized counterpart.

5 Conclusions

Scheduling combined tasks and messages is important in distributed real-time systems since time-critical tasks must be completed before their deadlines to prevent possibly catastrophic consequences. In this paper, we have presented both centralized and decentralized algorithms for the problem of optimally

scheduling periodic tasks and their inter-task communication messages to minimize the number of tasks missing deadlines.

Concurrent execution of tasks and inter-task messages communications are first modeled as a sequence of continuous-time Markov chains based on which the DP technique is applied to derive optimal scheduling policies. For the centralized case, the optimal policy is computed by assuming the up-to-date global system state is available to each computing node (CN). For the decentralized case, however, we assume that all CNs periodically broadcast their local states so that other CNs can make better scheduling decisions. The optimal decentralized scheduling policy and its optimal state broadcast frequency are derived by using the separation principle, i.e., separating state-estimation from decision-making.

Both optimal centralized and decentralized policies are derived *off-line*, and the results can be looked up when the system is in operation. Thus, the computation complexities with deriving such policies can be tolerated if the problem size is not too large. However, if the number of global states and the size of the policy space are large, then a simple but good approximation to this technique is important. This is a matter of our future inquiry.

References

[1] M. Aicardi, F. Davoli, and R. Minciardi, "Decentralized Optimal Control of Markov Chains with a Common Past Information Set," *IEEE Trans. on Automat. Contr.*, Vol. AC-32, No. 11, November 1987, pp. 1028-1031.

[2] E. Denardo, "Dynamic Programming: Models and Applications," Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[3] A. Griefer and R. Strong, "DCF: Distributed Communication With Fault Tolerance," *Proc. 7th Annual ACM Symp. on Principles of Distributed Computing,* Aug. 1988, pp. 18-27.

[4] Y. C. Ho and K. C. Chu, "Team Decision Theory and Information Structures in Optimal Control Problems - Part 1," *IEEE Trans. on Automatic Control*, Vol. AC-17, no. 1, Feb. 1972, pp. 15-22.

[5] K. Hsu and S. Marcus, "Decentralized Control of Finite State Markov Processes," *IEEE Trans. on Automatic Control,* Vol. AC-27, no. 2, April 1982, pp. 426-431.

[6] L. Kleinrock, *Queueing Systems, Volume I : Theory,* Wiley & Sons, 1975.

[7] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. of ACM,* Vol 20, No. 1, 1973, pp. 46-61.

[8] J. P. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," *Proc. of IEEE Real-Time Systems Symposium,* 1990, pp. 201-209.

[9] Jane W. S. Liu *et al.*, "Algorithms for Scheduling Imprecise Computations," *IEEE Computer,* May 1991, pp. 58-68.

[10] M. A. Marsan, G. Balbo and G. Conte, "A Class of Generalized Stochastic Petri Nets for the Performance Analysis of Multiprocessor Systems," *ACM Trans. on Computing Systems,* Vol. 2, No. 2, May 1984, pp. 93-122.

[11] D. T. Peng, and K. G. Shin, "Modeling of Concurrent Task Execution in a Distributed Real-Time Computer Systems," *IEEE Trans. on Computers,* Vol. C-36, No. 4, Apr. 1987, pp. 500-516.

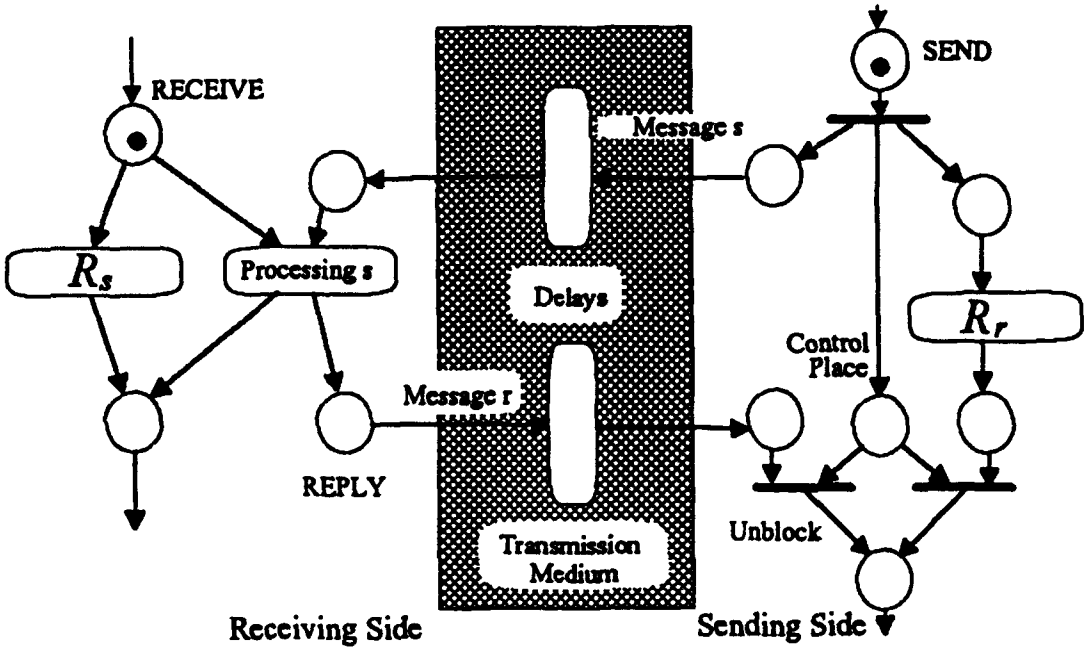
[12] M. Pinedo and L. Schrage, "Stochastic Shop Scheduling: A Survey," in *Deterministic and Stochastic Scheduling,* Dempster, *et al.* (eds), Reidel, Dordrecht, The Netherlands, 1981, pp. 181-196.

[13] S. M. Ross, *Applied Probability Models with Optimization Applications,* Holden-Day, 1970.

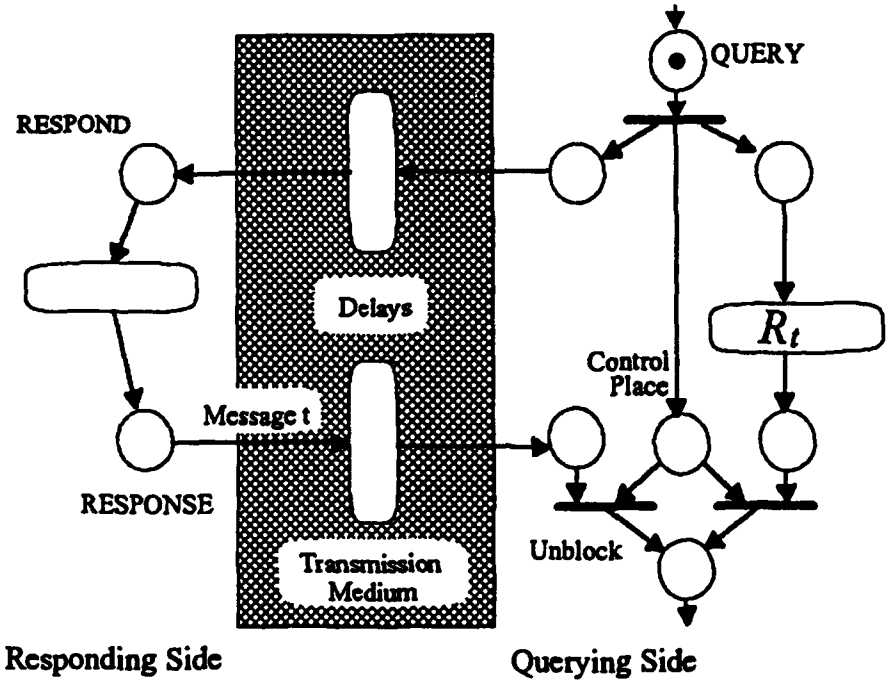
[14] K. G. Shin and M. E. Epstein, "Intertask Communications in an Integrated Multi-Robot System," *IEEE J. of Robotics and Automation,* Vol. RA-3, no. 2, Apr. 1987, pp. 90-100.

[15] Andre M. van Tilborg and G. M. Koob (eds), *Foundations of Real-Time Computing: Scheduling and Resource Management,* Kluwer Academic Publishers, 1991.

[16] H. S. Witsenhausen, "Separation of Estimation and Control for Discrete Time Systems," *Proc. of the IEEE,* Vol. 59, No. 11, 1971, pp. 1557-1566.



(a). SEND-RECEIVE-REPLY



(b). QUERY-RESPONSE

Figure 1: GSPN Model of Default Activity

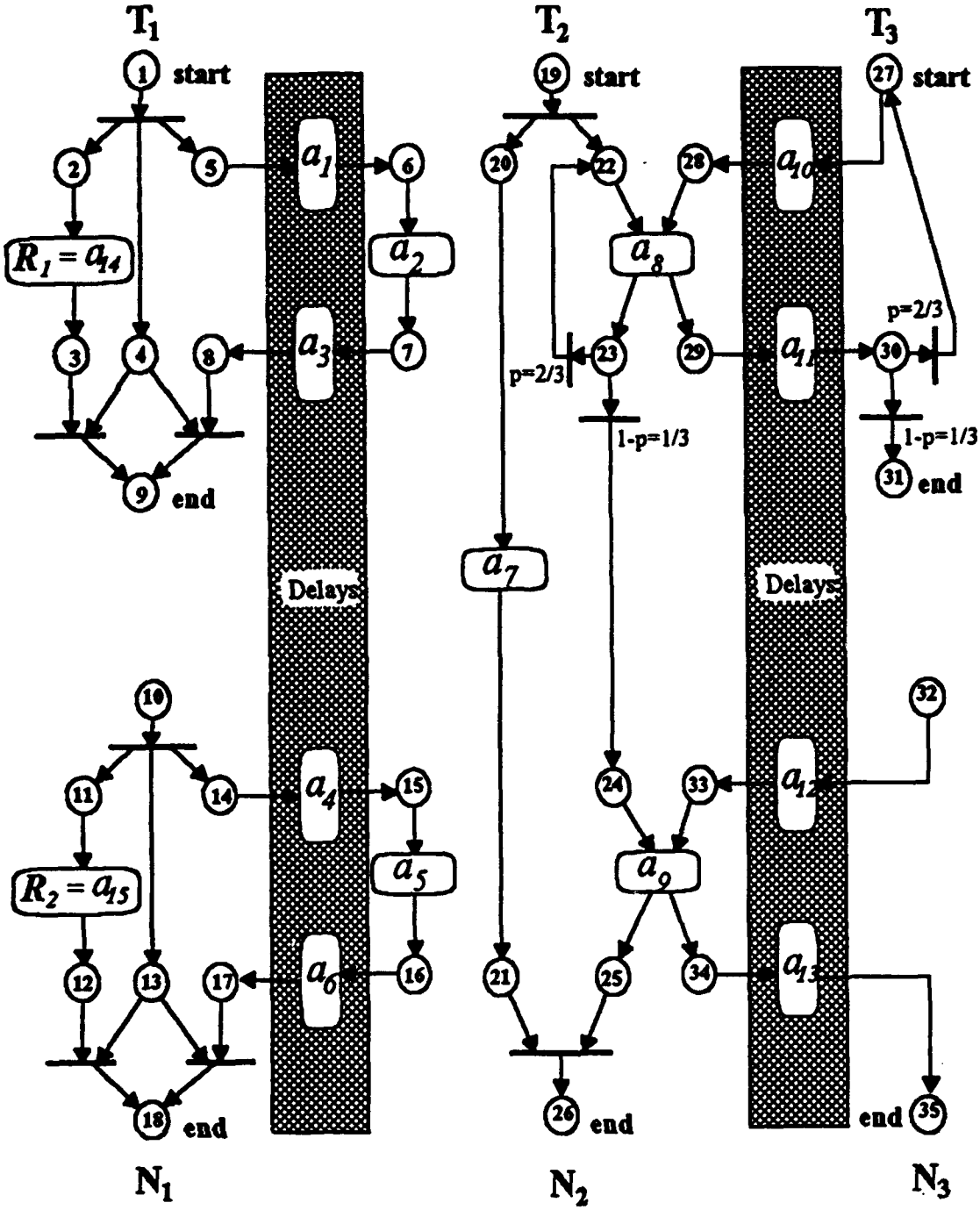
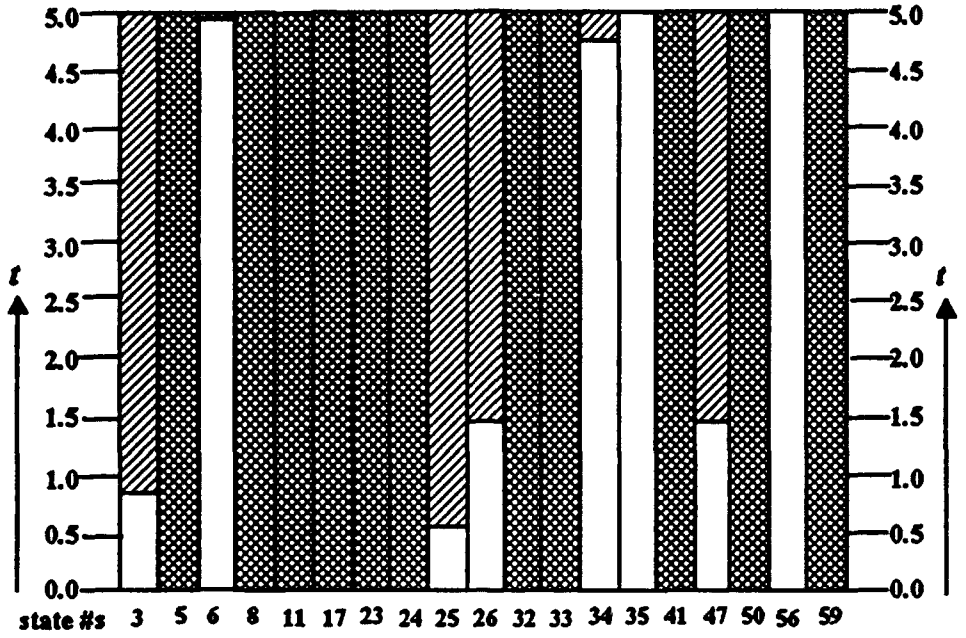
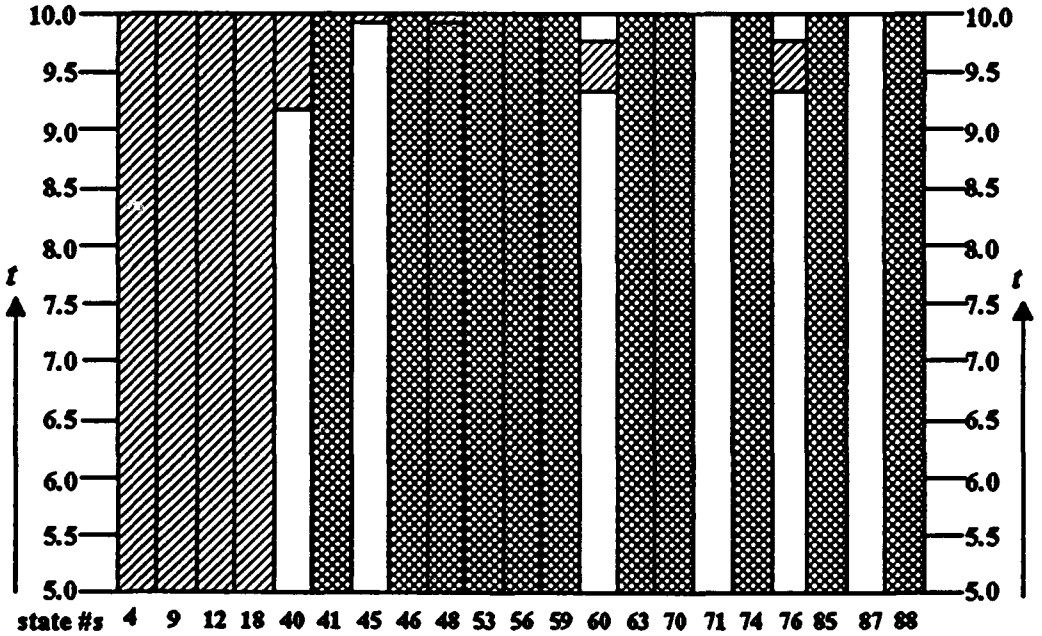


Figure 2: A Simple Task System



Legend: ▨ : a_2 □ : a_7 ▩ : a_8

(a). $I_1 = [0, 5)$



Legend: ▨ : a_5 □ : a_7 ▩ : a_9

(b). $I_2 = [5, 10)$

Figure3: Optimal Scheduling Policy for N_2

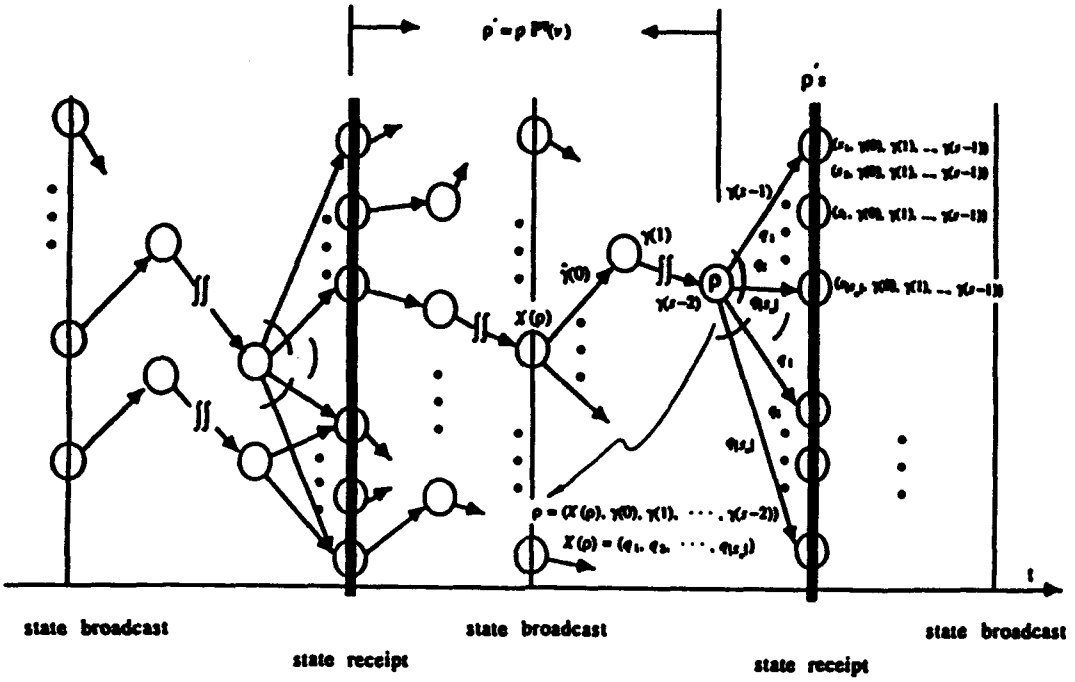


Figure 4: Probability State and One-Step Transition Probability

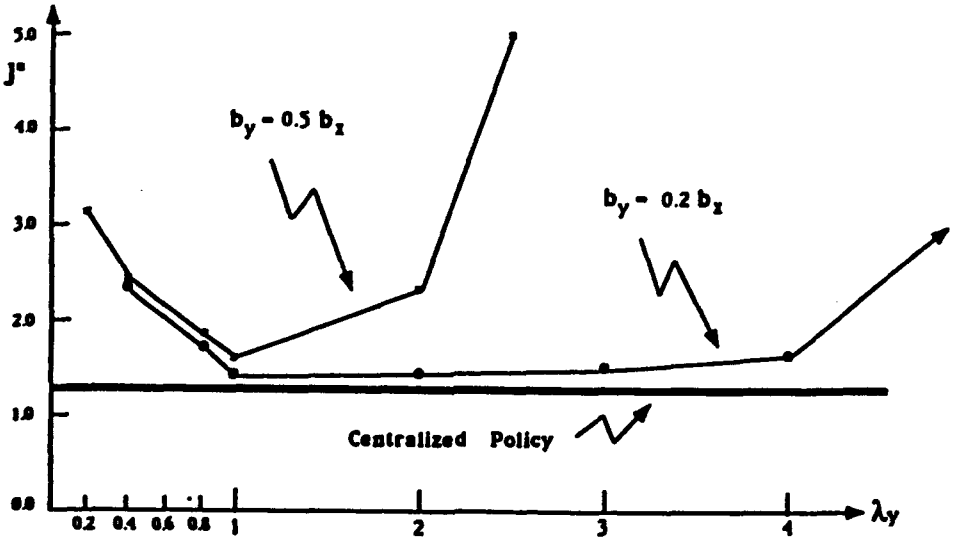


Figure 5: J*'s for $b_y = 0.2\ b_x$ and $b_y = 0.5\ b_x$

APPENDIX A

SYSTEM STATES OF FIGURE 2

A state s_i is represented as $i = \{j \mid \text{place } \phi_j \text{ has a token}\}$.

(a). $I_1 = [0, 5)$

3 = 2 4 6 20 22 27
5 = 2 4 5 20 22 28
6 = 6 9 20 22 27
8 = 5 9 20 22 28
11 = 2 4 6 20 22 28
17 = 6 9 20 22 28
23 = 2 4 7 20 22 28
24 = 2 4 6 21 22 28
25 = 2 4 6 20 22 29
26 = 2 4 6 20 24 29
32 = 7 9 20 22 28
33 = 6 9 21 22 28
34 = 6 9 20 22 29
35 = 6 9 20 24 29
41 = 2 9 20 22 28
47 = 2 4 6 20 24 31
50 = 8 9 20 22 28
56 = 6 9 20 24 31
59 = 3 9 20 22 28

(b). $I_2 = [5, 10)$

4 = 11 13 15 20 22 32
9 = 15 18 20 22 32
12 = 11 13 15 20 22 33
18 = 15 18 20 22 33
40 = 11 13 15 20 24 32
41 = 11 13 14 20 24 33
45 = 15 18 20 24 32
46 = 14 18 20 24 33
48 = 11 13 15 20 24 33
53 = 11 13 15 21 24 33
56 = 15 18 20 24 33
59 = 11 13 16 20 24 33
60 = 11 13 15 20 25 34
63 = 15 18 21 24 33
70 = 16 18 20 24 33
71 = 15 18 20 25 34
74 = 11 18 20 24 33
76 = 11 13 15 20 25 35
85 = 17 18 20 24 33
87 = 15 18 20 25 35
88 = 12 18 20 24 33

REQUIREMENTS

Comparing Formal Approaches for Specifying and Verifying Real-Time Systems

C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw*

1 Introduction

Engineering embedded systems, such as the mission-critical computer (MCC) systems developed for the Navy, has become difficult due to the complexities of stringent requirements for hard-real-time performance, dependability, security, etc. Of particular importance is the development of unambiguous, complete, and consistent requirements specifications for such systems. Experience has shown that system errors found late in the development process often are the result of poorly understood requirements specifications [19]. Furthermore, such system errors are best detected as early as possible to avoid much more expensive modifications later in development [3].

One approach to this problem is the use of formal methods in specifying requirements and in verifying critical properties of those requirements. Formal methods, based upon precise mathematical theories and models, have the potential for improving the correctness of requirements specifications, especially those for the most critical aspects of the system such as security and real-time performance. It has been suggested (see, e.g., [20]) that these formal methods need to be tested on actual real-time systems. Such testing will allow the scalability of the methods to be assessed and will also uncover new problems requiring formal solution. A recent survey reports that formal methods have been used successfully in the development of actual systems, but that much needs to be done in such areas as integrating formal methods with informal methods in the development process and in introducing better formal models for hard real-time requirements [10].

The goal of our research is to understand better the hard real-time aspects of requirements. Recently, a large number of formal methods have been invented for specifying and verifying real-time systems. However, a greater understanding is needed of how they compare—e.g., what classes of problems they can solve, the availability and quality of mechanical support, etc. To provide insight into the utility of different methods for solving real-time problems, we have developed a generic version of a real-time railroad crossing system. We are using this example as a benchmark for comparing different formal approaches for specifying real-time systems and for analyzing their properties. In this paper, we provide a formal statement of the problem; describe three formal approaches that can be applied, namely, process algebra, model checking, and general-purpose theorem provers; and summarize efforts currently in progress to use each approach to specify the system of interest and prove properties about its behavior. In Sections 4 and 5, we describe initial results obtained with an approach based on the process algebra CSP and an approach using the general-purpose theorem proving system PVS, respectively.

2 Generic Railroad Crossing (GRC) Problem

2.1 Background

The original example, developed by Leveson to illustrate her software safety techniques [18] and extended to a real-time version to illustrate Modechart [16], involves a system operating a gate

*C. Heitmeyer, R. Jeffords, and B. Labaw are with the Naval Research Laboratory, Washington, DC 20375. The work reported here is supported in part by ONT grant N6092193WRW0066.

2 Generic Railroad Crossing (GRC) Problem

2.1 Background

The original example, developed by Leveson to illustrate her software safety techniques [18] and extended to a real-time version to illustrate Modechart [16], involves a system operating a gate at a railroad crossing. The system must ensure that the system cannot enter an unsafe state. In particular, it must satisfy a *safety property*: i.e., whenever a train is in the crossing, the crossing gate must be down.

To make the problem somewhat more realistic, we have generalized it. While the previous versions describe a system with a single track and at most two trains in the region of interest both traveling in the same direction, our version allows several tracks and an unspecified number of trains traveling in both directions. In addition to the safety property, our version includes a *utility property* to ensure that automobiles have fair access to the crossing. The purpose of the utility property is to make sure the system performs its function and to avoid degenerate solutions, e.g., a solution that lowers the gate and keeps it lowered. Safety-critical systems must not only operate safely. To be useful, they must also perform certain functions within specified time intervals. We note that the utility property requires bounded liveness, which turns out to be a safety property.

2.2 GRC Problem Statement

The system to be developed operates a gate at a railroad crossing. The railroad crossing I lies in a region of interest R, i.e., $I \subset R$. A set of trains travel through R on multiple tracks in both directions. A sensor system determines when each train enters and exits region R. To describe the system formally, we define a gate function $g(t) \in [0, 90]$, where $g(t) = 0°$ means the gate is down and $g(t) = 90°$ means the gate is up. We also define a set $\{\lambda_i\}$ of *occupancy intervals*, where each occupancy interval is a time interval during which one or more trains are in I. The ith occupancy interval is represented as $\lambda_i = [\tau_i, \nu_i]$, where τ_i is the time of the ith entry of a train into the crossing when no other train is in the crossing and ν_i is the first time since τ_i that no train is in the crossing. Figure 1 shows two examples of occupancy intervals.

Given two constants ξ_1 and ξ_2, $\xi_1 > 0$, $\xi_2 > 0$, the problem is to develop a system to operate the crossing gate that satisfies the following two properties:

Safety Property:	$t \in \cup_i \lambda_i \Rightarrow g(t) = 0$	The gate is down in all occupancy intervals
Utility Property:	$t \notin \cup_i [\tau_i - \xi_1, \nu_i + \xi_2] \Rightarrow g(t) = 90$	The gate is up as often as possible

Figures 2a and 2b illustrate the Safety and Utility Properties.

3 Formal Approaches

Several formalisms are available to specify the system described above and to reason about its properties. These formalisms fall into three classes:

- **General-Purpose Theorem Provers** (e.g., Boyer-Moore [4], EVES [9] [17], EHDM [23], PVS [21] [22] [26] [27], HOL [13]),
- **Model Checkers** (e.g., Clarke's CTL [5], the Modechart verifier [28]), and
- **Process Algebras** (e.g., CSR [12], Cleaveland's Concurrency Workbench [7], and CSP [11]).

We note that verification tools based on the two latter approaches, model checking and process algebras, are highly specialized and provide verification with little human intervention. In contrast,

a proof generated with a mechanical theorem prover usually requires considerable human guidance. Efforts are currently in progress to apply one or more examples of each approach to the GRC problem. To develop insight into the styles of specification and verification that are most natural and effective for a given approach, these efforts are, to the extent feasible, proceeding independently.

Our initial evaluation focused on the FDR (Failures Divergence Refinement) tool [11] for automatically checking CSP specifications. Bill Roscoe of Oxford developed the original CSP specifications of the GRC problem which NRL has modified and extended. The analysis in Section 4 is based on the FDR version developed by NRL.

A second evaluation was based on a solution developed using the theorem-proving system PVS by Natarajan Shankar of SRI International [25]. That solution used a real-time model based upon Unity [6] with real-time constraints expressed using a past time operator Since. At NRL we experimented with the same basic model, but used a complementary Till operator that expressed future time. The analysis in Section 5 covers both of these efforts.

We evaluated the suitability of the CSP and PVS solutions using criteria defined in reference [8]. These criteria include conciseness, expressibility, ease of use, and scalability. In evaluating expressibility, we compared the ease of expressing both the system specifications and the properties of interest.

4 CSP Solution

4.1 Overview

Verification in FDR means checking that one CSP process *refines* another CSP process. In the GRC example, it must be shown that the CSP process that models the system behavior (which we treat as the specification) refines a more abstract CSP process that encodes a system property, such as the Safety Property or the Utility Property.

The current version of FDR does not have an intrinsic model of time. To model time, we interleave clock-pulse events among the other system events. For this model of time, it is essential to verify the following two properties before addressing the Safety and Utility Properties:

1. **No-Deadlocks Property.** The specification is deadlock-free.
2. **Non-Zeno Property.** The specification does not exhibit Zeno behavior: between any two clock-pulses, there is never an infinite number of other events.

In our verification, we defined the conjunction of properties 1 and 2 as a single property, called the TimeOK Property.

4.2 Evaluation

Conciseness

FDR supports only restricted parameterization of CSP processes and numeric expressions. Parameterization of high level processes formed by parallel composition of other processes is not allowed. Similarly numeric functions with arguments are not allowed in parameterized expressions.

In our experiments we found both these limitations too restrictive, and used the UNIX m4 macro tool as a preprocessor to generate CSP specifications. Full parameterization of the processes and identifiers would have been more convenient if included directly in FDR.

Expressibility for the System Specification

CSP/FDR is particularly effective for modeling the control flow in concurrent finite state systems that communicate via mutual synchronization. It is also convenient for specifying nondeterminism. However, the following limitations of (untimed) CSP and the FDR tool hamper expressibility:

- Because timing is an add-on feature, modeling delays and deadlines is awkward.
- The omission of strong typing in CSP/FDR means that some common errors, such as undefined constants, go undetected.
- No graphical representations of CSP processes are available. Some users may prefer graphical representation to aid in comprehension of CSP processes.

An intrinsic weakness of this approach, shared with many model checkers, is that a concrete finite model of the specifications is required. (This can also be viewed as an advantage, since concrete models permit decision procedures that lead to completely automatic verification.) This manifests itself in a number of ways:

- Modeling data retained in the state of a process is weak—it is addressed only to a limited extent by channels and parameterized processes.
- Modeling is limited to finite state systems.
- General specifications of relationships between arbitrary variables cannot be nandled.

Ease of Use

The basic use of FDR to check traces refinement is not difficult to learn. Traces refinement, a familiar approach used in other systems, e.g., the Constrained Expression Toolset [2], is closely related to regular language recognition. The verification process is straightforward and completely automatic.

That the FDR tool is a prototype is evident. The tool operates on textual specifications in a largely batch-oriented style (e.g., a large chunk of specification is parsed at one time; error reporting is minimal and often cryptic). Moreover, there is little tool support for creating specifications. Although an excellent user manual and tutorial [11] are available, detailed instructions for using the tool are generally lacking.

Scalability

Our more ambitious experiments with FDR quickly led to unacceptable response times. The major underlying cause was exponential increase in the size of the CSP processes as system parameters grew larger. The basic limitation to concrete models and the associated exhaustive search strategy inherent in refinement checking is the essential cause of this exponential increase.

The compositionality of CSP refinement provides an approach to scalability problems. If the most abstract version of a system refines (satisfies) a property, and its components are *refined* (in the dual sense that we refine them to a more concrete form and the refinement relation holds) independently, then the most concrete version also satisfies the property.

This form of composition is straightforward if we are working from an abstract form to increasing levels of detail. Our case was more difficult since we were working backward and wished to find a more abstract form. The natural candidate for the more abstract form in GRC was a specification that collapsed the multitude of trains into a single Abstract Train. With this approach we were able to reduce the time to verify the TimeOK property from 18 minutes elapsed time to 2 minutes elapsed time.

Expressibility and Validation of Properties

Validating that the formal expression of the properties in FDR is equivalent to those of the GRC problem statement is difficult, even though both forms are formal. The major difficulty is that the language for expressing the properties that FDR requires is the same CSP language used to express the specifications rather than a more abstract, declarative language such as the CSP traces language. Use of this language makes it awkward to express properties that refer to state: e.g., the Safety property can be paraphrased as "In-State-Crossing implies In-State-Down."

The ease of validation was improved by developing two separate independent versions of both the Safety Property and the Utility Property. Our attempt to prove the equivalence of the different versions led us to discover some errors. The final result was two sets of CSP specifications of these two properties. By way of example, two versions of the Utility Property were formulated, one deterministic, the other nondeterministic and easier to understand due to separation of concerns.

5 PVS Solution

5.1 Overview

Specification and verification using a general-purpose theorem prover such as PVS requires both the encoding of a real-time specification/verification method (RSVM) as well as the specification and verification of the system in question. Optionally, one also has complete freedom to modify or create anew some RSVM. Both the RSVM and system specification must be encoded as axioms, definitions, proof strategies, etc. in the logic language supported by the theorem prover.

The RSVM developed by Shankar and encoded in PVS [25] consists of a state-transition model similar to Unity [6] with a non-decreasing real-valued time associated with each new system state. Timing constraints are added via a Since operator, which provides the time since a condition defined on the state variables was last true with respect to the current system state. This single operator is sufficient to express both deadline and delay constraints upon a system. This method also separates the concurrent behavior of the system from the timing behavior.

A complementary Till operator, which indicates the nearest future state when a condition becomes true, was developed in our experiments at NRL. The Till operator was added to express timing constraints in a more natural way than could be expressed by exclusive use of the Since operator.

Conciseness

The language of PVS is Higher-Order Logic (HOL). In general, HOL provides a concise notation for expressing system specifications as well as the encoding of the RSVM. In many situations HOL allows more concise specification of constructs than First Order Logic (FOL) since it allows parameters to be functions–this is not allowed in FOL.

Expressibility for the System Specification

On the other hand the use of HOL as a general-purpose specification language may not adequately address concerns about domain-specific special notations to include graphics. The notations and concepts of HOL such as quantifiers, lambda expressions, etc. are not likely part of the vocabulary of the practicing engineer. It would be useful to have a front-end tool, fully integrable with PVS, for experimenting with such domain-specific notations. PVS does anticipate part of this need by providing LaTeX output, but this is limited to formatting standard mathematical notations.

Appropriate encoding of the RSVM into a general-purpose theorem prover can support the development of quite general models. The model of the GRC in PVS allows an arbitrary (even infinite) number of trains. Furthermore it supports timing parameters as variables and relationships among them rather than specific values; e.g., to ensure safety of the system, the sum of deadlines on moving the gate down and any signaling or computation overhead must be less than the minimal time it requires some train to reach the crossing after entering the region R. General specifications promote ease of change, reuse, and better understanding of boundary conditions (e.g., the relationship between gate and train crossing time mentioned previously).

In the development of an RSVM, particular care must be taken that there is a balance between expressibility concerns and others such as verification ease. In the case of FDR, the ease of verification via an automatic decision procedure was the overriding decision in limiting CSP processes to finite concrete models. For the RSVM developed by Shankar, there may have been too much emphasis upon ease of verification via the exclusive use of the Since operator for timing constraints. The timing

specifications using the Since operator are less understandable than the delay/deadline paradigm of Modechart [16] since the former looks backward in time while the latter looks forward.

To investigate if the delay/deadline paradigm could naturally be encoded in a PVS specification (as well as to gain experience with PVS by actual proving new theorems rather than simply examining proofs) we added the Till operator to the RSVM. We showed that it is feasible to use the Till operator to make specification of timing more natural without excessive complication in verification for the Safety property.

Ease of Use

The current user interface to PVS is provided by emacs. This type of interface, although better than a simple "dumb terminal" interface as in the case of FDR, seems somewhat dated in this age of bit-mapped graphics. The user documentation assumes familiarity with emacs. For the emacs novice it would be preferable to include the minimal subset of emacs required for running PVS as part of the documentation (the standard emacs online tutorial is insufficient).

More importantly, the use of a general-purpose theorem prover requires both general theorem proving skills (i.e., the mathematical maturity and expertise to develop manual informal proofs) as well as expertise in the use of the tool since there is considerable interaction between PVS and the user. Furthermore the level of effort required to formally prove a theorem with assistance from PVS is at least an order of magnitude greater than that required to do an informal proof. For critical applications, this effort may be worthwhile. Errors may be detected that might not otherwise be found in the "social process" of peer review of informal manual proofs. Moreover, formal proofs may lead to better understanding of all assumptions required for a complete proof [24].

Thus general-purpose theorem provers such as PVS appear to be more appropriate for experimental use in developing verification paradigms (such as RSVM's) in special domains rather than for production use by a practicing engineer.

That PVS implements a strongly-typed version of HOL is quite useful even if full proofs are not attempted. The preliminary type-checking phase of PVS can eliminate common specification errors related to type incompatibility.

Scalability

In many ways, general models scale up better than concrete models. For example, repetitive similar constructs (such as the individual trains of the GRC) are not much more difficult to specify or verify than a single construct in PVS. By comparison, extending the number of trains in the concrete model of FDR results in exponential explosion.

On the other hand, the difficulty of informal theorem proving in general, as exacerbated by the level of detail required for formal, machine-assisted proof, makes scaling up very difficult. Two approaches to simplifying proofs may be of benefit:

- Development of theory of composition of proofs that allows proofs to be developed for components, and the components combined without having to reprove results for the overall system. The compositionality of FDR is an example of this approach. A general approach to composition is given in [1]. SRI is currently investigating composition for PVS.

- Development of powerful lemmas and proof strategies that hide much of the detail of formal proofs. Well-known decision procedures, such as those for Presburger arithmetic and numerical inequalities, are provided in PVS. This eliminates much of the drudgery associated with low -evel details of formal proofs. Additional proof strategies for the RSVM need to be investigated (our experimentation at NRL did not address this aspect).

Expressibility and Validation of Properties

HOL is sufficiently expressive to provide high level expression of the properties (Safety and Utility) in a form that may easily be validated with the formal statement in the GRC. The validation approach

used for FDR could also be applied: two or more statements of a property could be specified and then proved to be equivalent, although this was not part of our experiments.

6 Summary

Our initial experiments with CSP/FDR and PVS have given us considerable insight into the utility of the process algebra and general purpose theorem prover approaches for specifying and verifying real-time properties. Analyzing the different solutions to the GRC problem should help identify the strengths of each approach and how each approach can be used productively to develop industrial-strength real-time systems.

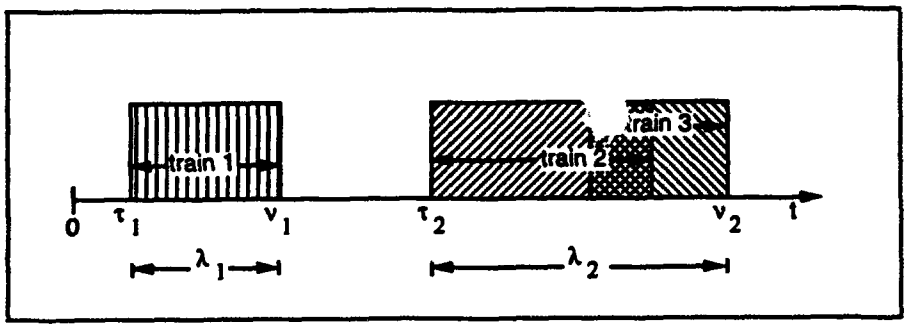


Figure 1. Two examples of occupancy intervals

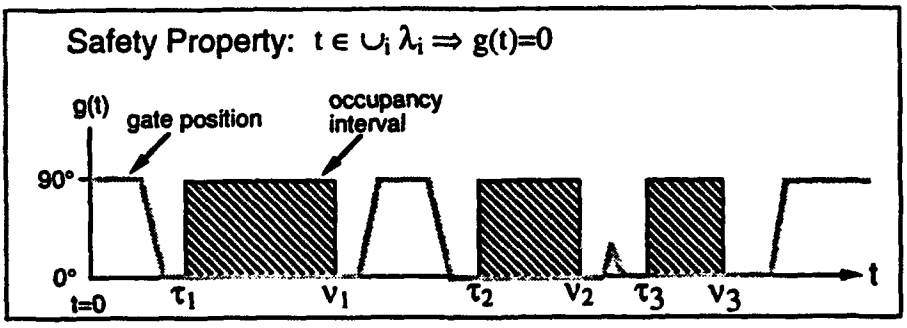


Figure 2a. Safety Property specifies relation of gate position to occupancy intervals

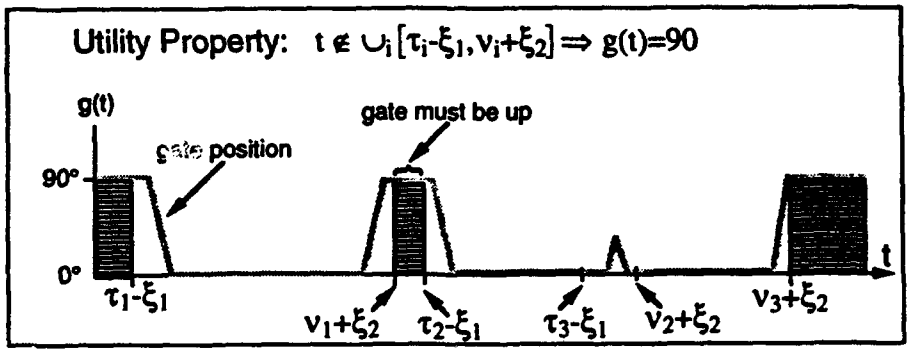


Figure 2b. Utility property specifies relation of gate position to occupancy intervals and constants ξ_1 and ξ_2

References

[1] M. Abadi and L. Lamport, "Composing Specifications," *ACM Trans. Prog. Lang. and Sys.*, Vol. 15, No. 1, Jan. 1993, pp. 73-132.

[2] G. S. Avrunin, U. A. Buy, J. C. Corbett, and L. K. Dillon, "Automated Analysis of Concurrent Systems with the Constrained Expression Toolset," *IEEE Trans. Softw. Eng.*, Vol. SE-12, No. 11, Nov. 1991, pp. 1204-1222.

[3] B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[4] R. S. Boyer and J. S. Moore, *A Computational Logic Handbook*, Boston, MA, 1988.

[5] E. M. Clarke, E. Emerson, A. Sistla, "Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications," *ACM Trans. Prog. Lang. and Sys.*, Vol. 8, No. 2, Apr. 1986.

[6] K. M. Chandy and J. Misra, *Parallel Program Design: A Foundation*, Addison-Wesley, Reading, MA, 1988.

[7] R. Cleaveland, J. Parrow, and B. Steffen, "The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems," *ACM Trans. Prog. Lang. and Sys.*, Vol. 15, No. 1, Jan. 1993, pp. 36-72.

[8] P. C. Clements, C. E. Gasarch, and R. D. Jeffords, "Evaluation Criteria for Real-Time Specification Languages," NRL Memo. Report 6935, Naval Research Lab., Wash, DC, 1992.

[9] D. Craigen, "Reference Manual for the Language Verdi," TR-91-5429-09a, Odyssey Research Associates, Ottawa, Ont., Canada, Sep. 1991.

[10] D. Craigen, S. Gerhart, and T. Ralston, "An International Survey of Industrial Applications of Formal Methods," NRL Report, Naval Research Lab., Washington, DC (in press).

[11] "Failure Divergence Refinement, User Manual and Tutorial," Version 1.2, Formal Systems (Europe) Ltd., Oxford, UK, 3 Dec. 1992.

[12] R. Gerber and I. Lee, "Communicating Shared Resources: A Model for Distributed Real-Time Systems," *Proc. Real-Time Systems Symposium*, Santa Monica, CA, Dec. 1989, pp. 68-78.

[13] M. Gordon, "Mechanizing Programming Logics in Higher Order Logic," Tech. Report 145, University of Cambridge, Cambridge, UK, Sept. 1988.

[14] C. L. Heitmeyer, P. C. Clements, B. G. Labaw, A. K. Mok, "Engineering CASE Tools to Support Formal Methods for Real-Time Software Development," *Proc. CASE '92 Fifth Intl. Workshop on Computer-Aided Softw. Eng.*, Montreal, Canada, 6-10 July 1992.

[15] A. Hall, "Seven Myths of Formal Methods," *IEEE Software*, Vol. 7, No. 5, Sep. 1990, pp. 11-19.

[16] F. Jahanian and D. A. Stuart, "A Method For Verifying Properties of Modechart Specifications," *Proc. Real-Time Systems Symposium*, Huntsville, AL, 6-8 Dec. 1988.

[17] S. Kromodimoeljo, W. Pase, M. Saaltink, D. Craigen, and I. Meisels, "A Tutorial on EVES," Odyssey Research Associates, Ottawa, Ont., Canada, 10 Feb. 1993.

[18] N. G. Leveson and J. L. Stolzy, "Analyzing Safety and Fault Tolerance Using Time Petri Nets," *TAPSOFT: Joint Conf. on Theory and Practice of Software Development*, Springer-Verlag, Mar. 1985.

[19] R. Lutz, "Analyzing Software Requirements Errors in Safety-Critical Embedded Systems," TR 92-27, Iowa St. Univ., Ames, Iowa, Aug. 1992.

[20] J. S. Ostroff, "Survey of Formal Methods for the Specification and Design of Real-Time Systems," *Tutorial on Specification of Time*, 1991 (to appear).

[21] S. Owre, N. Shankar, and J. M. Rushby, "The PVS Specification Language (Draft)," Computer Science Lab., SRI Intl., Menlo Park, CA, Feb. 1993.

[22] S. Owre, N. Shankar, and J. M. Rushby, "User Guide for the PVS Specification and Verification System (Draft)," Computer Science Lab., SRI Intl., Menlo Park, CA, Feb. 1993.

[23] J. Rushby, F. von Henke, and S. Owre, "An Introduction to Formal Specification and Verification Using EHDM," Tech. Report SRI-CSL-91-2, SRI Intl., Menlo Park, CA, 1991.

[24] J. M. Rushby and F. von Henke, "Formal Verification of Algorithms for Critical Systems," *IEEE Trans. Softw. Engin.*, Vol. 19, No. 1, Jan. 1993, pp. 13-23.

[25] N. Shankar, "Mechanized Verification of Real-Time Systems Using PVS," SRI Intl., Menlo Park, CA, 1993 (to appear).

[26] N. Shankar, S. Owre, and J. M. Rushby, "PVS Tutorial," Computer Science Lab., SRI Intl., Menlo Park, CA, Feb. 1993.

[27] Shankar, S. Owre, and J. M. Rushby, "The PVS Proof Checker: A Reference Manual (Draft)," Computer Science Lab., SRI Intl., Menlo Park, CA, Feb. 1993.

[28] D. A. Stuart, "Implementing a Verifier for Real-Time Systems," *Proc. Real-Time Systems Symposium*, Orlando, FL, Dec. 1990, pp. 62-71.

Advanced Integrated Requirements Engineering System (AIRES):

Processing of Natural Language Requirements Statements

1993 Complex Systems Engineering Synthesis and Assessment Technology Workshop
(CSESAW '93)

July 20-22, 1993

Washington D. C.

James D. Palmer and Richard Evans

The Center for Software Systems Engineering

School of Information Technology and Engineering

George Mason University

Fairfax, VA 22030-4444

AIRES: An Advanced Integrated Requirements Engineering System

AIRES is an advanced integrated requirements engineering system which has been developed to assist users and designers in the preparation of correct requirements that accurately reflect user needs. As such, AIRES considers natural language prose statements as the most natural media format for user expression of system and software requirements information. AIRES supports the management of the requirements engineering life cycle, as shown in Figure 1, through modules and CASE tools directly aimed at support and automation of the specific processes related to elicitation, organization, assessment, prototyping, and transformation.

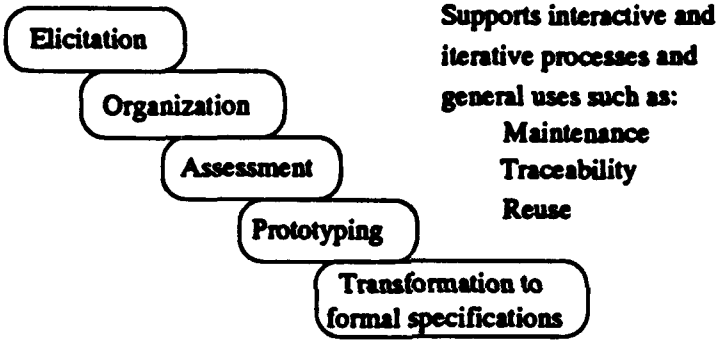


Figure 1 Requirements Engineering Lifecycle

AIRES provides assistance in the elicitation and capture of requirements information from users through an innovative conceptual approach of multi-group decision support systems whose purpose is to elicit and capture information in a consensus building environment. The AIRES assessment modules take this information and existing specifications from legacy systems, conditions these statements and automates error discovery. Error correction is accomplished by the user/designer working in consort. The AIRES prototyping module may work on requirements information directly from the elicitation process and legacy systems or from the outputs of the assessment modules. Once requirements sets are identified as prototype candidates, AIRES, through its systems architecture, provides access to standard tools for construction prototypes. The final module of AIRES automates transformation from natural language prose statements to the formal language of finite state machines through use of StateMate, a commercial CASE tool. From this point onward traditional software development processes take over with the user and designer assured of correct requirements specifications, that are as error free as possible, as the transformation to design commences.

Requirements Engineering Objectives and Needs

The ideal objectives in the management of the requirements engineering process are: to develop the characteristics of a new or modified system that are complete, consistent, correct, feasible, maintainable, precise, traceable, testable, unambiguous, understandable, validatable, and verifiable. While these characteristics may represent an ideal set of features, it is not possible to assure that one or any group of these will be met in a given set of requirements. To demonstrate that the foregoing is true, we need only examine one or two of these features. Clearly, it is not possible to determine that any set of requirements is complete, for not even the user knows when a system is complete. Understandability is another elusive feature in that we must ask to whom and for what purpose is the set of requirements to be understandable.

Clearly, the question to ask is: What is needed to manage the requirements engineering process? First, we must involve the user/designer from the onset of the process; minimize transformations that alter the original meaning of the statement; assess requirements statements prior to transformations so as to provide analysis of any errors introduced by the user rather than those introduced by the transformation interpretation process; provide for full traceabilility, both forward from the highest level and backward from the most detailed level; develop prototypes for purposes of risk management; and finally, assign metrics to requirements statements to provide for validation and verification during other phases of the development life cycle.

Two major goals for AIRES are to provide the user and designer automated support to enable them to find things that should be in a set of requirements and, conversely, to find things that should not be in a set of requirements before any irreparable damage is done to the requirements through transformations to formal specifications.

AIRES Architecture and Operation

AIRES represents a significant advance in the development of an enabling requirements engineering process coupled with automated approaches to obtain the right requirements from the user that may be made free from errors of certain types (ambiguity, conflict, redundancy, inconsistency, etc.) without compromise of user intent. As illustrated in Figure 2, AIRES is an integrated environment, supporting and spanning the requirements engineering life cycle.

SOFTWARE ENGINEERING, REQUIREMENTS ENGINEERING, AND AIRÈS

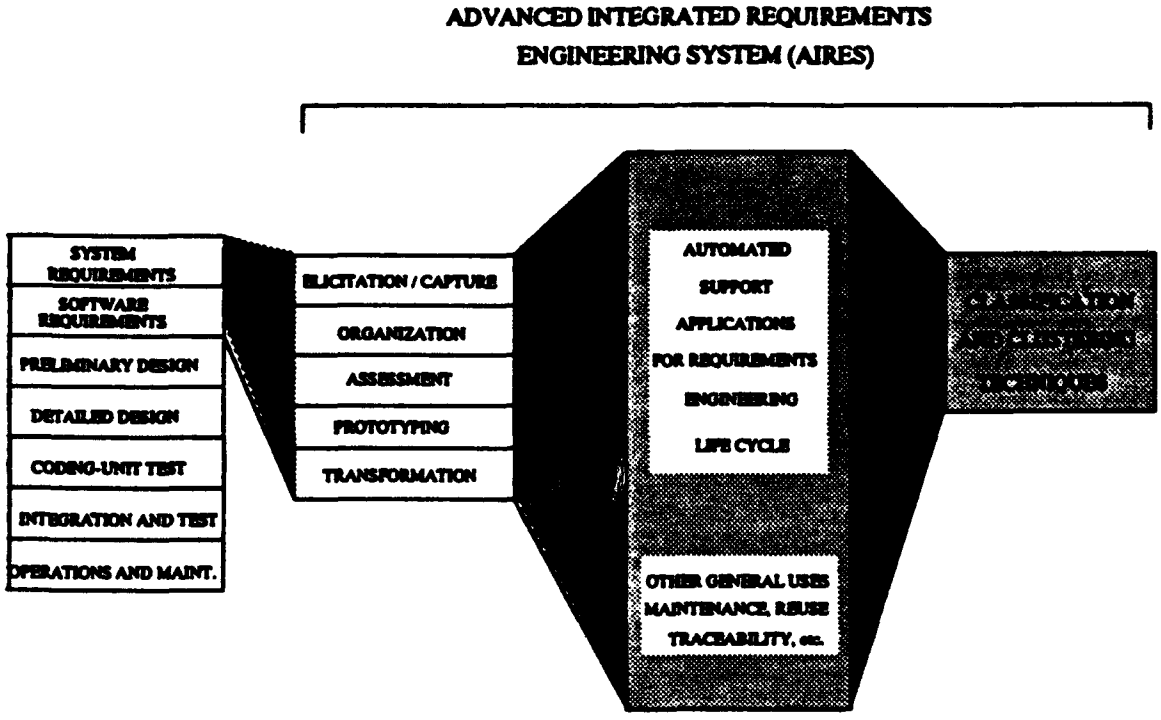


Figure 2 AIRES

With the AIRES approach, each activity in the requirements engineering life cycle is supported by a set of automated CASE tools, which taken together, comprise the suite of CASE tools in the AIRES environment. Elicitation is supported by a computer supported cooperative work environment that features a multi-group decision support system to enable groups of users and designers to work together even if spatially or temporally distributed. In this approach to elicitation, an evolutionary decision meeting model is formed in the context of overall software systems engineering that adopts the management technology of systems engineering and applies it to software to help ensure that correct software is designed, and not just that software designs are correct. The organize function is aimed at facilitating user/designer groups in organizing information such that related concepts are grouped together and high priority items are given prominence. In the instance of grouping related concepts, we provide for conceptual clustering algorithms for a variety of types of clustering. The role of priority setting falls upon the user/designer reaching consensus as to which ones are most important for the system under

consideration. In the next step of the requirements engineering process, we engage in assessment activities using classification and clustering of requirements sets. Classification is a process which adds knowledge about requirements, while clustering gives us the means to form meaningful groups of requirements. Prototyping enables the identification of requirements at risk, namely those that continue to undergo change (volatile sets); those that contain ambiguities; those that are inconsistent or in conflict; and those that require special handling to provide understanding such as human computer interfaces or database management systems. Finally, we look to the transformation of these sets of informal requirements statements to the formal language of a specification. For AIRES, we have also selected a transformation to finite state machine language such as that found in the commercial CASE tool "StateMate".

Syntax of Natural Language Requirements

The key technological aspects of AIRES are realized through the utilization of the syntax of natural language prose statements coupled with language semantics derived from domain specific knowledge bases; rule sets; a predefined requirements taxonomy; and weighted sets of thesaurus of verbs and nouns, synonyms and antonyms, and key phrases for identification of specific characteristics important for assessment, identification of high risk clusters and systems features. This technique supports evaluation of requirements clusters for prototyping; identification of errors such as conflict, inconsistency, and ambiguity; determination of coupling and cohesion within and across requirements and clusters of requirements; determination of the ripple effect impact of adding new requirements to existing systems (maintenance); storage and retrieval functions (reuse) of specific requirements (or clusters of requirements); generation of traceability matrices for complete auditing; and assessment of the degree of volatility in requirements generation. All of this is directly aimed at the management of the requirements engineering process and the management of risk in requirements engineering activities.

AIRES and Requirements Engineering Benefits

AIRES provides several requirements engineering benefits. These include the ability to facilitate the elicitation of requirements information through multi-group decision support system aids; to examine legacy system specifications for problems and errors; the ability to correctly and exhaustively identify requirements specifications subject to impact through the maintenance process of adding new functionality to these legacy systems; and the ability to support requirements risk assessment and risk management. Other facets in AIRES include the capability to archive and retrieve requirements specifications for reuse without the necessity of significant investment to condition these in advance for reuse; the ability to perform forward and

backward traceability; the coupling of requirements to sizing and estimating tools; and the capability to assign metrics to requirements specifications, in accord with the classification taxonomy utilized, to facilitate the design of validation tests upon delivery.

There are several outcomes associated with AIRES operation. The application of taxonomies and rule sets supports the determination of conflict within and across statements and clusters of statements; the identification of consistency within and across statements and clusters of statements; the definition of coupling and cohesion factors within and across pairs as well as clusters of requirements; and the determination of completeness across requirements and their clusters using domain knowledge. Additional outcomes in the application of taxonomies and rule sets include the identification of potential ambiguity, redundancy, and internal completeness; the definition of the ripple effect impact of adding new requirements to existing systems; and establishing the storage and retrieval functions for specific requirements. The application of classification and clustering techniques, in the context of predefined taxonomies and associated rule sets, also supports the definition of the degree of volatility in the requirements generation; the preparation of traceability matrices for complete auditing of all requirements throughout the life of the project (including maintenance); and requirements prototyping.

Bibliography

[AIKE89] Aiken, Peter H., *A Hypermedia Workstation for Requirements Engineering*, published Ph. D. Dissertation, George Mason University, School of Information Technology and Engineering, Fairfax, VA 1989.

[ARMO93] Armour, Frank, *A Risk Management Approach for Prototyping Systems Requirements* published Ph. D. Dissertation, George Mason University, School of Information Technology and Engineering, Fairfax, VA., 1993.

[BEAM88] Beam, Walter R., Palmer, James D., and Sage, Andrew P. *Systems Engineering for Software Productivity* IEEE Transactions on Systems, Man, and Cybernetics, Vol SMC-17, No. 2, March/April 1988.

[BROU92] Brouse, Peggy *A Process for use of Multi-media Information in Requirements Identification and Traceability*, published Ph. D. Dissertation, George Mason University, School of Information Technology and Engineering, Fairfax, VA 1992.

[EMME93] Emmert, Barbara *Multi-Group Decision Support Systems: Integration and Analysis of Requirements Information*, published Ph. D. Dissertation, George Mason University, School of Information Technology and Engineering, Fairfax, VA 1993.

[FIEL91] Fields, N. Ann *An Evolutionary Group Decision Model for Computer Supported Cooperative Work* published Ph. D. Dissertation, George Mason University, School of Information Technology and Engineering, Fairfax, VA 1991.

[LIAN91] Liang, Yiqing, 1991, *Software Requirements Classification*, published Ph. D. Dissertation, George Mason University, School of Information Technology and Engineering, Fairfax, VA 1991.

[MYER88] Myer, Margaret *A Knowledge-Based System for managing Software Requirements Volatility* published Ph. D. Dissertation, George Mason University, School of Information Technology and Engineering, Fairfax, VA 1988.

[PALM90] Palmer, J. D., Liang, Yiqing, and Wang, Lillian, Classification as an Approach to Requirements Analysis, *Proceedings of the 1st ASIS SIG/CR Classification Research Workshop*, Ed. Sussane M. Humphrey and Barbara H. Kwasnik, Toronto, Ontario, Canada, November 4, 1990.

[PALM92] Palmer, J. D., and Liang, Yiqing, *Indexing and clustering of software requirements specifications*, Information and Decision Technologies April 1992.

[PALM92a] Palmer, J. D., and Fields, N. Ann *An Integrated Environment for Software Requirements Engineering* IEEE Software May 1992.

[PALM92b] Palmer, J. D., Fields, N. Ann, and Emmert, Barbara, *Computer Supported Cooperative Work Environment for Multiple Spatially Distributed Groups: A Case Study* 1992 IEEE International Conference on Systems, Man, and Cybernetics, October 1992.

[PFLE89] Pfleeger, Shari L. *An Investigation of Cost and Productivity for Object-Oriented Development* published Ph. D. Dissertation, George Mason University, School of Information Technology and Engineering, Fairfax, VA 1989.

[SAMS89] Samson, D. E., 1989, *Automated Assistance for Software Requirements Definition*, published Ph. D. Dissertation, George Mason University, School of Information Technology and Engineering, Fairfax, VA, 1989.

REQUIREMENTS MANAGEMENT/REQUIREMENTS ENGINEERING (RM/RE)

LUKE CAMPBELL
NAVAL AIR WARFARE CENTER AIRCRAFT DIVISION
FLIGHT TEST AND ENGINEERING GROUP, SY30
PATUXENT RIVER, MD 20670
(301) 826-7601
FAX (301) 826-7607

Requirements Management / Requirements Engineering (RM/RE)

To control costs, better productivity, and decrease the resources required for development and support of products, NAVAIR-546 has developed a process which accounts for the structure of all developments: Requirements Management and Requirements Engineering (RM/RE).

This RM/RE process is used throughout the life cycle of the product, and is itself under constant improvement. RM/RE establishes, controls, tracks, and engineers requirements for avionics systems. A cornerstone to RM/RE is the use of new automated software tools which greatly enhance the human ability to manage, track, update and control requirements. These tools, together with new techniques and methodologies, are combined in a process to ensure requirements are properly elicited, defined, tracked, modeled, and tested throughout the life cycle.

RM/RE is the baseline process to which all other processes must attach; managing requirements allows program personnel to manage all other aspects of the system engineering program. The primary steps which we have defined for the RM/RE process are:

- Requirements Elicitation - the process of documenting the needs and resolving the disparities among the involved stakeholders for the purpose of defining and distilling requirements to meet the constraints of these stakeholders.
- Requirements Inspection/Validation - the process which involves checking the accuracy of the products of the preceding operations, in order to validate that the requirements derived are an accurate reflection of the user needs.
- Requirements Analysis - the process of studying and refining user needs in terms of system, hardware, or software requirements.

The RM/RE Process is a combination of technical engineering disciplines and an administrative process for establishing, validating and maintaining the many requirements of a system throughout its life cycle.

A requirement is a statement of need, a characteristic of a need, or a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document or documents. The process of defining requirements, within the context of RM/RE, encompasses the three interrelated operations mentioned earlier: elicitation, inspection/validation, and analysis.

Application of these three interrelated operations is referred to as RM/RE. The foundation of success for any project is dependent on the quality of its requirements and the management of these requirements. Identification and correction of incomplete or inconsistent requirements early in the system life cycle will help to alleviate costly and time-consuming modifications later, or prevent development of a system which does not perform as required. Studies have shown that requirements management errors contribute to the majority of system defects and are the most costly to remove.

A poorly-written attempt to define a requirement will not accurately describe its author's need, will not accurately translate this need, and will not be correctly interpreted by the developer of the system. The result of a poorly written requirement which is not rectified will be a system development defect.

A system development defect exists when the written specification does not describe the developed product. System development defects can be divided into four broad categories: requirements defects, design defects, code/build defects, and documentation defects. Each defect type produces downstream problems, i.e. a requirements defect results in design defect(s) which then cause code/build defect(s)

and documentation defect(s). When defects are identified early in the development cycle, fewer problems will be encountered as system development progresses.

Well-written requirements will be worded to make more than one interpretation unlikely, since requirements are read and interpreted within the same organization by persons of varied backgrounds. A well-written requirement will concisely describe a need. Accurate translation of this need by the system developer is essential. Requirements will be written with the imperative "shall." When statements written with "shall" are contained in contractual documents, they are legally binding.

Although human intervention is required during all phases of the development process, RM/RE should be accomplished using automated tools. It is not practical to manually decompose, analyze, and trace each requirement. A further advantage to automation is the resulting ease of change impact assessments. Wherever changes are made, an automated requirements database will provide the process for identifying all linked requirements which also are affected.

1.0 Requirements Elicitation

Requirements elicitation involves "capturing" thoughts and ideas from stakeholders and depositing these ideas in a central repository. Key to requirements elicitation include capturing rationale for each requirement, and reconciliation of requirements (using automated software tools) to assure that the requirements are ordered and connected.

To date, the process step of elicitation has been haphazard and lacking in engineering rigor. Generally, a specification is developed based on a previous program, and changes are added by various domain experts. There are attempts to control the integration of requirements, but these generally fall short of the need. The resulting product is often inconsistent, behind schedule, and ripe for problem reports or change requests.

A better method for elicitation involves using automated software tools and techniques, such as storyboarding and prototyping, in conjunction with a Requirements Elicitation Conference attended by all stakeholders. This elicitation method allows the capture of requirements and the associated rationale in a forum where all stakeholders are represented and a consensus is formed.

The elicitation method requires a more intense peak of activity for the stakeholders, which should be accomplished quickly (in several days) with the use of techniques such as storyboarding (which enhance the gathering of requirements), scribes (which assemble the requirements) and automated tools (which quickly sort and categorize the requirements). Within the requirements elicitation process are five subprocesses:

- Capture Requirements from Stakeholders
- Requirements Reconciliation (Creation)
- Initiate Requirements Traceability
- Stakeholders' Input Complete Determination
- Establish Database

1.1 Capture Requirements from Stakeholders

Upon initiation of a program, inputs are received from the stakeholders; these inputs include the system requirements and all applicable documents. The most advantageous manner in which to acquire the requirements is in a Requirements Elicitation Conference, where all stakeholders are gathered in one place to provide and reach a consensus on inputs, linkages, and rationale. New system requirements originate from deficiencies in existing systems' capabilities due to new or enhanced technologies, or due to newly identified threats. These deficiencies are examined and a Mission Needs Statement (MNS) is issued which defines the need for the system. The MNS is translated by OPNAV in to the

Operational Requirements Document (ORD). From the requirements of the ORD, a requirements database is established. After establishing the requirements database, and as system development progresses, there will be additional input to this process in the form of approved changes. These requirements changes will be captured in the database, keeping a current baseline for the requirements at all times. Automated requirements management tools are available for establishing, maintaining, and updating the database.

1.2 Requirements Reconciliation (Creation)

During the Requirements Reconciliation (Creation) process, all linkages between requirements should be defined and documented. This can be done using a requirements management tool. Requirements should flow down from the high-level requirements in the ORD to the next succeeding lower-level design specification. High-level requirements (parents) should be decomposed into lower-level requirements (children) in the same document or in successive multiple lower-level documents.

Proper RM/RE ensures complete decomposed flowdown, so that there are no barren requirements (high level requirements without children). Similarly, orphans (low-level requirements without parents) should not exist. In addition to requirements in the parent-child domain, peer relationships, in which the linked requirements populate the same level, likely will exist. Requirements which lack essential relationships should be reconciled. The use of techniques such as storyboarding, lexical analysis, and rapid prototyping should be employed to reconcile requirements which are barren or orphaned.

1.2.1 Storyboarding is a modeling technique that extends from requirements analysis and simulation methodology. A storyboard is a sequence of displays that represents the functions that the system may perform when formally implemented. The intent of storyboarding is to provide a means of better defining and validating system requirements. Storyboarding provides a means to:

- o more precisely establish what to build and how to specify it,
- o explore alternative designs and man/machine interfaces,
- o promote synergism among stakeholders.

The use of storyboarding as a technique to verify requirements definitions and to establish and maintain specifications provides several benefits. Characteristics of the system can be examined, user feedback can be obtained, alternatives can be examined and evaluated, and problems can be detected early and corrected.

1.2.2 Lexical Analysis is a methodology for examining the complexity, elements, and the relationship of words or vocabulary from its grammar and construction. The process includes analyzing the syntax of natural-language statements and then classifying the requirements according to similarities. Once the requirements are classified, they must be examined for imprecise and ambiguous words, conflicts among quality-metrics must be detected, and the identified problems must be presented to the stakeholders for clarification and resolution.

1.2.3 Rapid prototyping is a methodology which allows quick development of a preliminary design or model of a system to define, analyze and/or validate the characteristics of a proposed system. It is a method to verify specifications for completeness and effectiveness.

1.3 Initiate Requirements Traceability

A database of proposed, or candidate, requirements should be established, and must be updated when necessary. The inputs to this database contain: system requirements, design requirements, hardware/software requirements, and test requirements. Other associated data elements should be entered for each requirement, such as method of testing, rationale, date of origination, linkages, etc.

The populating of the database is an iterative process, which begins as soon as the RM/RE process is initiated. As requirements are identified, each is entered in the database as a requirement candidate, pending reconciliation, traceability analysis, and stakeholders' approval. The database should provide traceability of all requirements and any exceptions.

In general, a requirements database can be created and maintained using a number of commercially available database management software packages. The benefits to using a database manager are the ability to rapidly store, edit, organize, analyze, and retrieve large amounts of data.

Several vendors have customized database managers to support requirements management and analysis. These products can save considerable time and effort needed to customize generic database managers for RM/RE support.

All requirements must result from a decomposition or synthesis of the higher level requirements defined in the ORD. The requirements are analyzed for traceability to determine if any of the higher level requirements were not decomposed, are not linked to a higher level requirement, or do not adequately describe the higher level requirement. The resulting linkages form a requirements traceability "tree." The traceability tree shows the hierarchical decomposition of the ORD requirements. The traceability tree should be analyzed to ensure that there is complete traceability between the ORD and the lower level specifications, as applicable (i.e., there are no missing or excess requirements).

If an ORD requirement does not decompose when translated into lower-level specifications, then either the high-level requirement does not actually exist, and it should be removed from the ORD, or the lower-level specification needs to be modified to include the decomposition. At a Requirements Inspection Conference (RIC), all irregularities must be examined in order to reach a consensus on their resolution. Each group of requirements that satisfies a single ORD requirement is analyzed to ensure that the respective groups do not conflict with each other and that they completely satisfy the higher level requirement.

1.4 Stakeholders Input Complete Determination

Prior to certifying the establishment of a baseline requirements database, a decision must be reached that all stakeholders' inputs have been elicited and documented. All requirements in the ORD must be identified, and traceability mechanisms must be in use. This list of requirements becomes the agreement between the OPNAV sponsor, PEO/PMA, and NAVAIR. It is imperative that requirements are clearly understood and traced through to implementation. If input is not complete, the process of obtaining requirements from stakeholders must continue. When the process is complete, the requirements database can be defined as "baseline."

1.5 Database Establishment

After requirements have been reconciled, the stakeholders ratify that each candidate requirement is a system requirement. The system requirements and associated data elements establish the requirements baseline. These baseline requirements are embodied in a baseline requirements database. This database is maintained and modified throughout the life of the program. Configuration management procedures must then be applied to any request for modification of any requirement.

1.5.1 Database Access.

There are two subprocesses for database access: Queries and Reporting, & Database Modification. Queries and Reporting provides the mechanism to access the database to retrieve information on the requirements and associated characteristics residing in the baseline requirements database. The information can be requested via individual ad hoc questions, or can be requested via ad hoc or pre-programmed report formats. Presentation of the requested information is in the form of on-screen

individual responses, or on-screen or hard copy reports. Modifications to the baseline requirements database can only be effected after all CM procedures have been fulfilled. The individual responsible for database administration is the only person authorized to modify the database.

2.0 Requirements Inspection/Validation

Requirements Inspection/Validation is the process of determining and linking the relationships between technical documentation. Inputs to this top level process include the entire realm of military and DoD standards, plus the system documentation available at the time of inspection: the ORD, the System Specification, and the preliminary SOW.

Another aspect of Requirements Inspection/Validation is holding conferences to review traceability deficiencies. Barren requirements, orphan requirements, ambiguous, untestable, and negatively stated requirements all must be examined. The practice of holding conferences promotes communication among team members. Outputs from the Requirements Inspection/Validation process include program-applicable standards, with the appropriate sections linked to the requirement(s) that reference them, as well as coordinated (properly linked and traceable) system requirements. Within the Requirements Inspection/Validation process are three subprocesses:

- o Requirements Validity Determination
- o Authenticated Requirements Determination
- o Forward Inputs to Baseline Requirements Database

2.1 Requirements Validity Determination

The result of Requirements Inspection/Validation determines the validity of the requirement update. If the process ascertains that a proposed requirement update is invalid, the proposed update must be re-analyzed in the Requirements Analysis Process.

2.2 Authenticated Requirement Determination

The Stakeholders' Inspection/Validation authenticates the requirement update. Authentication is approved by a single point of authority designated by the PMA. If authentication is not approved, then the proposed update must be re-analyzed in the requirements analysis process.

2.3 Forward Updates to Baseline Requirements Database

Approved updates are forwarded to the individual responsible for database administration for inclusion in the database.

3.0 Requirements Analysis

Requirements Analysis is the process of determining applicable standards, performing trade studies, reconciliation (or refinement), and cost/schedule impact analysis for each requirement. This process is performed iteratively on each new requirement. Important to this process is paying particular attention to the interaction between requirements. As development progresses and more is learned about the system, resultant changes will affect this interaction.

A change to a requirement may have a significant affect on other system requirements. Therefore, a change impact analysis must be carried out for every change request or proposed remedial action. Impact assessment is an iterative process. Once the affected requirements have been identified, all directly associated requirements must be identified. These associated requirements must be assessed for impact. This process continues until no associated requirements are affected. The subprocesses for the Requirements Analysis Process are:

- o Determine Applicable Standards
- o Trade Studies
- o Requirements Reconciliation

3.1 Determine Applicable Standards

From system requirements, DoD STDs, MIL STDs, and DoD Instruction 5000.2, the appropriate requirements can be determined for the following disciplines: maintainability, reliability, survivability, safety, EMC/EMI, human factors, ILS, QA, producibility.

Inputs such as specification numbers and paragraphs for each requirement from other functional disciplines within an organization must be solicited and coordinated. The output will be linkages between requirements and paragraphs in the standards.

3.2 Trade Studies

Trade Studies are the inquiry and investigation of the results of choosing one method of accomplishing a requirement over another. This usually takes place in terms of off-the-shelf solutions versus building requirement-specific solutions ("make or buy" decisions). Trade Studies should be accomplished before proceeding to the next RM/RE process step of Requirements Reconciliation (Refinement).

3.3 Requirements Reconciliation (Refinement)

Requirements Reconciliation (Refinement) is the process of integrating requirements within models and prototypes to evolve a picture of the completed system.

3.3.1 Specification Modeling

Specification Modeling is used to evaluate the correctness and performance of a system specification. Specification Modeling provides a more easily understood translation of the specifications requirements. Modeling ensures the completeness and correctness of the specification and validates that the requirements meet the needs of the end-users.

Specification Modeling is accomplished by translating the specification's written words, using a structured methodology, into a graphical and operable/executable language. Using this language, the specification can be visualized, analyzed, and operated by end-users. If execution of the model indicates that the system will fail to meet the users' requirements, the model is modified until a specification evolves that meets the requirements.

3.3.2 Architectural Modeling

Architectural Modeling is used to evaluate the correctness and performance of the architectural design of a system. Both hardware and software performance are evaluated by simulating the model derived from the system specification. Correctness is evaluated by executing, during the simulation, assertions (consistency constraints) that the evaluator/analyst/designer attaches to each design specification component.

Automated tools can be used to improve the quality of complex systems and reduce the cost and time required to design, implement, and optimize such systems. Tools that are capable of simulating a high degree of concurrent processing are particularly well-suited to model computer hardware and software systems within avionics systems, especially the timing considerations that are important to real-time processing environments.

3.3.3 Prototyping

Prototyping is a technique used to demonstrate the desired functionality or behavior of a proposed system. Although prototyping is often characterized as a risk management technique, it is actually a requirements generation and validation technique that reduces risk through improved system requirements. What makes prototyping so attractive (and effective) is that computer-aided tools have made the technique an inexpensive and rapid means to address system requirements early in the system development life cycle. Prototyping has the potential for improving a system engineering effort, but cannot guarantee its success. It is important to understand that prototyping augments, rather than replaces, the traditional system engineering process of specification, design, build, and test.

Prototyping has different meanings depending on whether it deals with hardware or software. For hardware, prototyping typically refers to a well-defined activity in DEMVAL or E&MD, where a physical model, exhibiting all the essential requirements as a guide for further production, is produced in advance. In contrast, software prototyping is not associated with a specific acquisition phase and does not rely on having the product characteristics well-defined in advance. Its primary goal is to serve as a learning vehicle to provide more precise ideas or specifications about what the target system should be. In many situations, a software prototype does not become part of the final product.

Although there are several categories of prototyping, exploratory prototyping, where the emphasis is on clarifying requirements of the target system, is commonly used during RM/RE. Exploratory prototyping focuses on communication problems between developers and prospective end-users, particularly in the early stages of system development. The developers often have limited information about the intended application, while the end-users have no clear idea of what the system must do for them. In this situation, a practical demonstration of possible system functions serves as a catalyst to elicit ideas and promote a creative cooperation between all the stakeholders. Such a demonstration does not have to focus on one particular solution, but can point out alternatives whose respective merits can be discussed. The prototype is an aid for establishing the features a target system should incorporate. These are subsequently codified in the systems specification.

COMPUTER SECURITY, SAFETY AND RESILIENCE REQUIREMENTS AS PART OF REQUIREMENT ENGINEERING

Prepared by DNJ Mostert and SH von Solms

Rand Afrikaans University
Department for Computer Science
P O Box 524
Johannesburg
2000
South Africa

Tel: 27 11 489 2847
Fax: 27 11 489 2138
E Mail Address: basie@rkw.rau.ac.za

COMPUTER SECURITY, SAFETY AND RESILIENCE REQUIREMENTS AS PART OF REQUIREMENT ENGINEERING

ABSTRACT

Computer security, safety and resilience are usually implemented only after a system has been developed. This leaves a lot of potential risks that must be accounted for at huge costs at a later stage. This article takes computer security, safety and resilience right to the beginning of the systems development life cycle - the user requirement specification.

Limited reference was found in the literature on how to determine the requirements for computer security, safety and resilience. This article proposes a method for determining and specifying computer security, safety and resilience requirements and to include these as part of the user requirement specification.

By using this methodology a complete set of computer security, safety and resilience requirements can be determined and specified as early as possible during the development phase.

This methodology is based on the definition of a requirements matrix by a Constraints Engineer. The importance of the different computer security, safety and resilience requirements will be rated in relation to the functional requirements, and applicable counter measures will be allocated. This will lead to justifiable costs for implementing computer security, safety and resilience for applicable systems.

The complete set of computer security, safety and resilience requirements can be used as a reference after implementation of the system to determine whether all the computer security, safety and resilience requirements have been accounted for.

ABOUT THE AUTHORS

DNJ MOSTERT

A computer consultant. Currently working towards a Phd degree at the Rand Afrikaans University.

SH VON SOLMS

Head of the department for Computer Science at the Rand Afrikaans University.

1 INTRODUCTION

Computers are increasingly used in environments where failures cannot be tolerated and where errors would have dangerously unpredictable results. [2]

Computer security, safety and resilience are usually only "implemented" after development of the system. For systems such as high performance transaction-, process control- and other safety critical systems, it became important for computer security, safety and resilience to be "part" of the system right from the user requirements specification of the system. Therefore it is important to determine and specify the computer security-, safety- and resilience requirements (hereafter referred to as CSSR) as an integrated part of the functional system requirements of the computerized system during the user requirement specification phase of systems development. Limited reference was found in the literature on methods to determine or specify computer security-, safety- or resilience requirements for a computerized system.

This article will propose a methodology used on two levels to determine and specify computer security, safety and resilience requirements as part of the user requirement specification.

> The *first level* of this methodology will be known as the Constraints Acquisition methodology (CAM) and will be used during determination of systems requirements
>
> The *second level* of this methodology will used during determination of the detail software requirements for the system and will be known as CAM/S.

A Constraints Engineer is a specialist regarding computer security, safety and resilience for a specific domain of systems. The Constraints Acquisition methodology (CAM) and Constraints Acquisition Methodology for Software (CAM/S) are the tools the Constraints Engineer will use to make sure that the CSSR requirements are part of the user system requirement specification and of the user software requirements.

Section 2 will show where that CAM and CAM/S extends the existing computer systems engineering process to include the CSSR requirements.

Section 3 will briefly explain the steps to determine the functional system requirements.

Section 4 will discuss the Requirements- and Counter-measure matrixes, which form the basis for the methodology to determine and specify computer security, safety and resilience requirements as part of the user requirement specification.

Section 5 of this article will explain the Constraints acquisition methodology for CSSR elicitation, using the matrixes introduced in Section 4.

Section 6 will explain the Constraints acquisition methodology for Software. This section will show the relationship between CAM and CAM/S. CAM/S will be used for CSSR elicitation during user software requirements determination and specification.

2 COMPUTER SYSTEMS ENGINEERING AND CAM / CAM/S

This section will show where CAM and CAM/S fits into the existing computer engineering processes.

The process from exploring the need (as specified by the user or as dictated by the business), up to implementing a computerized system can be defined as computer systems engineering. This process includes tasks like:

- specification of the system requirements,
- the software development life cycle and
- the hardware development life cycle.

User requirements specification will be done during each of these tasks in more or less detail. During the task of specifying the system requirements it is important to determine and specify the global system functions (hardware and software functions). Therefor it is important to determine the CSSR requirements bearing in mind the global picture of the system. During the Systems development life cycle it is important to expand the "system requirements" into detail software requirements. At

this stage it is important to also determine the CSSR requirements for each of those detail software requirements. This will be illustrated in Section 6 of this article.

The process of computer systems engineering can be represented by the following diagram, which also shows where CAM and CAM/S fit into the whole process. [1]

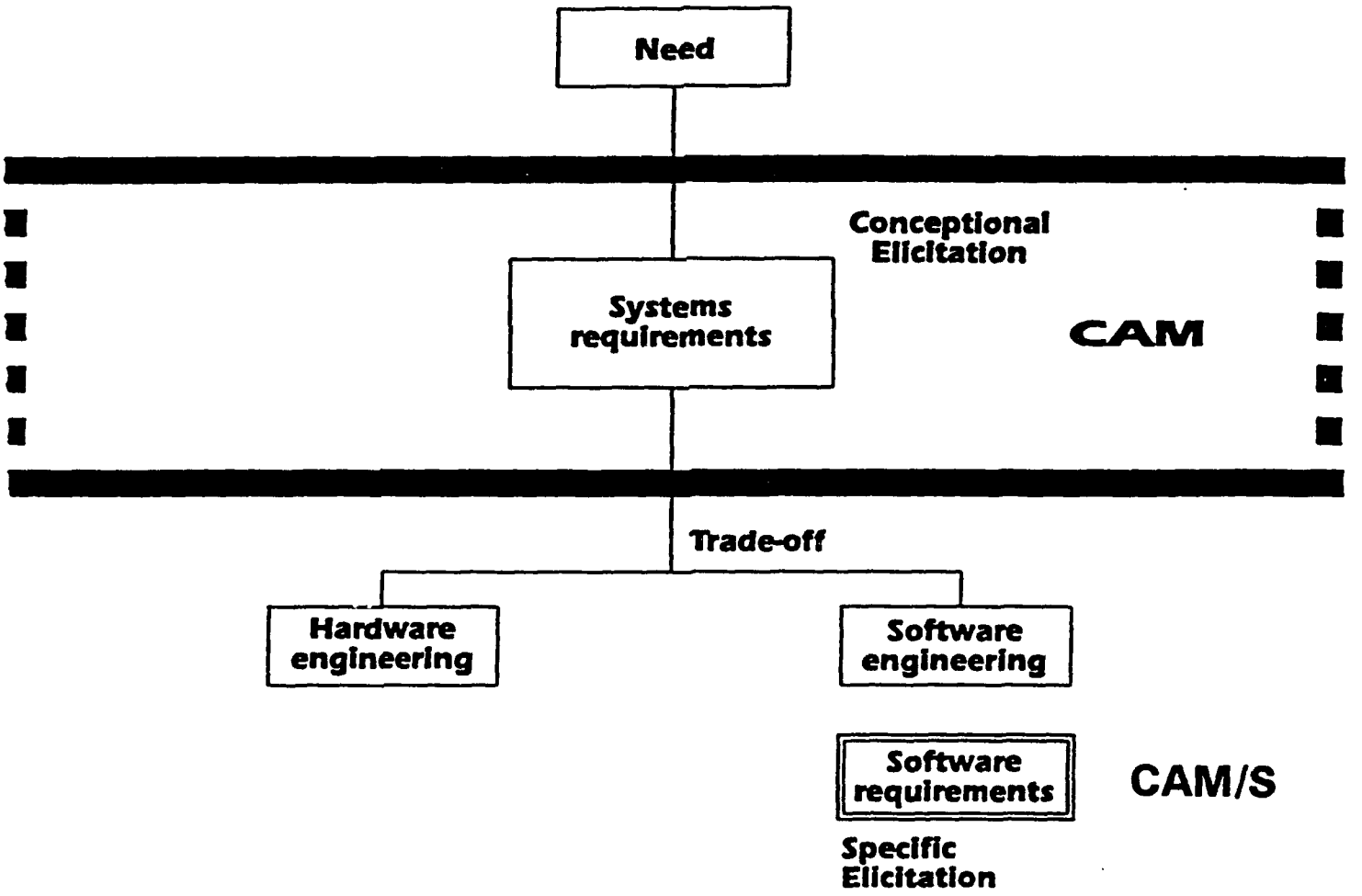


Figure 1 - The process of Computer systems engineering and CAM, CAM/S

2.1 SYSTEM REQUIREMENTS - CAM

The first step in the development of a computerized system can be characterized by the specification of the system requirements. This step precedes both hardware and software engineering.

The major objectives [1] of this step are to:

- Evaluate the systems concepts for feasibility, cost benefit and business needs
- describe the system interfaces, functions and performances
- perform preliminary functional systems requirements and design. Part of this will be to specify the systems requirements. The specification of system requirements can be accomplished in different ways. It can be an informal description of the requirements or, at the other extreme be a more formal method of a mathematical description of the requirements. It can also be a process of an informal description that "develops" into a more formal description.
- allocate functions to hardware, software and supplementary systems elements
- establish cost and schedules constraints.

This paper will concentrate on the informal description (third objective) of systems requirements. This informal system requirement will be extended to include CAM.

2.2 SOFTWARE DEVELOPMENT LIFE CYCLE - CAM/S

After determining the system requirements, certain user requirements (functions) are allocated to be implemented in software. These function will be expanded in an attempt to satisfy, mainly the following objectives [1]:

- uncovering the flow of information
- describe the software functions, validation requirements and design constraints.

Keeping in mind the overall CSSR requirements it is also important to determine and specify the CSSR requirements for these detail software requirements. Using CAM/S as a tool the CSSR requirements can be determined and specified.

3 STEPS IN FUNCTIONAL SYSTEM REQUIREMENTS ACQUISITION

It is useful to describe the process of user functional system requirements acquisition through the following steps [3]:

- Elicitation
- Formalization
- Validation

The methodology proposed in this paper will extend the elicitation and formalization steps to include the CSSR requirements with requirements acquisition.

4 BASIS FOR THE CONSTRAINTS ACQUISITION METHODOLOGY (CAM) & CONSTRAINTS ACQUISITION METHODOLOGY FOR SOFTWARE (CAM/S)

Determining the functional system requirements can be a very complex and tedious process. Proven methods and methodologies exist for determination and specification of the functional requirements. The determination of CSSR requirements received very little attention in these methodologies. The lack of a method (how) to determine and specify CSSR requirements during this phase inspired this study.

4.1 DOMAIN OF SYSTEMS

If we consider, for example, a domain of runway control systems, typical objects are: runways, aircraft, pilots and control towers. These objects are related to each other in a specific way, and therefore the CSSR requirements and counter measures that will be implemented will be the same for all systems in this domain. Only the degree of importance of the different CSSR requirements and counter measures will vary from system to system.

4.2 THE REQUIREMENTS- AND COUNTER MEASURE MATRIXES AND THE ROLE OF THE CONSTRAINTS ENGINEER

CAM and CAM/S is based on two sets of matrixes, one set for the requirements and one set for the counter measures. Although the detail level of the user requirements will differ between CAM and CAM/S the basis for using the two matrixes will be the same. A set of matrixes will be defined for CAM and a different set of matrixes will be defined for CAM/S.

A Constraints Engineer will complete these sets of matrixes. For the purpose of this article, a Constraints Engineer is a computer security, safety and resilience specialist who specializes in the domain of the required system. The paragraphs below describe the role of the Requirements and Counter measure matrixes which forms the basis of CAM and CAM/S. The Constraints Engineer and his responsibility in setting up these sets of matrixes, will also be discussed.

4.2.1 Requirements matrix sets

During development of the CAM (to be discussed in section 5) and the CAM/S (to be discussed in section 6) it was important to represent the CSSR requirements in relation to:

- the *functional requirements* of the system. For every functional requirement of the system a specification of the CSSR requirements for that functional requirement must be done.

- the *"environment"* that the system will operate in. The Users (people) of the system, are "part" of the system and are one of the major components that can jeopardise or be jeopardised by the system. Therefore it is important to specify the CSSR requirements relating to these people. An example of this requirement can be to implement the division of responsibility.

Every form of technology that will be used to implement the system brings its own kind of risk. Therefore it is also important to look at the CSSR requirements in relation to the technology to be used. For example implementing a system under the DOS operating system introduces a different

risk as opposed to a system implemented under UNIX.

To represent the relationship between a CSSR requirement, the functional requirements of the system and the environment of the system a three dimensional matrix (Requirements Matrix) is used. This matrix indicates the sub-elements of the specific CSSR requirement on the x-axis, the functional requirements of the system on the y-axis and the "environment" on the z-axis. [4] also represents the environment in a matrix format.

The sub-elements of the CSSR - requirements are:

- *Computer Security requirement:*
  - Confidentiality
  - Availability
  - Integrity
  - Assurance
- *Safety*
  - Reliability
  - Danger of being out of order
- *Resilience requirement*
  - Availability

- **Reliability**
- **Assurance**

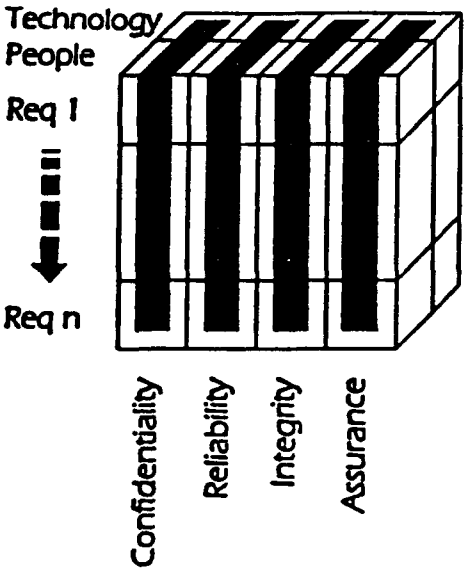
The Requirements Matrix (figure 2a) for computer security, the CS-Requirements Matrix, can be represented as a three dimensional matrix with:

- *x-axis* representing the sub-elements of computer security (confidentiality, availability, integrity and assurance). These dimensions cover the whole spectrum of computer security.

- *y-axis* representing the high level functional requirements of the required system. These requirements can be determined during sessions with the users or can be domain knowledge and can be specified in different detailed levels.

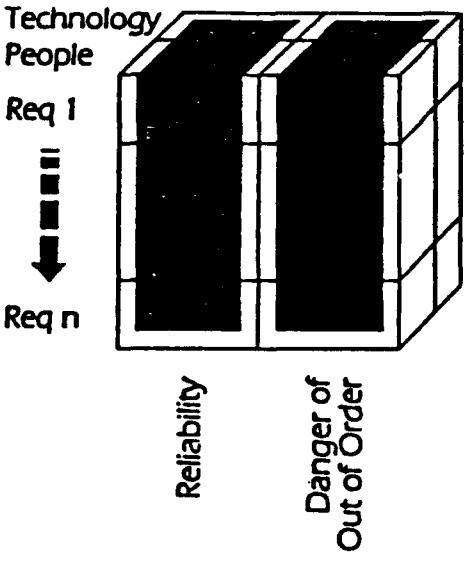
- *z-axis* representing the relevant components of the computer system, (the "environment" as discussed on a previous page).

Requirement matrixes must also be defined for Safety, the S-Requirements matrix, and Resilience the R-Requirements matrix, as indicated in the figure below. (Fig 2b & c)



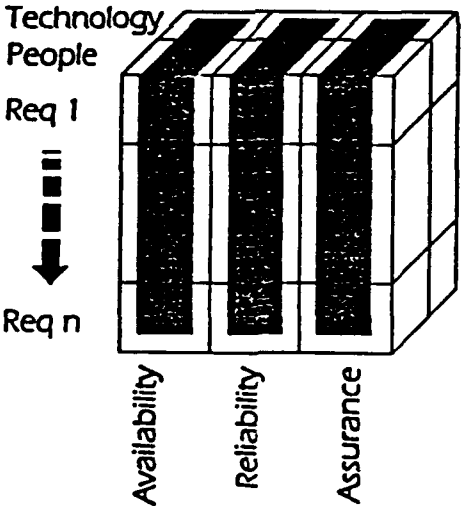
Computer Security Requirement Matrix
(CS-Requirements Matrix)

(a)



Safety Requirement Matrix
(S-Requirements Matrix)

(b)



Resilience Requirements Matrix
(R-Requirements Matrix)

(c)

Figure 2 - Requirements Matrixes

4.2.2 The Constraints Engineer

The methodology, CAM and CAM/S as described later in this paper, provide tools to the Constraints Engineer, that can be used to determine and represent the computer security, safety and resilience requirements.

The Constraints engineer creates the framework of a requirements matrix by assigning the x,y and z-axes to each of the three matrixes. The initial matrixes will create an overview of the framework which can be specified in more detail in CAM/S. This framework of the requirements matrixes can be used to structure questions used to determine CSSR requirements from the users.

The importance of the CSSR requirements can be rated by the Constraints Engineer on a scale from 0 to 3. A rating will be done for each cell of a three dimensional requirements matrix. Possible ratings allocated by the Constraints engineer are shown in Table 1.

Rating	Classes
0	No need
1	Required
2	Important
3	Very important

Table 1 - Ratings allocated by the Constraints Engineer

After completion of a specific matrix, these "ratings" can be represented and manipulated on a spreadsheet as discussed in paragraph 5.2.3.

Although completion of the requirements matrixes will be time consuming it is similar to some of the processes used in some risk analysis packages. This step by step determination of the requirements will guarantee a complete set of CSSR requirements.

4.2.3 The Counter-measure matrixes

A domain of systems introduces different objects that are related to each other in a specific way, and therefor the counter measures for the risks introduced by these systems must also be the "same" but possibly of different intensity. For example for a runway control system, the pilots will always issue landing requests. The security requirement can differ depending on the organization that will use the system. A military environment might require more security measures than a cargo carrier.

The ratings on the requirements matrix done by the constraints engineer can be used for determining a set of necessary counter measures for a specific CSSR requirement. The Counter-measure matrix contains the counter measures for a specific domain of systems.

These counter-measures matrixes are matrixes which are built-up, by the constraints engineer, after developing numerous systems. A counter-measure matrix will represent the ratings of the constraints engineer, as in Table 1, in relation to:

- the different sub-elements of the specific CSSR requirement. For example, the different sub-elements for the computer security requirement, are

confidentiality, availability, integrity and assurance

- the environment of the system (See paragraph 4.2)

Using the rating of the constraints engineer, the counter measures applicable to the specific CSSR requirement, can be determined.

This matrix will form the logical link between the CSSR requirements and the counter measures needed to address the CSSR requirements.

A Counter-measure matrix is represented as a three dimensional matrix with:

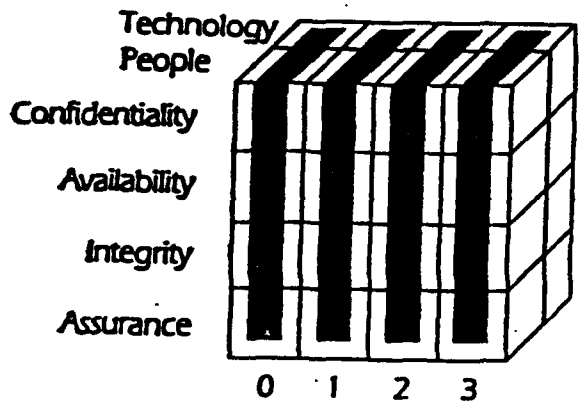
- the x-axis representing the rating 0 - 3

- the y-axis representing the sub-elements of the specific CSSR requirement. For example, for the Computer Security Counter measures matrix (CSCM-matrix) the y-axis will be confidentiality, availability, integrity and assurance. The same format holds for the Safety Counter-measures matrix (SCM-matrix), and the Resilience Counter-measure matrix (RCM-matrix).

- the z-axis giving the relevant components of the computer system (the environment).

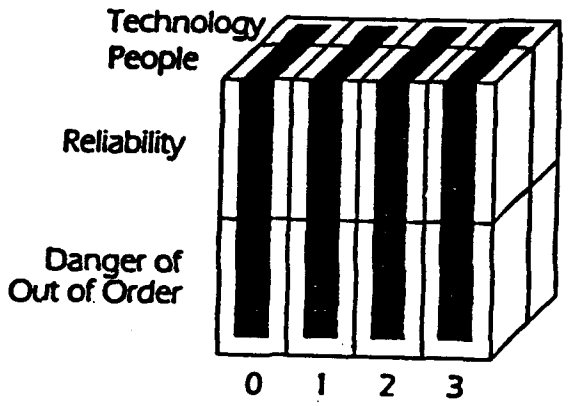
Each cell (x,y,z) of the matrix will contain the counter measures applicable for a domain of systems.

Figure 3 below indicates the CSCM, SCM and RCM matrixes.



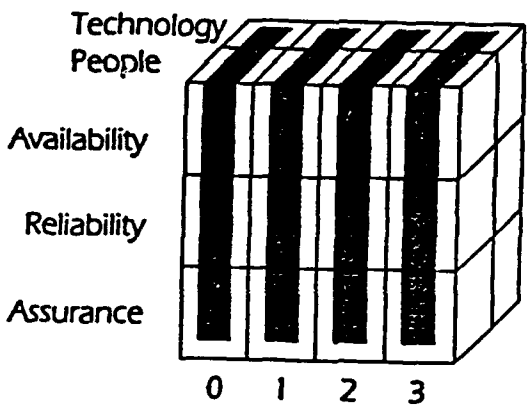
Computer Security Counter Measure Matrix (CSCM Matrix)

(a)



Safety Counter Measure Matri: (SCM Matrix)

(b)



Resilience Counter Measure Matrix (RCM Matrix)

(c)

Figure 3 - Counter measure matrixes

It should be clear to the reader that these Counter-measure matrixes are created through experience and expertise in the relevant areas.

5 CONSTRAINTS ACQUISITION METHODOLOGY (CAM)

Having now discussed the CSSR Requirements-matrixes, and the CSSR Counter-measure matrixes, we now show how the Constraints Engineer uses these matrixes to build CSSR into the functional system requirements.

The elicitation step will start of with an initial "discussion" between the User/s, the Requirements Engineer and the Constraints Engineer. The rest of this article will concentrate on the role of the Constraints Engineer.

The **elicitation step** usually takes the form of a "brain-storming session", whose goal is achieving a consensus among a group of users about **what** they want. During the elicitation step, the requirements engineer acts as a facilitator.[3] During this phase the constraints engineer will:

5.1 Classify the system or part thereof, according to the initial specified functional requirement in a domain. The following are examples of domains:

- safety critical systems
- process control systems
- on-line, networking systems
- batch systems

5.2 Compile the CSSR Requirement matrixes

5.2.1 Construct the framework ("lay-out") of the CSSR requirement matrixes by determining:

- the classes of users
- the relevant components of the computer system that will be applicable to the requirements (the y and z axis) (See section 4.2.1)

5.2.2 Populate each of the 3 requirement matrixes by rating the need for the specific CSSR requirement in relation to the functional requirement and the relevant components of the computer system, for each of the 3 CSSR-requirements, i.e. Computer security,

Safety and Resilience. Ratings are determined as set out in Table 1.

Determining the specific rating can be done by restating each requirement as a question. For example, what is the requirement for confidentiality by functional requirement 1 used for user 1? The answer to this question can be answered by:

- the user
- existing policies
- domain knowledge, for example it is accepted that a pin code should be included in an auto teller request

The answer to this question can be rated by the requirements engineer on a scale 0 to 3. 0 meaning no requirement and 3 meaning an important requirement. If the answers can be answered by more than one source, an average rating can be used.

Each CSSR requirements matrix must be completed for each of the functional requirements (y-axis), and each of the elements in the environment (z-axis).

5.2.3 Represent the resultant information on a spreadsheet.

Having determined the consent of the CSSR-matrixes during the elicitation phase, it is now used during the formalization phase.

During the formalization step of requirements acquisition the Constraints engineer.

5.3 Represents these ratings on the initial high level functional flow diagram or context diagram if applicable.

5.4 Map the requirements matrixes to the counter measure matrixes for this domain of systems.

5.5 From these matrixes the counter measures for the CSSR requirements can be listed per requirement. Cross references to counter measures can be eliminated at this stage.

At this stage the counter measures necessary to achieve the required computer security, safety and resilience in the newly planned system, had been determined, and can now be "designed into" the system right from the beginning.

Determining the CSSR requirements using the Constraints Acquisition Methodology described above, had now been done on a thorough step-by-step way, and not by any ad hoc guesses.

6 CONSTRAINTS ACQUISITION METHODOLOGY FOR SOFTWARE (CAM/S)

CAM/S extend CAM to determine and specify the CSSR requirements during the software development life cycle. CAM/S use some of the deliverables of CAM and produce the table of counter measures for the requirements to be implemented in software. A similar methodology CAM/H can be developed that will address the requirements to be implemented in hardware.

6.1 INPUTS AND MAJOR DELIVERABLES OF CAM/S

The following figure gives an indication of the relationship between CAM and CAM/S.

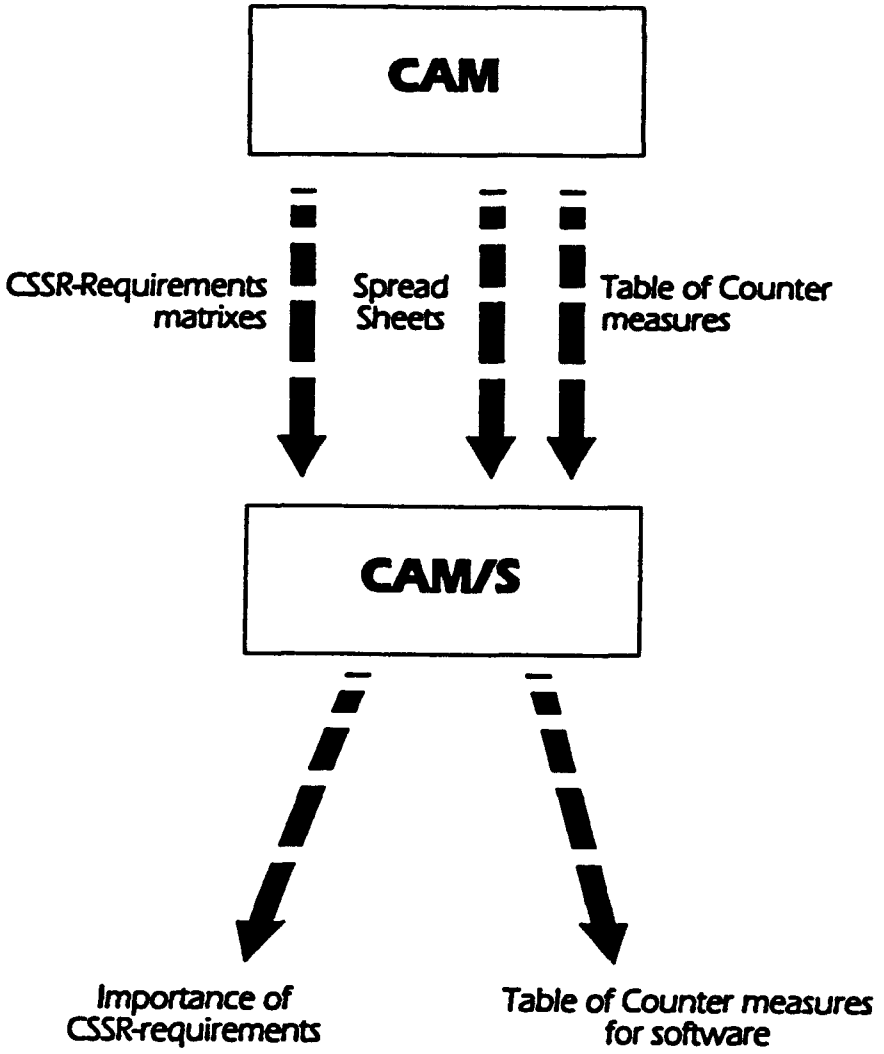


Figure 4 - Inputs and deliverables of CAM/S

6.2 CAM/S METHODOLOGY

6.2.1 Construct the set of requirements matrixes for software. Using the set of requirements matrixes from CAM, identify only those requirements for implementation in software. The following figure (figure 5) shows the requirements matrix for computer security. In this case requirements 2, 7, ... n were those specifically relevant to Software implementation. Similar matrixes must be constructed for safety and resilience.

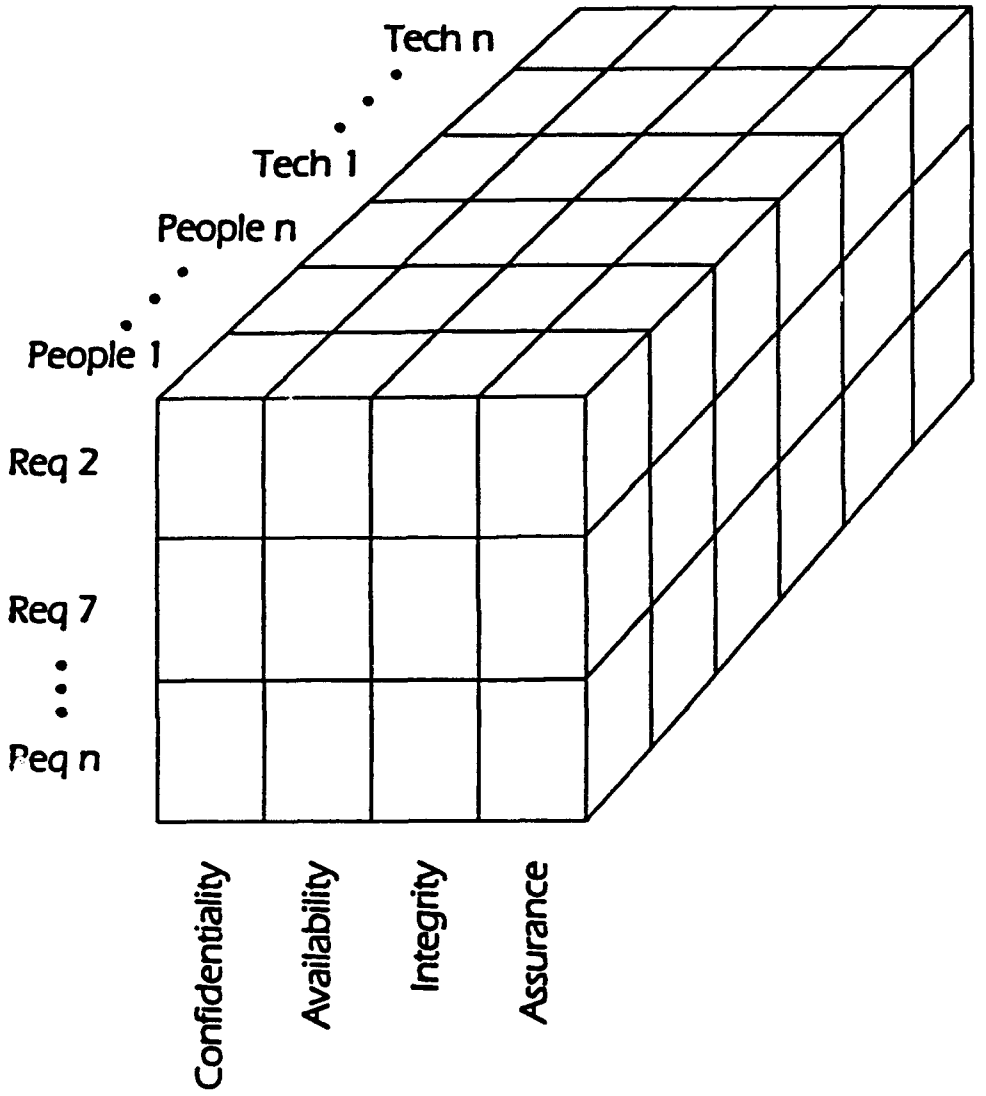


Figure 5 - Requirements matrix for computer security

6.2.2 Complete the ratings of the requirements matrix on the spreadsheet. An example is given in *Fig. 6* of the requirements matrix spreadsheet for computer security.

COMPUTER SECURITY				
	Confidentiality	**Availability**	**Integrity**	**Assurance**
Req 2				
Req 7				
. . .				
Req n				

Figure 6 - Spreadsheet of the Requirements matrix for Computer Security

6.2.3 **Determine the detail requirements and construct the new set of Requirements matrixes.** An example of the requirements matrix for computer security is indicated in *Fig 7* below. This example show that requirement 2 was expanded into requirement 2.1 - 2.3. These detailed requirements is indicated on the y-axis of the requirements matrix.

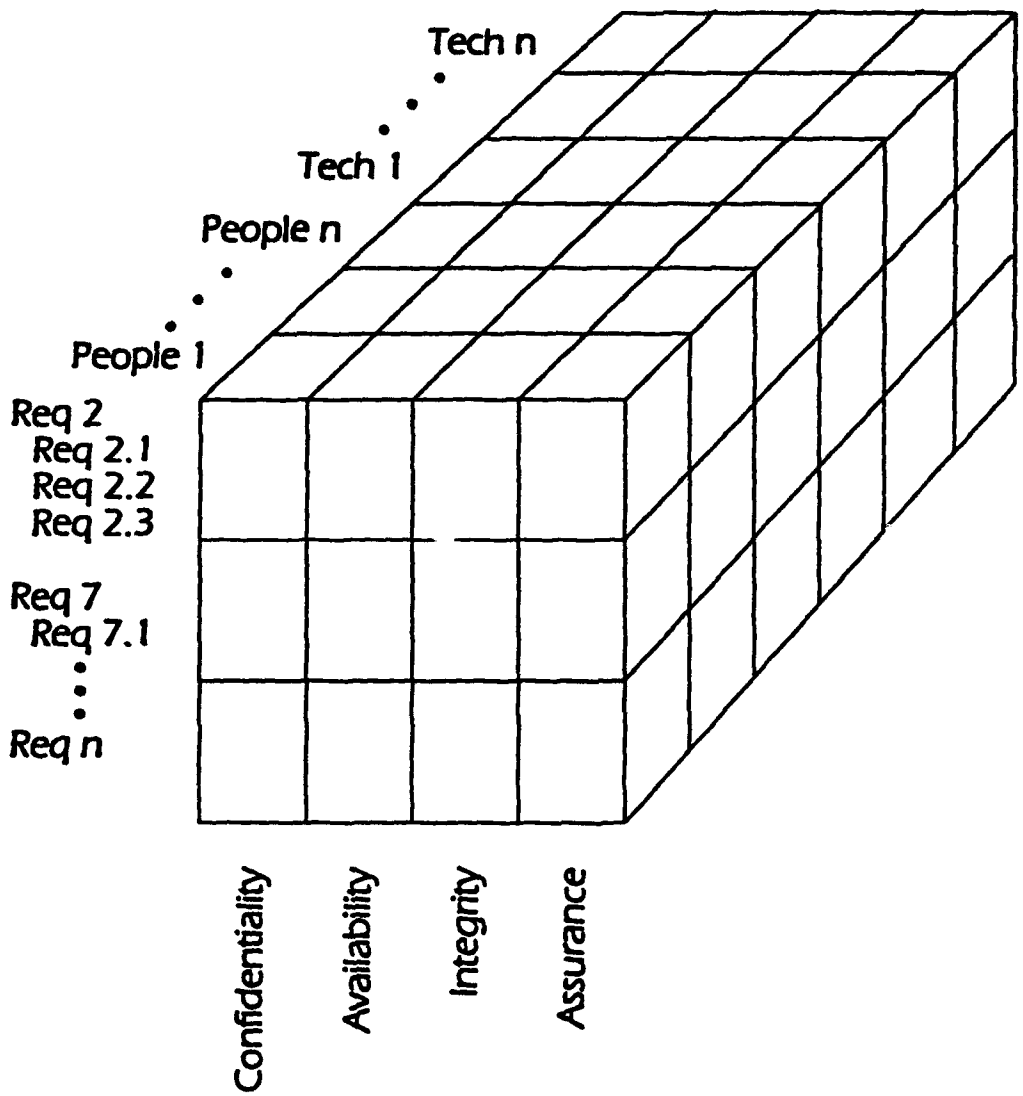


Figure 7 - Requirements matrix (Computer security) for function to be implemented in software.

- To populate each of the requirements matrixes make use of the process as described in paragraph 5.2.2.
- Represent the resultant information on the spreadsheet. Indicate the information from the Requirements matrix, from CAM.

COMPUTER SECURITY				
	Confidentiality	**Availability**	**Integrity**	**Assurance**
Req 2	3			
Req 2.1 ***Req 2.2*** ***Req 2.3***	2 2 3			
Req 7 *Req 7.1* . . .				
Req n				

Figure 8 - Spreadsheet for the requirements matrix (computer security) for software requirements.

6.2.4 Represent this ratings on a high level flow diagram.

6.2.5 Map these ratings of the requirements matrix to the counter measure matrix. This counter measure matrix will only contain the counter measures applicable to requirements to be implemented in software.

7 REFERENCES

[1] R.S. Pressman, "Software Engineering a Practitioners Approach", McGraw-Hill pp 119, 1982

[2] F.G.F Davis, R.E. Gantenbein, "Responding to Catastrophic Errors: A Design Technique for Fault-Tolerant Software", Journal for System Software, pp 243 - 251, 1992

[3] H.B. Rubenstein, R.C. Waters, "The Requirements Apprentice: Automated Assistance for Requirement Acquisition," IEEE Trans. Software Eng., Vol 17, no.3 pp 226-240, March 1991

[4] P.G. Brown, "QFD: Echoing the voice of the customer", AT & T Technical Journal, pp 18 - 32, March/April 1991

6.2.6 List the counter measures per requirement

	Computer Security	Safety	Resilience
Req 2			
Req 2.1 Req 2.2 Req 2.3			
Req 7			
Req 7.1 Req 7.2 . . .			
Req n			

Figure 9 - List of counter measures per requirement

PANEL PAPERS

DISTRIBUTED DESIGN OF COMPUTER-BASED SYSTEMS: TRACEABILITY

Stephanie White
Grumman Corporate Research Center
MS A08-35
Bethpage, NY 11714

Abstract. Distributed computer-based systems are complex due to their size, heterogeneous nature, and the dynamic interdependency of their components. Hardware and software are usually developed by a number of companies which the prime contractor and customer must rigorously monitor and control. Engineers and managers need traceability for control. They must trace requirements and design decision dependencies to create a complete and consistent design, to understand the impact of change, and to perform re-engineering without introducing errors. Automatic compilation of software and silicon may eventually eliminate the need for traceability between formal specification and end-product, but traceability will still be needed for tracing textual requirements and design decisions to formal specifications and test cases. This paper discusses the need for traceability, current practice and feasibility.

INTRODUCTION

Managers need traceability to check status and completeness, and engineers need traces to develop, test, and change the system. Unfortunately, there is little software engineering research on traceability (Finkelstein 91). Researchers are concentrating their efforts on automatically transforming detailed formal specifications into efficient software programs. When changes are necessary, engineers will change the formal specification and the "specification compiler" will generate new software. Automated software generation is already feasible for software that is not distributed or high performance. If hardware becomes sufficiently efficient, specification compilers will become a reality. At that time, traceability from formal software specifications to code will no longer be necessary. However, traceability will still be needed from high level goals, policies, and textual requirements to the formal specification and test cases.

The system engineer's need for traceability cannot be resolved by automated code generation. System design and validation require tracing from high level mission goals to policies for achieving those goals, and to system requirements, design concepts, tradeoffs, decisions, and rationale. Higher level system requirements must be allocated and traced to component requirements and traced to test cases to make sure the system requirements are met.

Dorfman estimates that there are 250 requirements in the hierarchy for every system level requirement (Dorfman 91). This creates a vast network of inter-related information. Traceability of all requirements and development information is prohibitive. It may be unnecessary or undesirable (considering overhead) to maintain linkages between less significant or non-critical requirements and every output related to them (Ramesh et al. 92).

Nevertheless, we should apply traceability with as much rigor as possible. One of the reasons for tracing information is to support re-design when requirements are modified, which can happen during or after development. It is difficult to predict in advance what may change; even non-critical requirements may be modified. Without traceability, requirements changes may not be communicated to a group affected by the requirement flowdown. Our experience is consistent with Dorfman's: "more of the requirements problems observed in system development are due to failures in requirements management than in the technical functions" (Dorfman 91).

To support impact assessment and redesign, we need more than a connection between document paragraphs, and more than a link between a requirement and a designed function or test. We must be able to trace threads of behavior from an Operational Concept Document to detailed threads in a system behavioral model. We must trace subsystem design decisions to be sure they are not in conflict with system design decisions, track that fault tolerant and other non-functional requirements are met, and validate that an entire requirement is satisfied, when the requirement traces to more than one entity.

TRACING BEHAVIOR

An Operational Concept Document (OCD) describes how the system will be used. It includes a description of environment actions (operator or other system) and corresponding system actions. System behavioral requirements, which should be traced from the OCD, are usually specified using stimulus-response threads that cross function boundaries. Unfortunately, current traceability techniques are limited for tracing behavior (DoD Software Technology Strategy 91).

Behavior is difficult to trace using current methods because current trace mechanisms link functions rather

than threads. Methods that link functions associate stimulus-response requirements with significantly large system subsets or the entire system.

Contractors should be tracing behavioral threads to more detailed behavior, and associating this behavior with test cases that drive simulations, prototypes, and implementations.

TRACING PRODUCT ATTRIBUTES

Product attributes, are difficult to trace due to their pervasive nature. We discuss the difficulties for several of these attributes.

Timing has numeric constraints and is associated with normal and exceptional behavior paths that link input to output. Timing can be traced if we solve the behavior tracing problem. The problem is that stimulus-response paths include operating system calls, database accesses, network interaction, and man-machine interface functions, as well as application functions. Timing is also dependent on resource utilization, so degraded conditions must be considered, vastly increasing the number of links.

Efficiency is a function of required processing and resource utilization. Reliability is mean time to failure in operational use and is related to failure during test. Availability is related to reliability and mean time to repair. These attributes are frequently associated with system components, but inter-component interfaces can affect the attribute.

Safe, survivable, and secure are rated by criticality level. Designers try to partition the system so that "highly critical" applies to a small part of the system, but this is difficult as we do not know how to create impenetrable boundaries. Verifying that attributes are met is equivalent to proving that bad things cannot happen, which is difficult or impossible. Therefore, strict design and coding principles must be followed, and we must trace flowdown and verification so that verifications can be repeated if a change occurs.

Fault tolerance is related to a pervasive philosophy (e.g., fail-operational/fail-safe, hot standby, compartmentalization) and to hardware and software design decisions. Tracing whether the fail-operational/fail-safe philosophy has been met is equivalent to inspecting all design and implementation. It is difficult to test a system for fault tolerance, as we cannot introduce every kind of fault. Every test that results in a failure should be analyzed for additional fault tolerance requirements, causing iterative test, requirements, design, code, test cycles, and iterative traces.

User-friendly is considered a characteristic of man-machine interfaces, including menus, screens, and on-line help. It means the system is easy to learn, easy to use, and pleasant to use. Processing and communication delays, subsystem data availability and diagnostic data availability affect user-friendliness during operation and training. Delays and data availability involve many system parts, creating traceability problems.

The pervasive nature of most product attributes indicates that it is not feasible to trace these attributes to the lowest level. To understand what must be traced, we need a defined process for specifying and verifying product attributes.

CONCLUSIONS

When contractors say they perform requirements management, they generally trace document paragraphs, functions, test cases, and resource utilization. This level of requirements management is minimal, compared to what is needed. Traceability should be part of an overall process for requirements/design flowdown and verification. This process must indicate what should be defined, traced, inspected, modeled, tested, and analyzed.

BIBLIOGRAPHY

DoD Software Technology Strategy, prepared for Director of Defense Research and Engineering, Dec. 1991.

Dorfman, M., "System and Software Requirements Engineering", in *M.H. Thayer and M. Dorfman, eds., System and Software Requirements Engineering*, Tutorial, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 4-16.

Finkelstein, A., "Tracing Back from Requirements", *IEE Colloquium on Tools and Techniques for Maintaining Traceability During Design* (Digest No. 180) Dec. 1991, pp. 7/1-7/2.

Ramesh, B., Abbott, A.G., Busch, M.R., and Edwards, M., "An Initial Model of Requirements Traceability, An Empirical Study", Technical Report, Naval Postgraduate School, Sept. 1992.

Author's Biography: Stephanie White is Principal Engineer, Software Process, in Grumman's Corporate Research Center, where she researches system and software requirements methods and leads an inter-divisional team solving system/software engineering interface issues. Her research interest is in modeling and analyzing system behavior, and she is vice-chair of the IEEE Computer Society CBSE Task Force. Previous experience includes engineering and management for aircraft and space programs, as well as college teaching. Stephanie has a BA and MS in mathematics, and a PhD in computer science.

Distributed Design of Computer-Based Systems: Needed Academic Programs

Dr. Julian Holtzman
CECASE/University of Kansas

Abstract. This paper summarizes the portion of the panel session on Distributed Design of Computer Based Systems devoted to a discussion of academic programs. Some of the key elements identified by the CBSE Task Force for Distributed Design of Computer Based Systems in both research and practice are briefly reviewed. Academic programs and some comments on supporting research follow. The basic conclusion is that current teaching programs are inadequate in content and emphasis on the Systems Engineering aspects and that university-based research, in general, lacks a focus on solving problems of significance to practitioners and rarely is "scaleable" to real-world situations.

Introduction

The IEEE Task Force on Computer Based Systems Engineering has been meeting in workshop formats for the past two-and-a-half years. Many key issues pertinent to the System Engineering of Computer Based Systems are being identified and becoming the subject of active working groups. Presently there are six working groups, including Process, Models, Tools, Education, Case Studies, and Research.

The Education Working group has been reviewing existing programs, with special emphasis on the issues and results arising from the other technical groups. The consensus has been that few academic programs provide the background for their graduates to perform the functions of CBSE and that, in general, university-based research does not meet industry needs.

Review of Some Key CBSE Issues

In a paper to be published shortly in the IEEE Computer Journal, the panel members (and some others from the CBSE group) have stated the following.

> The nature of CBSs requires a different systems engineering knowledge base than that normally required to engineer non-CBSs. All CBSs involve application software and associated services that are conceptual in nature and inherently difficult to grasp. Requirements satisfied by software are frequently ambiguous and subject to change. This leads to design changes that may sacrifice system architecture flexibility to ensure performance requirements are met. Furthermore, software changes in complex CBSs can result in unpredictable behavior, both internal and external to the CBS. The nature of distributed CBS(s) is unique in that CBS resources are frequently geographically dispersed and under the control of different organizations. To exchange data among such systems requires interfaces to describe content, and protocols to describe format.

The paper proceeds to identify critical issues and reports on the current state of the practice. For purposes of this panel, we summarize below some of the key issues pertinent to academic programs.

CBSE is concerned with the following responsibilities in addition to those of traditional Systems Engineering:

- Design decisions concerning the distributed nature of the CBS (its architecture).
- Allocation of resources to component developers and management of the coordinated process.
- Allocation of functions and data to CBS resources (processors, software, datastores, displays, Human Computer Interface).
- CBS strategies with respect to safety, security, and fault tolerance.
- Global system management strategy.
- Performance allocations (timing, sizing, availability).
- Testing (component, integration, interoperability with the external environment).
- Logistics support (maintenance, training).
- Implementation of the CBS within the existing environment (e.g., bandwidth, memory size, I/O subsystem, database system), system environment constraints (e.g., operational environment, security measures), and performance thresholds (e.g., timeliness, throughput, availability); that is, the more traditional Systems Engineering issues.

Considerations from an Academic Perspective

A general observation at this point is that CBSE education must be grounded in both Computer Engineering and Computer Science but with a strong emphasis on System Engineering issues. Review of some key academic programs in Computer Engineering in North America revealed that most of the programs reflected the underpinnings of the normal Electrical Engineering programs: e.g. communications, VLSI design, and so forth. Only in the last few years have Computer Engineering programs begun to take on their own identity.

Computer Science programs also reflect the emphasis of their origin rather than the needs of the practitioners. Most are based on abstract mathematical theory and stress provability more than on what will work, and seem driven by the desire to produce people to write compilers.

Other CBSE related programs are extensions of Industrial Engineering, Operations Research, and so forth. Interestingly, these programs address some key systems engineering issues but lack the depth in the background discipline areas needed for CBSE.

Undergraduate programs commonly are greatly lacking in a systems engineering emphasis, or even flavor. The traditional departments, over the last twenty or thirty years, have erected almost insurmountable barriers between themselves at the same time these barriers are breaking down in industry. Specialties are dropped in one department and picked up by another (e.g., control theory from EE to Aerospace) rather than being shared by interdisciplinary groups of interested parties. Nonetheless, it is important to address the complex system engineering issues in all disciplines; and it may require courses labeled "Systems Engineering for XX" to meet the current academic cultural environment.

Most undergraduate programs concentrate on what can be termed grading by artifact. That is, very little attention is given the method used in obtaining a solution; rather, we grade the specific answer. Thus it should not be a surprise that graduates of such programs do not understand the process of problem solving but are specialists in very narrow particular solution methods. Moreover, little attention is given to seeking and evaluating alternate solutions. Optimization is rarely examined, and certainly even more rarely is system optimization considered. Of special concern to a CBSE program is the concentration on multiple small problems that are presented to the students. They do not usually have many, or good, experiences in large, team-oriented projects--projects that would be representative of a CBS.

There is little exposure to, and experience in, micro-economics. Instead, a single course in macroeconomics is included in most programs. It is little wonder, then, that engineers and systems engineers find themselves the "victims" of cost effectiveness analyses rather than being in charge of them.

Traditional university research is focused on publications in refereed journals--many of them, and rapidly. Since both the tenure and promotion considerations hinge on publications, concentration on industrial strength and industry-pertinent research is not present. Typically, a research problem is selected that will yield publishable results without much concern as to the realism of the simplifying assumptions. Results of the research are rarely tested against real benchmarks. As in the design of courses, the problems selected are usually of small size. Scaling to commercial-sized problems is almost never done.

Recommendations

A fundamental issue the CBSE Education Working Group is debating is the level; i.e., undergraduate versus master. There are proponents of an undergraduate program, in spite of the observation that it is highly unlikely that an individual four or five years out of high school will perform systems engineering work even at the detail design level. A consensus seems to be developing among the working group members that a hybrid undergraduate program concentrating on discipline fundamentals with a strong emphasis on true design principles, addressing many of the issues raised above, is the desired course of action. A very early draft of a model program will be given during the panel discussion.

There is general agreement between NCOSE members and CBSE Task Force members that an identifiable Systems Engineering program at the graduate level is a necessity. Both organizations have working groups addressing this problem. During the panel session, we will present a preliminary draft of a Master's level program for CBSE with two tracks: one predominantly technical with a strong emphasis on formal methods, and the second with a strong flavor of management. The reality of program duration constraints suggests two tracks, even though a single track with elements of each would be more desirable.

There are several extremely serious issues that must be addressed beyond the technical content of academic programs. The research content and nature of graduate programs requires definition and support. The source of faculty with the appropriate CBSE expertise is of concern. It is most likely that participation by industry will be required. Some of the cultural aspects have been discussed above. A publication from the National Research Council outlining the need for a resurgence in both research and teaching in design observed that universities tend to, first, deny there is a problem and, second, state that it is impossible to change. NCOSE and the Task Force, in partnership with academia, have a significant effort in front of them.

Author's Biography

Dr. Holtzman is a Professor at the University of Kansas and is Director of KTEC's Center for Excellence in Computer Aided Systems Engineering. He has over 33 years of experience in higher education and in the private sector. He has worked as a member of the technical staff at Hughes Aircraft, rising to the rank of senior research scientist while at Lockheed Missiles and Space Corporation, and has served as Chairman of the Electrical and Computer Engineering Department. Dr. Holtzman has authored or been a co-author on 90 papers. He has had approximately seven million dollars in sponsored research as Principal Investigator. Dr. Holtzman's current research is in Systems Engineering focusing on Tool Integration, Software Reuse, and Enterprise Modeling.

Distributed Design of Computer Based Systems: Methodology

David W. Oliver
GE Corporate Research and Development
P.O. Box 8, Bldg. K-1, Room 3C3
Schenectady, N.Y. 12301

Abstract. The distributed design of computer based systems requires a methodology that supports both the modeling of the distributed system and the ability to assess the impact of requirements and engineering change while the system is at any stage of development. Model Based Systems Engineering, MBSE, provides an approach to these twined problems with the advantages of tailorability to application, culture, notations, and tools. It is critical that traceability linkages be constructed in the normal course of engineering development and that they be adequate for an analysis of the impact of change.

Modeling of Distributed Systems

The systems modeling of large distributed systems is done in tiers, beginning at context level and continuing to a separation tier where hardware, software, the tasks of people, and facilities can be separated and specified independently. The systems modeling must assure that the implementations from all these disciplines will work together upon integration. This requires the careful modeling of threads through the system as shown in Figure 1.

The threads can be threads of control which may or may not involve people in the loop. The threads may be stimulus response threads which may or may not involve people in the thread. These threads pass through many layers and components of the system and it is the system responsiveness that is critical. There may be several independent threads which can occur randomly in time and can interfere under special occurrence conditions. It is critical that such situations be modeled and that simultaneous need for the same resource be prevented. The use of bus structures, complex queues, and distributed data storage must be carefully evaluated.

Thread analysis begins at context level by thoroughly modeling the behavior of the external components or entities in the environment. A methodology, MBSE, has been described for accomplishing this, (Oliver 1993a), (Oliver 1993b), (Oliver 1993c), in a manner that is tailorable to different representations of the information,

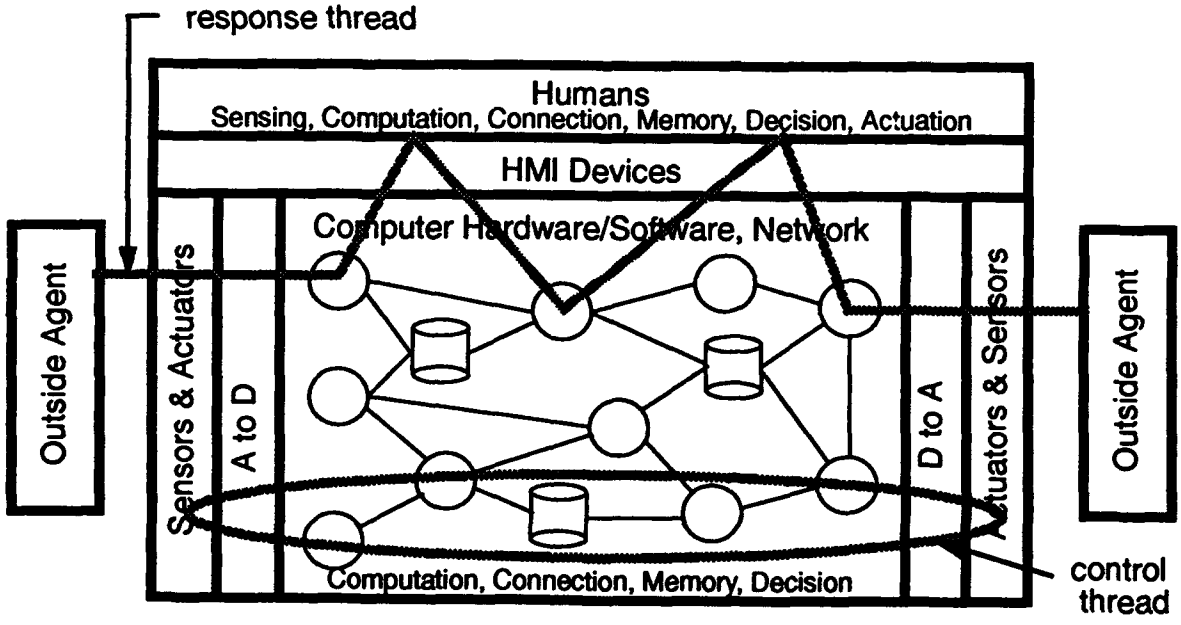


Figure 1. Critical threads through a system which has been fully decomposed to separate software, hardware, and operator components

different notations and different tools. The seven core engineering steps and their application to tiers of decomposition is shown in Figure 2.

These models capture behavior, what a component does, and are called Conceptual or What Models. The models are executable. They link, for traceability, to the Operations Concept documents. As the behavior of the system is modeled in a hierarchy of increasing detail, What Models in the same executable notation are developed at each tier. The external behaviors are linked to the system behavior and executed together. This process

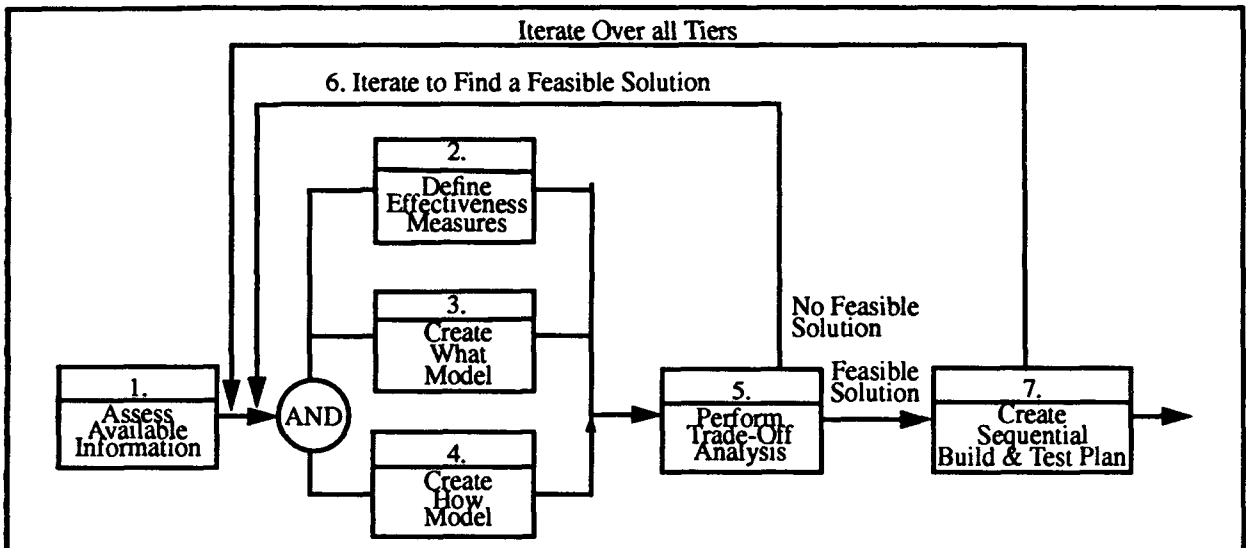


Figure 2. Systems Engineering, Core Steps

drives threads and time lines through the entire system to assess resource conflicts.

The thread analysis also supports the development of the Sequential Build and Test Plan, step 7. Issues can be raised at any point of development and the issue descriptions link to the associated entities in the models. Resolution of the issues is recorded and linked similarly. Inherent in the method are trade-off and optimization. The behavior of all components must be modeled Data store notation is not adequate Requests to operating systems, busses, and DBMS's must be modeled.

Information models, How Models, have been described, (Oliver 1993a), for each of the seven core engineering steps. They show the relationships among all the entities in the models. Under the sponsorship of the IEEE Task Force on Computer Based Systems Engineering the core steps and their information models are being scrutinized by Systems Engineering groups around the world for correctness and completeness.

Acknowledgments

This work was supported by the Naval Surface Warfare Center, Engineering of Complex Systems (ECS), Contract N60921-92-C-0206. The author thanks Philip Q. Hwang and Jonah Lavi for encouragement and support.

References

(Oliver 1993a) Oliver, David W., "A Tailorable Process Model for CBSE.",Interim Contract Report, Engineering of Complex Systems, Contract N60921-92-C-0206, U.S. Navy, NSWC

(Oliver 1993b) Oliver, David W., "Descriptions of Systems Engineering Methodologies and Comparisons of Information Representations.", NCOSE, July 1993, Washington, D.C.

(Oliver 1993c) Oliver, David W. "Automated Optimization of systems Architectures for Performance.", NCOSE, July 1993, Washington, D.C.

Author's Biography

Dave Oliver has worked at GE-Corporate Research and Development as a staff physicist, systems engineer and manager for thirty two years. He is currently developing Systems Engineering methodology and tools, adding optimization and object-oriented methods for large systems which may be real-time and distributed. He led the development of the Teamwork Ada CASE tool, and a X-Ray tomographic inspection system for turbine blades. He has contributed to medical real-time ultrasonic diagnostic systems, eddy current imaging, and high temperature crystal growth and material processing. He has managed GE-CRD technical liaison with all the GE departments and the CRD Automation and Control Laboratory for factory automation and quality. He is a graduate of VPI and MIT with a Ph.D. in physics 1961.

Distributed Design of Computer-Based System

David G. Owens
6111, avenue Royalmount
Montreal Canada PQ H4P 1K6

This paper uses a drawing tool, a spread sheet, and functional strings, to support high level analysis of a conceptual system. By providing the data required and investigating suspicious areas, the Systems Engineer can minimize the number of surprises that occur later in the developmental process. The same spread sheet information can be built on to assess and present the difference in cost and performance compared to a baseline.

Simple mistakes are the most embarrassing. When a system's function and interfaces are defined in a static structure, it is often discovered that it could not perform the minimum operations dynamically. When discovered in an early stage of the design, the problem may be easy to correct, but, it was still embarrassing to get caught by a design oversight.

The purpose of this short paper is to offer a method to define and analize a static view of a functional allocation. This is done by looking at estimates of the best case and worse case dynamic limits of the allocated system, which allows an understanding of the functional operation within a given set of system constraints. Performing this analysis in a complex system requires the use of an accounting system that is as simple as the pencil but has the power of the computer. A spread sheet program fits the bill. This is done using a top level equation:

$$\text{Dynamic_Capability_USED} =$$

$$\sum \%I/O_used - per_function \pm \sum tollerences$$

$$+$$

$$\sum \%CPU_used_ - per_function \pm \sum tollerences$$

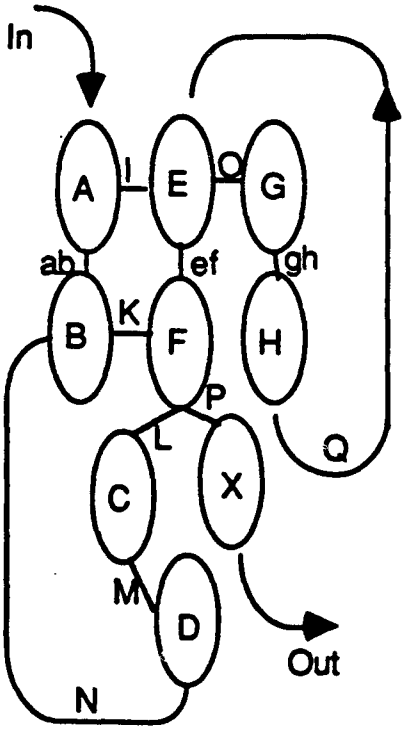
$$+$$

$$\text{Reserve}$$

When this equation and the spread sheet become ineffective, move the analysis to a data base or systems engineering design/analysis tool.

A technique used to identify and isolate strings of linked computation used by models of high performance computer systems is recommended to support the spread sheet model. This simple allocation of known or desired information put into an operational (time based) string allows high level accounting practices to be used to keep the books on a percentage capability used bases. These strings can be parallel or serial operations. The strings can have bounded interactions showing communication or constraints. The details of the operations can be at a high level (see Figure 1.), and the dynamics need only include best and worst case performance estimates. Interactions through physical communication channels include needed volume and rate estimates with best and worst case limits.

The view of a simple system (Figure 1) can be deceptive. The following example points out how easy it is to assemble a set of operations and tolerances and match them with implementation capability and constraints. Then one can perform a top level "does it fit?" analysis. The Model is easy to implement with spread sheet tools such as Excel or Lotus.



Defined Operations
and Allocated Structure

Figure 1.

How should the model work? All good models are assembled to answer the specific questions of the operation and performance within the constraints of the proposed implementation. This model will be analyzed for bottle necks and performance tolerances. It could also answer questions about allocation of maximum or minimum performance or many other numeric data, such as errors or allocated reliability. Once the model is created (defined), simple statistical or numerical analysis techniques built into spread sheets can be used to automate and display the data. A graphical representation with the spread sheet's numbers helps provide consistency of the model at the level of detail included.

The following example uses a top level description of a process to perform an operational constraints analysis. The system is defined by Figure 1. The flow through the system uses ab, I, K, O, gh, Q, ef L, M, N, & P, communication paths through 9 processes to finally get to the output.

$$Estimate-of-\text{Dynamic_Capability_USED} =$$

$$\sum Input_used \approx 40\% \pm 10\%$$

$$\sum Process_used \approx 60\% \pm 20\%$$

$$\sum Output_used \approx 70\% \pm 20$$

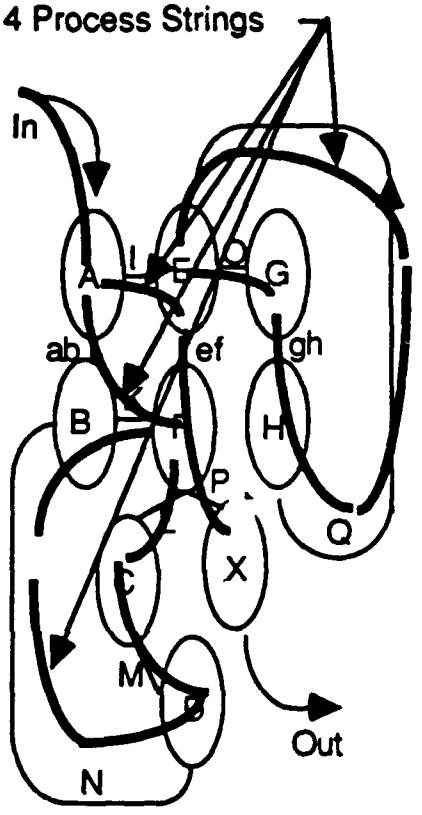
$$Required_Reserve \approx 10\%$$

$$Estimated_Confidence_level \approx 95\%$$

This model has more than 81 potential states of operation. The allocated processes and communications could require a significant effort to completely define or simulate. Using Excel's Γ function, if there is a skew between the estimates and the operational data, the margin is about 18% and is still safe.

Figure 2. This model depicts a more detailed description of the system with normal processes defined. The string analogy shows processes and how they are allocated resources and communications by stringing a series of functions that would be followed in the systems normal operation. The two normal strings are: A="IN-A-ab-B-K-F-P-X-Out"; and B= "IN-A-I-E-ef-F-P-X-Out". Both Normal threads have error correcting strings "L-C-M-D-N-B-K-F" and "O-G-gh-h-Q-E" that add complexity to the allocating processes to nodes B-E & F. The effect of these two looped back strings can be analyzed using the spread sheet.

Assuming equal distribution between A & B, node F is loaded to 60 %. If A uses its secondary string 50% of the time, the load on F jumps to 90% load. If the loading distributions are not normal then there is likely an intermittent problem with the systems operation. Using the statistical package within the spread sheet allows the generation of a complete set of data on this potential bottleneck.



Defined Operations
and Allocated Structure

Figure 2.

String analysis uses a set of heuristics to group, classify and analyize static and dynamic state space at a high level. The technique does not require assumptions to be atomic at any level. Lower level detailed interactions and constraints are included in the top system description only when necessary. The analysis moves to Petri Nets or complete tools such as RDD or State Mate as the information and precision increases.

Abstract. This paper introduces a method of modeling that uses spread sheet tools to support the analysis of high level function and performance allocations early in the design. The thrust of this short paper is in support of the Computer Based Systems Engineering Panel discussion of the need for modeling during system design.

Panel Description: Computer Security Tradeoffs

Catherine Meadows
Code 5543
Naval Research Laboratory
Washington DC, 20375

In this panel we propose to consider tradeoffs between system security and other critical system requirements, where by system security we mean the maintenance of data confidentiality and integrity in the face of hostile intruders who are actively trying to subvert the security policy. In the past, security was usually identified with confidentiality, and it was usually considered more or less in isolation from other system properties, particularly in the DoD. That is, the main function of a secure system was to be secure; other properties were usually secondary. Such an approach was adequate when it was applied to the problem of protecting data in stand-alone operating systems that had no other critical requirements and were not part of any larger system. But now we are seeing the need for for more complex systems with other requirements as critical as security, including availability, reliability, and timing constraints. It is these kinds of features that are often most in conflict with security requirements. Guaranteeing secrecy may mean making both the data and the system less available. Requiring that the system not operate in certain insecure modes may make it less reliable. Finally, the time involved in enforcing secrecy and integrity constraints may negatively affect performance, both because of the time involved in checking the constraints, and because the restriction of information flow to secure channels may prevent information from being processed in the fastest and most convenient way.

In order to build a system in which various critical properties can be guaranteed to an acceptable degree, it is necessary to understand the tradeoffs between security and the other properties. In this panel we are gathering together researchers who are working on computer security as it affects various other possibly critical properties of a system We plan to consider the following questions:

[1] What effect does enforcing security requirements have on a system's ability to meet other requirements, such as reliability, availability, and real-time requirements? In particular, what kinds of security requirements are in direct conflict with other systems requirements, and what kinds can be seen as working together with other systems requirements? For example, in what ways is a secure system more or less reliable, and in what ways is a reliable system more or less secure?

[2] Are there aspects of security (e.g., covert channel rates) that can be easily quantified and compared with other quantifiable requirements? What aspects can not be so easily quantified? What can we done to make them more quantifiable?

[3] Are there techniques from security that can be helpful in assuring that a system meets its other requirements, or vice versa?

[4] If a system must meet other requirements such as dependability, etc., are there any kinds of security requirements that it is more likely to have? What information from systems development is most likely to give us an answer to this question?

[5] Does one need to think of security in a different kind of way when thinking of it in conjunction with other system requirements? In particular, does one need a different way of developing and defining security policies?

[6] Properties not usually associated with security (e.g., timing or reliability) may also be considered security properties when they must be maintained in face of hostile attack. How does this change the way we think about them, and how does it change the way we think about security? Finally, how does thinking of these properties in this way affect the tradeoffs?

[7] How compatible with other system objectives are the assurance techniques used in computer security? How are assurance problems addressed in other, related communities?

Proposed Panelists:

Catherine Meadows, NRL, chair
Marshall Abrams, MITRE
Teresa Lunt, SRI
Carl Landwehr, NRL

APPENDIX A

LIST OF PANELS

RESEARCH AND TECHNOLOGY VISION PANEL
Chair: Phillip Q. Hwang
Members: Bruce Blum, David Owens, Gary Koob, Evan Lock

COMPUTER SECURITY TRADE-OFF PANEL
Chair: Cathy Meadows
Members: Marshall Abrams, Teresa Lunt, Carl Landwehr

COMPUTER BASED SYSTEM ENGINEERING (CBSE) ISSUES AND DIRECTIONS PANEL
Chair: Dave Oliver
Members: Phil Hwang, Stephanie White, David Owens, Nick Karangelen

SYSTEM INTEGRATION PANEL
Chair: Evan Lock
Members: Nick Karangelen, Kane Kim, Robert Goettge, Osman Balci, Norman Scheidewind

REQUIREMENTS AND TRACEABILITY PANEL
Chair: Stephanie White
Members: Ralph Jeffords, Richard Evans, Luke Campbell, Dan Mostert, Dave Bergstein

APPENDIX B

LIST OF ATTENDEES

MARSHALL ABRAMS
M/S: Z202
MITRE
7525 COLSHIRE DR
MCLEAN VA 22102-
PHONE: (703)883-6938
FAX: (703)883-1397
EMAIL: ABRAMS@MITRE.ORG

MACK ALFORD
M/S:
ASCENT LOGIC CORP
180 ROSE ORCHARD WAY STE 200
SAN JOSE CA 95134-
PHONE: (408)943-0630
FAX: (408)943-0705
EMAIL: ALFORD@ALC.COM

ED ANDERT
M/S:
CONCEPTUAL SOFTWARE SYSTEMS
P O BOX 727
YORBA LINDA CA 92686-0727
PHONE: (714)996-2935
FAX: (714)572-1950
EMAIL: ANDERT@ORION.OAC.UCI.EDU

ANNETTE ASHTON
M/S: CODE K52
NSWCDD

DAHLGREN VA 22448-5000
PHONE: (703)663-7927
FAX:
EMAIL: AASHTON@RELAY.NSWC.NAVY.MIL

OSMAN BALCI
M/S: DEPARTMENT OF CS
VIRGINIA TECH

BLACKSBURG VA 24061-0106
PHONE: (703)231-4841
FAX: (703)231-6075
EMAIL: BALCI@VTOPUS.CS.VT.EDU

SHERRY BARKER
M/S: CODE B10
NSWCDD

DAHLGREN VA 22448-5000
PHONE: (703)663-7378
FAX:
EMAIL:

DAVID BERGSTEIN
M/S: CODE N51
NSWCDD
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
PHONE: (301)394-2770
FAX: (301)394-4130
EMAIL: DBERGST@NSWC.NAVY.MIL

BRUCE BLUM
M/S:
JHU/APL
JOHNS HOPKINS RD
LAUREL MD 20723-6099
PHONE: (301)953-6235
FAX: (301)953-6904
EMAIL: BIB@APLCOMM.JHUAPL.EDU

DAVID BRITTON
M/S:
TRIDENT SYSTEMS INC
10201 LEE HWY STE 300
FAIRFAX VA 22030-
PHONE: (703)691-7792
FAX: (703)273-6608
EMAIL:

JOHN BURCH
M/S:
AEPCO
15800 CRABBS BRANCH WAY STE 300
ROCKVILLE MD 20855-
PHONE: (301)670-6770
FAX: (301)670-9884
EMAIL:

LUKE CAMPBELL
M/S: BLDG 2035 SY30
NAWC-AD/PAX

PAX RIVER MD 20670-
PHONE: (301)826-7601
FAX: (301)826-7607
EMAIL: LCAMPBEL@TECNET1.JCTE.JCS.MIL

JOSEPH CHIARA
M/S: MTE
AF SPACE AND MISSILES CEN
LAAFB
LOS ANGELES CA 90009-
PHONE: (310)363-3521
FAX: (310)363-0265
EMAIL: CHIARA@MT2.LAAFB.AF.MIL

DONG CHOI
M/S: BOX 0191
UNIVERSITY OF PENNSYLVANIA
3650 CHESTNUT ST
PHILADELPHIA PA 19104-6107
PHONE: (215)573-4503
FAX:
EMAIL:

THOMAS C. CHOINSKI
M/S: CODE 2151
NUWCDET
BUILDING 80
NEW LONDON CT 06320-
PHONE: (203)440-5391
FAX: (203)440-5243
EMAIL:

DANIEL DAYTON
M/S:
JRS RESEARCH LABS
1036 W TAFT AVE
ORANGE CA 92665-4121
PHONE: (714)974-2201
FAX: (714)974-2540
EMAIL: DAN@JRS.COM

WILLIAM DUDZIK
M/S:
AST INC
5113 LEESBURG PK STE 514
FALLS CHURCH VA 22305-
PHONE: (703)845-0040
FAX: (703)845-0042
EMAIL:

MICHAEL EDWARDS
M/S: CODE B40
NSWCDD
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
PHONE: (301)394-4187
FAX: (301)394-1175
EMAIL: MEDWARD@NSWC-WO.NAVY.MIL

WILLIAM EVANCO
M/S: M/S Z385
MITRE CORP
7525 COLSHIRE DR
MCLEAN VA 22102-3481
PHONE: (703)883-6102
FAX: (703)883-5787
EMAIL: EVANCO@MITRE.ORG

RICHARD EVANS
M/S: ST II RM133
GEORGE MASON UNIVERSITY
4400 UNIVERSITY DR
FAIRFAX VA 22030-4444
PHONE: (703)993-3724
FAX: (703)993-1521
EMAIL: REVANS@GMUVAX2.GMU.EDU

WILLIAM FARR
M/S: CODE B10
NSWCDD

DAHLGREN VA 22448-5000
PHONE: (703)663-8388
FAX: (703)663-4568
EMAIL: WFARR@S850.MWC.EDU

JAMES FRANCIS
M/S:
STRATEGIC INSIGHT LTD
2011 CRYSTAL DR STE 101
ARLINGTON VA 22202-
PHONE: (703)553-9700
FAX: (703)553-9665
EMAIL:

ARMEN GABRIELIAN
M/S:
UNIVIEW SYSTEMS
1192 ELENA PRIVADA
MOUNTAIN VIEW CA 94040-
PHONE: (415)968-3476
FAX: (415)968-3476
EMAIL: ARMEN@WELL.SF.CA.US

DENNIS GARROOD
M/S:
ALLIANT TECHSYSTEMS INC
6500 HARBOUR HEIGHTS PKWY
EVERETT WA 98275-
PHONE: (206)356-3293
FAX: (206)356-3185
EMAIL:

JOSEPH GERSTNER
M/S:
XRF
8370 GREENSBORO DR. #919
MCLEAN VA 22102-
PHONE: (703)442-9020
FAX: (703)442-9020
EMAIL: JGERST@CS.GMU.EDU

ROBERT GOETTGE
M/S:
AST INC
12200 E. BRIARWOOD AVE. STE 260
ENGLEWOOD CO 80112-
PHONE: (303)790-4242
FAX: (303)790-2816
EMAIL:

JEFFERY GRADY
M/S: SPACE SYSTEMS DIVISION
GENERAL DYNAMICS
6015 CHARAE ST
SAN DIEGO CA 92122-
PHONE: (619)547-7108
FAX: (619)974-4000
EMAIL:

ROBERT HALLIGAN
M/S:
TECHNOLOGY AUSTRALASIA PTY LTD
1010 DONCASTER RD
DONCASTER E VIC AUSTRALIA 3109
PHONE: 61.3.841.9733
FAX: 61.3.841.8374
EMAIL:

STEVE HARRISON
M/S: CODE 2153
NUWCDET

NEW LONDON CT 06320-
PHONE: (203)440-6153
FAX: (203)440-5987
EMAIL: HARRISONSJ@NUSC.NAVY.MIL

ROGER HILLSON
M/S: CODE 5583
NAVAL RESEARCH LABORATORY
4555 OVERLOOK AVE SW
WASHINGTON DC 20375-5337
PHONE: (202)404-7332
FAX: (202)767-1122
EMAIL: HILLSON@AIT.NRL.NAVY.MIL

NGOCDUNG HOANG
M/S: CODE B40
NSWCDD
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
PHONE: (301)394-4877
FAX: (301)394-1175
EMAIL: NHOANG@NSWC-WO.NAVY.MIL

JULIAN HOLTZMAN
M/S: CE/CASE
UNIVERSITY OF KANSAS
2291 IRVING HILL RD NICHOLS HALL
LAWRENCE KS 66045-
PHONE: (913)864-7759
FAX: (913)864-7789
EMAIL: HOLTZMAN@KUHAB.CC.UKANS.EDU

STEVEN HOWELL
M/S: CODE B40
NSWCDD
10901 NEW HAMPSHIRE AVE.
SILVER SPRING MD 20903-5640
PHONE: (301)394-3987
FAX: (301)394-1175
EMAIL: SHOWELL@NSWC-WO.NAVY.MIL

MICHELLE HUGUE
M/S: AEROSPACE TECHNOLOGY CENT
ALLIED-SIGNAL AEROSPACE CO
9140 OLD ANNAPOLIS RD
COLUMBIA MD 21045-
PHONE: (410)964-4158
FAX: (410)992-5813
EMAIL: MMH@BATC.ALLIED.COM

PHILIP HWANG
M/S: CODE A-10
DMA HQ(TIS)
8613 LEE HWY
FAIRFAX VA 22031-2137
PHONE: (703)285-9236
FAX: (703)285-9396
EMAIL: PQHWANG@CS.UMD.EDU

FARNAM JAHANIAN
M/S: MS H2-B22
IBM RESEARCH
P.O. BOX 704
YORKTOWN HEIGHTS NY 10598-
PHONE: (914)784-7498
FAX: (914)784-7455
EMAIL: FARNAM@WATSON.IBM.COM

RALPH JEFFORDS
M/S: CODE 5546
NAVAL RESEARCH LABORATORY
4555 OVERLOOK AVE SW
WASHINGTON DC 20375-5000
PHONE: (202)404-8493
FAX: (202)404-7942
EMAIL: JEFFORDS@ITD.NRL.NAVY.MIL

DAVID JENNINGS
M/S: CODE K51
NSWCDD

DAHLGREN VA 22448-
PHONE: (703)663-8157
FAX: (703)663-4568
EMAIL: DJENNIN@RELAY.NSWC.NAVY.MIL

ALLEN JOHNSON
M/S:
RAINBOW SYSTEMS ANALYSIS GRP
8920 BUSINESS PARK DR
AUSTIN TX 78759-
PHONE: (512)346-7999
FAX: (512)794-9997
EMAIL:

NICHOLAS KARANGELEN
M/S:
TRIDENT SYSTEMS INCORPORATED
10201 LEE HWY SUITE 300
FAIRFAX VA 22030-
PHONE: (703)273-1012
FAX: (703)273-6608
EMAIL: NKARANG@NSWC-WO.NAVY.MIL

JEE-IN KIM
M/S:
COMPUTER COMMAND AND CONTROL
2300 CHESTNUT ST STE 230
PHILADELPHIA PA 19103-
PHONE: (215)854-0555
FAX: (215)854-0665
EMAIL: KIM@CCCC.COM

KANE KIM
M/S: DEPARTMENT OF E/CE
UNIVERSITY OF CALIFORNIA

IRVINE CA 92717-
PHONE: (714)856-5542
FAX: (714)856-4076
EMAIL: KANE@BALBOA.ENG.UCI.EDU OR
KANE@ICS.UCI

GARY KOOB
M/S: CODE 3332
OFFICE OF CNR
800 N QUNICY ST
ARLINGTON VA 22217-5660
PHONE: (703)696-0872
FAX:
EMAIL:

NAOUFEL KRAIEM
M/S: MASI/CRI
UNIVERSITY OF PARIS I
17 RUE DE TOLBIAC
PARIS FRANCE 75013-
PHONE: 44.24.93.65
FAX: 45.86.76.66
EMAIL: NAOUFEL@MASI.IBP.FR

BRUCE LABAW
M/S: CODE 5546
NAVAL RESEARCH LABORATORY
4555 OVERLOOK AVE SW
WASHINGTON DC 20375-
PHONE: (202)767-3249
FAX: (202)404-7942
EMAIL: LABAW@ITD.NRL.NAVY.MIL

CARL LANDWEHR
M/S: CODE 5542
NAVAL RESEARCH LABORATORY
4555 OVERLOOK AVE SW
WASHINGTON DC 20375-
PHONE: (202)767-3381
FAX: (202)404-7942
EMAIL: LANDWEHR@ITD.NRL.NAVY.MIL

SI LE
M/S: CODE N51
NSWCDD
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
PHONE: (301)394-1971
FAX: (301)394-4130
EMAIL: SLE@NSWC-WO.NAVY.MIL

KWEI-JAY LIN
M/S: DEPARTMENT OF ECE
UNIVERSITY OF CALIFORNIA

IRVINE CA 92717-
PHONE: (714)856-7839
FAX: (714)725-3203
EMAIL: KLIN@UCI.EDU

JANE W.S. LIU
M/S: DEPARTMENT OF CS
UNIVERSITY OF ILLINOIS
1304 W. SPRINGFIELD AVENUE
URBANA IL 61801-
PHONE: (217)333-0135
FAX: (217)333-3501
EMAIL: JANELIU@CS.UIUC.EDU

EVAN LOCK
M/S:
COMPUTER COMMAND AND CONTROL
2300 CHESTNUT ST STE 230
PHILADELPHIA PA 19103-
PHONE: (215)854-0555
FAX: (215)854-0665
EMAIL: LOCK@CCCC.COM

TERESA LUNT
M/S: EL245
SRI INTERNATIONAL
333 RAVENSWOOD AVE
MENLO PARK CA 94025-
PHONE: (415)859-6106
FAX: (415)859-2844
EMAIL: LUNT@CSI.SRI.COM

W.L. MCCOY
M/S: CODE B10
NSWCDD

DAHLGREN VA 22448-5000
PHONE: (703)663-8367
FAX: (703)663-4568
EMAIL:

CATHERINE MEADOWS
M/S: CODE 5543
NAVAL RESEARCH LABORATORY
4555 OVERLOOK AVE SW
WASHINGTON DC DC 20375-
PHONE: (202)767-3490
FAX: (202)404-7942
EMAIL: MEADOWS@ITD.NRL.NAVY.MIL

JEFFREY MILLER
M/S:
SOHAR INC
133 ROLLINS AVE STE 5B
ROCKVILLE MD 20852-
PHONE: (301)230-5654
FAX: (703)734-6119
EMAIL:

DANIEL MOSTERT
M/S: DEPARTMENT FOR CS
RAND AFRIKAANS UNIVERSITY
P. O. BOX 524
JOHANNESBURG S AFRIC 2000-
PHONE: 27.11.4892847
FAX: 27.11.4892138
EMAIL: BASIE@RKW.RAU.AC.ZA

GILBERT MYERS
M/S: CODE 41
NRAD
271 CATALINA BLVD
SAN DIEGO CA 92152-5000
PHONE: (619)553-4136
FAX: (619)553-
EMAIL: GMYERS@NOSC.MIL

JOHN NALLON
M/S: MS 8404
TEXAS INSTRUMENTS
6500 CHASE OAKS BLVD
PLANO TX 75023-
PHONE: (214)575-3450
FAX: (214)575-5847
EMAIL: NALLON@DDD.ITG.TI.COM

SWAMINATHAN NATARAJAN
M/S: DEPARTMENT OF CS
TEXAS A&M UNIVERSITY

COLLEGE STATION TX 77843-3112
PHONE: (409)845-8287
FAX: (409)847-8578
EMAIL: SWAMI@CS.TAMU.EDU

DAVID OLIVER
M/S:
GE CORPORATE R&D CTR
P.O. BOX 8
SCHENECTADY NY 12301-
PHONE: (518)387-6458
FAX: (518)387-6104
EMAIL: OLIVERDW@CRD.GE.COM

DANIEL ORGAN
M/S: CODE 2151
NUWCDET
BUILDING 80
NEW LONDON CT 06320-
PHONE: (203)440-6546
FAX: (203)440-5243
EMAIL:

DAVID OWENS
M/S:
PARAMAX SYSTEMS CANADA
6111 AVENUE ROYALMOUNT
MONTREAL QE H4P 1K6
PHONE: (514)340-7031
FAX: (514)340-8318
EMAIL:

MOHSEN PAZIRANDEH
M/S:
INNOVATIVE RESEARCH INC.
180 COOK ST STE 315
DENVER CO 80206-
PHONE: (303)321-4955
FAX:
EMAIL: MOHSEN@CS.COLORADO.EDU

DAR-TZEN PENG
M/S:
ALLIED-SIGNAL MTC
9140 OLD ANNAPOLIS RD
COLUMBIA MD 21045-1998
PHONE: (301)964-4195
FAX: (301)992-5813
EMAIL: DTP@BATC.ALLIED.COM

PARAMESWARAN RAMANATHAN
M/S: DEPARTMENT OF E/CE
UNIVERSITY OF WISCONSIN
1415 JOHNSON DR
MADISON WI 53706-1691
PHONE: (608)263-0557
FAX: (608)262-1267
EMAIL: PARMESH@ECE.WISC.EDU

BALA RAMESH
M/S: CODE AS/RA
NAVAL POSTGRADUATE SCHOOL

MONTEREY CA 93943-
PHONE: (408)656-2439
FAX: (408)656-3407
EMAIL: RAMESH@NPS.NAVY.MIL

JOHN REILLY
M/S: CODE K54
NSWCDD

DAHLGREN VA 22448-
PHONE: (703)663-7257
FAX: (703)663-4568
EMAIL:

CHARLES ROBERTSON
M/S: ENGINEERING SUPPORT
AUTOMATED SCIENCES GROUP INC.
P.O. BOX 1750
DAHLGREN VA 22448-
PHONE: (703)663-5231
FAX: (703)663-3717
EMAIL:

ANDRES RUDMIK
M/S:
SPS INC
122 N 4TH AVE
INDIALANTIC FL 32903-
PHONE: (407)984-3370
FAX: (407)728-3957
EMAIL: AXR@SPS.COM

JOHN RUMBUT
M/S: CODE 2222
NUWC
BLDG. 1171-3
NEWPORT RI 02841-2047
PHONE: (401)841-3616
FAX: (401)841-
EMAIL: RUMBUT@ADA.NPT.NUWC.NAVY.MIL

CHARLES SADEK
M/S: CODE B40
NSWCDD
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
PHONE: (301)394-5187
FAX:
EMAIL:

RAJIV SAIN
M/S:
AUTOMATED SCIENCES GRP
16349 DAHLGREN RD
DAHLGREN VA 22448-
PHONE: (703)663-9231
FAX: (703)663-3717
EMAIL:

SAUMYA SANYAL
M/S: M155
FMC
4800 E RIVER RD
MINNEAPOLIS MN 55421-
PHONE: (612)572-7577
FAX: (612)572-4991
EMAIL: SANYALSK@NSD.FMC.COM

RICHARD SCALZO
M/S: CODE A10
NSWCDD
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
PHONE: (301)394-2926
FAX: (301)394-1164
EMAIL: RSCALZO@NSWC-WO.NAVY.MIL

CARL SCHMIEDEKAMP
M/S: CODE 7033
NAWC-AD
P.O. BOX 5152
WARMINSTER PA 18974-0591
PHONE: (215)441-1779
FAX: (215)441-3225
EMAIL: CARLS@NADC.NAVY.MIL

NORMAN SCHNEIDEWIND
M/S: CODE AS/SS
NAVAL POSTGRADUATE SCHOOL

MONTEREY CA 93943-
PHONE: (408)656-2719
FAX: (408)656-3407
EMAIL: SCHNEIDEWIND@NPS.NAVY.MIL

KANG SHIN
M/S: DEPARTMENT OF EE/CS
UNIVERSITY OF MICHIGAN

ANN ARBOR MI 48109-2122
PHONE: (313)763-0391
FAX: (313)763-4617
EMAIL: KGSHIN@EECS.UMICH.EDU

JAMES SMITH
M/S:
OFFICE OF THE CNR
800 N QUNICY ST
ARLINGTON VA 22217-5000
PHONE: (703)696-5752
FAX: (703)696-1330
EMAIL: JGSMITH@ITD.NRL.NAVY.MIL

SANG SON
M/S: DEPARTMENT OF CS
UNIVERSITY OF VIRGINIA
THORNTON HALL
CHARLOTTESVILLE VA 22903-
PHONE: (804)982-2205
FAX: (804)982-2214
EMAIL: SON@VIRGINIA.EDU

ALEXANDER STOYENKO
M/S: DEPARTMENT OF C/IS
NEW JERSEY INSTITUTE OF TECH
UNIVERSITY HTS
NEWARK NJ 07102-
PHONE: (201)596-5765
FAX: (201)596-5777
EMAIL: ALEX@VULCAN.NJIT.EDU

JAY STROSNIDER
M/S: DEPARTMENT OF E/CE
CARNEGIE MELLON UNIVERSITY
5000 FORBES AVE
PITTSBURGH PA 15213-
PHONE: (412)268-6927
FAX: (412)268-3890
EMAIL: JKS@USA.ECE.CMU.EDU

HAROLD SZU
M/S: CODE R44
NSWCDD
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
PHONE: (301)394-3097
FAX: (301)394-3923
EMAIL: HSZU@ULYSSES.NSWC.NAVY.MIL

MARY TAMUCCI
M/S:
MYSTECH ASSOCIATES INC.
5205 LEESBURG PIKE STE 1200
FALLS CHURCH VA 22042-
PHONE: (703)671-8680
FAX: (703)671-8932
EMAIL: TAMUCCI@NUSC.NAVY.MIL

BEVERLEY TANKSLEY
M/S:
MYSTECH ASSOCIATES INC.
5205 LEESBURG PIKE STE 1200
FALLS CHURCH VA 22042-
PHONE: (703)671-8680
FAX: (703)671-8932
EMAIL:

LONNIE WELCH
M/S: DEPARTMENT OF C/IS
NEW JERSEY INSTITUTE OF TECH
UNIVERSITY HTS
NEWARK NJ 07102-
PHONE: (201)596-5683
FAX: (201)596-5777
EMAIL: WELCH@VIENNA.NJIT.EDU

STEPHANIE WHITE
M/S: MS A08-35
GRUMMAN CRC

BETHPAGE NY 11714-3580
PHONE: (516)575-2201
FAX: (516)575-7716
EMAIL: STEPH@GDSTECH.GRUMMAN.COM

JAMES WILLIAMSON
M/S: CODE 703A
NAWC

WARMINSTER PA 18974-
PHONE: (215)441-1564
FAX: (215)441-3225
EMAIL: JAW@NADC.NAVY.MIL

MARK WILSON
M/S: CODE B40
NSWCDD
10901 NEW HAMPSHIRE AVE
SILVER SPRING MD 20903-5640
PHONE: (301)394-5099
FAX: (301)394-1175
EMAIL: MLWILSO@NSWC-WO.NAVY.MIL

DISTRIBUTION

	Copies
DOD ACTIVITIES (CONUS)	
ATTN CODE A-10 (PHILLIP HWANG) DEFENSE MAPPING AGENCY 8613 LEE HIGHWAY FAIRFAX VA 22031-2137	10
DEFENSE TECHNICAL INFORMATION CENTER CAMERON STATION ALEXANDRIA VA 22304-6145	12
ATTN CODE 5543 (KATHERINE MEADOWS) NAVAL RESEARCH LABORATORY 4555 OVERLOOK AVE SW WASHINGTON DC 20375	1
ATTN CODE 4411B (ELIZABETH WALD)	1
CODE 4411C (GRACIE THOMPSON) OFFICE OF NAVAL RESEARCH 800 NORTH QUINCY STREET ARLINGTON VA 22217-5000	1
ATTN CODE E29L COASTAL SYSTEMS STATION DAHLGREN DIVISION NAVAL SURFACE WARFARE CENTER 6703 WEST HIGHWAY 98 PANAMA CITY FL 32407-7001	4

	Copies
NON-DOD ACTIVITIES	
CENTER FOR NAVAL ANALYSES 4401 FORD AVENUE PO BOX 16268 ALEXANDRIA VA 22302-0268	2
ATTN GIFT & EXCHANGE DIVISION LIBRARY OF CONGRESS WASHINGTON DC 20540	4
ATTN GARY BERG-CROSS ADVANCED DECISION SYSTEMS, BOOZ-ALLEN & HAMILTON, INC. SUITE 600 1953 GALLOWS ROAD VIENNA VA 22182	1
ATTN LONNIE WELCH THE REAL-TIME COMPUTING LAB DEPT OF COMPUTER AND INFORMATION SCIENCE NJIT UNIVERSITY HEIGHTS NEWARK NJ 07102	1
ATTN WEI YEH	1
ADRIEN MESKIN ATR 15210 DINO DRIVE BURTONSVILLE MD 20866-1172	1
ATTN GRAEME JONES 27 TYNE STREET BROADGATE PRESTON, LANCASHIRE PRI 8ED ENGLAND	1

DISTRIBUTION (Continued)

	Copies
INTERNAL	
A	1
A44 (R SCALZO)	1
B	1
B02	1
B05 (H CRISP)	5
B10 (S BARKER)	1
B10 (W FARR)	1
B10 (H HUBER)	1
B10 (D PARKS)	1
B20	1
B30	1
B35 (M CHANG)	1
B35 (R HARRISON)	1
B35 (M MASTERS)	1
B42 (J MOSCAR)	1
B44	1
B44 (M EDWARDS)	1
B44 (N HOANG	1
B44 (S HOWELL)	10
B44 (M JENKINS)	1
B44 (C NGUYEN)	1
B44 (H ROTH)	1
B44 (M TRINH)	1
B44 (M WILSON)	1
B44 (C YEH)	1
D	1
D4	1
E231	2
E232	3
E342 (GIDEP)	1
F	1
G	1
K	1
L	1
N	1