

# NAVAL POSTGRADUATE SCHOOL MONTEREY, CALIFORNIA



## THESIS

**REQUIREMENTS ANALYSIS AND DESIGN OF A  
DISTRIBUTED ARCHITECTURE FOR THE  
COMPUTER AIDED PROTOTYPING SYSTEM  
(CAPS)**

by

Gary L. Kreeger

September 1999

Thesis Advisor:

Man-Tak Shing

Approved for public release; distribution is unlimited.

19991126 108

# REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE September 1999	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: REQUIREMENTS ANALYSIS AND DESIGN OF A DISTRIBUTED ARCHITECTURE FOR THE COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)		5. FUNDING NUMBERS	
6. AUTHOR(S) Kreeger, Gary L.		8. PERFORMING ORGANIZATION REPORT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.	
12a. DISTRIBUTION/AVAILABILITY STATEMENT: Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE:	
13. ABSTRACT (maximum 200 words)  <p>The Computer Aided Prototyping System (CAPS) developed at the Naval Postgraduate School is a powerful Computer Aided Software Engineering (CASE) tool for examining requirements and timing constraints for hard real-time systems. However, it remains a stand-alone system. Even if it is running on machines in multiple locations, there is no way to coordinate the efforts between the different locations. In today's software development environment, that proves to be a significant disadvantage. Additionally, providing support for more than just hard real-time software development would tremendously enhance CAPS.</p> <p>Our analysis details the requirements needed to make a distributed CAPS feasible. A distributed CAPS functioning over a network in a coordinated manner would be an invaluable asset to those developing software today, especially in the Department of Defense (DOD). Our work also produced an initial design architecture based on a three tiered client-server model and utilizing Java and the Common Object Request Broker Architecture (CORBA). The Java/CORBA combination greatly simplifies deploying a distributed CAPS over any heterogeneous network. Our preliminary implementation of CAPS with a NT client and a Solaris server demonstrates the efficacy of this design.</p>			
14. SUBJECT Software Engineering, Distributed Applications, Computer Aided Prototyping, CORBA		15. NUMBER OF PAGES 183	
		16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

**REQUIREMENTS ANALYSIS AND DESIGN OF A DISTRIBUTED  
ARCHITECTURE FOR THE COMPUTER AIDED PROTOTYPING  
SYSTEM (CAPS)**

Gary L. Kreeger  
Commander, United States Navy  
B.A., Austin College, 1982

Submitted in partial fulfillment of the  
requirements for the degree of

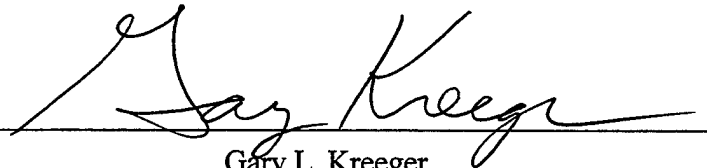
**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

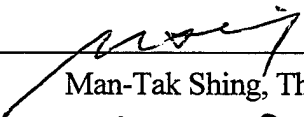
**NAVAL POSTGRADUATE SCHOOL**

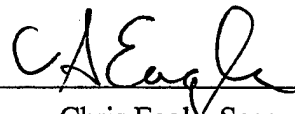
**September 1999**

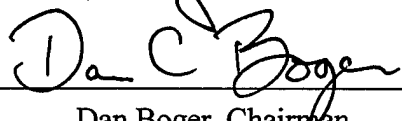
Author:

  
\_\_\_\_\_  
Gary L. Kreeger

Approved by:

  
\_\_\_\_\_  
Man-Tak Shing, Thesis Advisor

  
\_\_\_\_\_  
Chris Eagle, Second Reader

  
\_\_\_\_\_  
Dan Boger, Chairman,  
Computer Science Department

THIS PAGE INTENTIONALLY LEFT BLANK

## ABSTRACT

The Computer Aided Prototyping System (CAPS) developed at the Naval Postgraduate School is a powerful Computer Aided Software Engineering (CASE) tool for examining requirements and timing constraints for hard real-time systems. However, it remains a stand-alone system. Even if it is running on machines in multiple locations, there is no way to coordinate the efforts between the different locations. In today's software development environment, that proves to be a significant disadvantage. Additionally, providing support for more than just hard real-time software development would tremendously enhance CAPS.

Our analysis details the requirements needed to make a distributed CAPS feasible. A distributed CAPS functioning over a network in a coordinated manner would be an invaluable asset to those developing software today, especially in the Department of Defense (DOD). Our work also produced an initial design architecture based on a three tiered client-server model and utilizing Java and the Common Object Request Broker Architecture (CORBA). The Java/CORBA combination greatly simplifies deploying a distributed CAPS over any heterogeneous network. Our preliminary implementation of CAPS with a NT client and a Solaris server demonstrates the efficacy of this design.

THIS PAGE INTENTIONALLY LEFT BLANK

## TABLE OF CONTENTS

<b>I. INTRODUCTION.....</b>	<b>1</b>
A. BACKGROUND.....	1
B. THESIS OBJECTIVES .....	3
1. Requirements for a Distributed CAPS .....	4
2. Design Issues for a Distributed CAPS.....	5
<b>II. BACKGROUND.....</b>	<b>9</b>
A. WHY PROTOTYPE AT ALL? .....	9
1. The Waterfall Method .....	10
2. The Spiral Model .....	13
3. Summary of Prototyping Goals.....	15
B. THE EVOLUTION OF PROTOTYPING SYSTEMS.....	16
1. Automatic Code Generators .....	16
2. Requirements Specification Systems .....	17
3. Executable Specification Language .....	17
4. Megaprogramming Systems.....	18
5. Prototyping Languages .....	18
6. Computer Aided Software Engineering (CASE) Tools .....	19
7. Prototyping Systems Summary .....	20
C. CAPS DEVELOPMENT .....	21
D. PROTOTYPING RISKS .....	22
E. THE FUTURE.....	22
<b>III. REQUIREMENTS ANALYSIS FOR DEPLOYING CAPS IN A DISTRIBUTED ENVIRONMENT.....</b>	<b>25</b>
A. CREATING A WELL DESIGNED SYSTEM.....	25
1. UML Overview.....	25
a. <i>Visualizing</i> .....	26
b. <i>Specifying</i> .....	26
c. <i>Constructing</i> .....	26
d. <i>Documenting</i> .....	27
2. Using UML to Redesign CAPS.....	27
B. METHODOLOGY SPECIFICS .....	28
1. System Functions .....	28
2. Use Cases.....	28
3. Conceptual Model .....	29
C. REQUIREMENTS ANALYSIS RESULTS .....	30
1. Functional Requirements .....	30
2. High-level Use Cases.....	35
3. Expanded Uses Cases.....	41
4. Conceptual Diagrams.....	42



<b>IV. ARCHITECTURAL DESIGN FOR CAPS IN A DISTRIBUTED NETWORK.....</b>	<b>49</b>
A. DEFINING THE CAPS CLIENT-SERVER ARCHITECTURE.....	49
1. A Three Tier Client-Server Design .....	50
2. Component Responsibilities .....	50
B. IMPLEMENTING THE CAPS CLIENT-SERVER ARCHITECTURE.....	51
1. The Selection of Java.....	52
2. The Selection of CORBA .....	52
a. <i>CORBA Description</i> .....	53
b. <i>CORBA Advantages for a Distributed CAPS</i> .....	57
<b>V. A DISTRIBUTED CAPS PROOF OF CONCEPT IMPLEMENTATION .....</b>	<b>61</b>
A. PRODUCT CHOICES FOR IMPLEMENTATION .....	61
B. IMPLEMENTATION DECISIONS.....	62
<b>VI. CONCLUSIONS AND FUTURE WORK .....</b>	<b>67</b>
<b>APPENDIX A: EXPANDED USE CASES.....</b>	<b>71</b>
<b>APPENDIX B: CAPS SERVER IMPLEMENTATION SOURCE CODE .....</b>	<b>93</b>
<b>APPENDIX C: IDL-TO-JAVA COMPILER GENERATED SOURCE CODE ....</b>	<b>105</b>
<b>APPENDIX D: MODIFIED HSI SOURCE CODE .....</b>	<b>125</b>
<b>APPENDIX E: ACRONYMS.....</b>	<b>165</b>
<b>LIST OF REFERENCES .....</b>	<b>167</b>
<b>INITIAL DISTRIBUTION LIST.....</b>	<b>169</b>

## LIST OF FIGURES

Figure 2.1 The Importance of Early Requirement Validation Figure .....	11
Figure 2.2 Types of Errors in Requirements .....	12
Figure 2.3 Spiral Model .....	14
Figure 3.1 Conceptual Diagram - Network Support .....	44
Figure 3.2 Conceptual Diagram - Reuse Support .....	45
Figure 3.3 Conceptual Diagram - Management Support .....	46
Figure 3.4 Conceptual Diagram - Execution .....	47
Figure 3.5 Conceptual Diagram - User Inputs .....	48
Figure 4.1 OMG Reference Model Architecture .....	54
Figure 4.2 CORBA ORB Architecture .....	55

THIS PAGE INTENTIONALLY LEFT BLANK

## LIST OF TABLES

Table 3.1	User Input Functions-Requirements Analysis .....	30
Table 3.2	User Input Functions-System Modeling/Specification .....	31
Table 3.3	User Input Functions-Prototype Development .....	31
Table 3.4	User Input Functions-Reuse .....	32
Table 3.5	System Administration .....	33
Table 3.6	Network Support .....	33
Table 3.7	Project Management .....	34
Table 3.8	Conceptual Diagram Components.....	42

THIS PAGE INTENTIONALLY LEFT BLANK

## ACKNOWLEDGEMENTS

I would like to sincerely thank Professor Man-Tak Shing and LCDR Chris Eagle for all their great ideas and guidance and for always having time to talk. I'd like thank Erik and Lamb for making me laugh when I always needed to. Last, I want to give my deepest appreciation to Katie for her unyielding support which made this thesis possible.

## I. INTRODUCTION

In recent years, more and more attention has been placed on developing better methods for the production of software. This results from the fact that hardware has simultaneously dropped in price dramatically and increased in performance while software has continued to be plagued by cases of bug prone productions, incomplete designs that don't function as desired and in some high profile incidents, software projects that had to be abandoned as hopeless. Often the culprit in these cases is a poor understanding of the user's requirements. This leads to incomplete or erroneous functionality in a software system that, with traditional methods, remains undiscovered until near the end of development when the user sees a working product for the first time. At this point, it is difficult and very expensive to correct requirement deficiencies. This situation led to the concept of prototyping software systems early in development so that requirements could be validated in time to easily make changes before problems became too deeply rooted. The first prototypes were largely coded manually. This made timely analysis difficult. Today, technology has matured to the point that automatic generation of prototypes is not only feasible but practical as well [CHAV95]. As the software development process has become more and more distributed, a need has arisen for distributed prototyping tools.

### A. BACKGROUND

In the traditional waterfall method of software development, requirements analysis and subsequent design were done before little, if any, actually coding was done. This helped to preclude the analysis and design being unduly influenced by

implementation constraints or biases. However, the draw back was that once promulgated, these decisions tended to be seen as set in stone. Even if they were reviewed later in the development cycle, it was difficult to update them. The major drawback though was that the customer only saw their requirements on paper. Possibly they had some conceptual drawings of an interface, but clearly that was not interactive at all. This led to customers who didn't get hands-on experience with their product until some sort of alpha version was developed. At this point the project was nearing completion and making fundamental changes, even simple ones, was very difficult. The result was expensive software that did not perform as the customer actually wanted it to and was probably delivered late. Clearly there had to be a better way to develop software.

That better way is prototyping. The idea is simple but powerful. Take the customer's requirements and create executable models to allow customers to clarify the desired system functionality. The tremendous advantage of this was the customer had a model not drawn on paper but one that could be interacted with on the computer. The developer could use a prototype to help explain the system design to the customer. Customer feedback was immediate and clarification of requirements by the customer much clearer. Since the investment of time and effort to produce a prototype was relatively small and it was early in the development process, changes that the customer wanted were easily accommodated. However, the prototype was still largely a hand-coded product. In order to reach its full potential, prototyping would have to become even more timely.



Automation was the only way for prototyping to become timelier. This necessity led to the development of the Computer Aided Prototyping System (CAPS) at the Naval Postgraduate School [LUQI88a,LUQI96]. The goal of CAPS is to improve the efficiency and accuracy of evolutionary software development by providing tools that make it possible for the developer to quickly and systematically construct and execute prototypes [DAMP92]. Much effort has gone into achieving the goal of automating the prototyping process and the results thus far are impressive. However, the biggest drawback to wider use of CAPS is that it remains a local system. This makes it difficult, if not impossible, to coordinate a large project with many designers. The reality of software production today is that a team that is geographically dispersed will most likely carry out the development of any particular system. Thus to unlock the full potential of CAPS, it is necessary to implement it in a distributed, network environment.

## **B. THESIS OBJECTIVES**

A significant amount of research has gone into developing CAPS. However, today CAPS is implemented on a stand alone UNIX system. This arrangement has served the research done to date quite well. To unlock the potential that exists in CAPS though, it must become more versatile. This means making CAPS available to a wider client base and this requires distributing it over a network. That could be simply a local area network (LAN) but that would only be a half step. It should also be deployable over a wide area network (WAN) since more and more collaborators on projects are located over physically remote sites. Any discussion about a LAN or especially a WAN must first acknowledge the certainty that the network will be heterogeneous. Thus

many of the assumptions made when CAPS was a stand-alone system will no longer be valid.

Any architecture and subsequent implementation of CAPS on a network will have to address many requirements. The high reliability required in a prototyping system such as CAPS will be an even greater challenge in a distributed environment. Isolating errors in a stand-alone system is relatively easy when compared to the same process in a distributed environment. As the distributed environment grows, high availability becomes more important. Since the network is sure to be heterogeneous, the correctness of the system will depend on maintaining the consistency and integrity of the data as it is passed throughout an environment that includes a variety of operating systems, machine hardware and programming languages.

These are only a few of the more obvious obstacles to providing a distributed implementation of CAPS. Some of the problems will have straightforward, readily available answers. But due to the uniqueness of CAPS, many will not. Thus a comprehensive requirements analysis is imperative if CAPS is to be successfully deployed in a distributed environment.

### **1. Requirements for a Distributed CAPS**

With CAPS operating on a stand-alone UNIX system, it is possible to make many simplifying assumptions. Consequently, much of the current design will have to be reexamined. However, distributing CAPS is more complicated than simply taking the current system and putting it on a network. Before creating a design and architecture, the requirements themselves must be reevaluated. It is possible, indeed almost certain, that adapting CAPS to a distributed environment will alter many of the requirements that

led to the creation of CAPS. Below are characteristics that need to be included in the requirements analysis for distributing CAPS.

What will be the nature of the actual physical environment? The degree of transparency must be decided. It is assumed that any distributed network will be heterogeneous but that doesn't mean that it will necessarily support any hardware or operating system that is connected to the network. Exactly what will be supported must be determined. Will it be installed on a LAN, WAN or some other network design? The amount of fault tolerance must be considered. System performance encompasses several issues. What is the minimum system hardware requirement to be enforced? The amount of latency that can be tolerated will affect the design and must be determined. Is scalability important to a distributed CAPS? It may be possible to decide that there are practical limits to the size of any useful implementation. As with any distributed system, security becomes a much bigger concern. Certainly cost is a consideration if you want to widely deploy any system, as is ease of use. Ease of use will also include questions about the effectiveness of training. A key requirement will be the constraints placed on what language reusable components can be written in. The healthy population of the database of reusable components is critical to CAPS reaching its full potential, thus the impediments to achieving that goal should be as few as possible.

## **2. Design Issues for a Distributed CAPS**

The nature of the distributed architecture will determine many of the design specifics. What type of architecture is best for CAPS? The architecture can be centralized, replicated throughout the network, or something in between. One of the

most fundamental design questions deals with how to handle components. This is much more complicated and less straightforward than in an undistributed system. How different components communicate with each other must be decided and will be affected by where they are located. Closely associated with the question of communication is what degree of mobility components will have and how they will be synchronized. Components are not the only entities that must specify their behavior in a distributed environment. Data itself is severely influenced by the architectural design. What sort of access will users have to data in a distributed CAPS? It can be shared, distributed or some form of shared-distributed [PROT96]. The details of how persistent storage will operate must be consistent with the handling of data.

There is a collection of other design questions to be answered in addition to how components and data will behave. The mechanism for creation and subsequent destruction of process and threads must be specified. This must accommodate not only a distributed environment but the possibility of multi-processor machines within the network. With processes and data distributed throughout a network, provisions must be made for handling distributed exceptions. At the management level, distributed CAPS must provide for version control and merging of distributed data. The strategy for dealing with both of these functions will be critical to successfully distributing CAPS. It will be futile to take advantage of the improved productivity offered by distributing CAPS if a coherent, correct prototype is not the ultimate product.

A final consideration throughout the design process will be how or whether to reuse parts of the existing CAPS system. Clearly, some of it must be radically reengineered to support operations in a distributed network. However, much of the

current system may be useable in a distributed environment. Deciding what to keep unchanged and what to alter will be a constant balancing act. Any design must weigh the performance improvement associated with any change to the existing system against the cost of implementing that change.

THIS PAGE INTENTIONALLY LEFT BLANK

## **II. BACKGROUND**

The history of software prototyping shares many similarities with the larger subject of software engineering, and indeed with most emerging technologies. It has been the source of great debate and confusion as software engineers sought to define exactly what it was and how best to use it. Indeed, the debate and research continues unabated into what exactly constitutes the best way to prototype and how to use it most effectively. The one point that most software engineers agree upon is that the larger, increasingly complex systems of today can be more efficiently produced with the aid of prototypes. Thus, before looking too far into the future it would be valuable to take a look at the past and how we got to where we are today in working with software prototypes.

### **A. WHY PROTOTYPE AT ALL?**

As computer languages came into existence, a rather simple method of developing software emerged. It was simply code and fix. A programmer would sit down and write a program, run it and then try to fix the things that didn't work. This approach was sufficient for small, relatively uncomplicated programs. But just as structured programming emerged to bring some cohesion to writing computer programs and in turn was replaced by the object oriented paradigm, the code and fix method of developing software systems was destined to fall by the way side. As systems became more complicated and ever larger, one person was no longer able to do the amount of work required. The amount of work was even more than small groups of software engineers and programmers could handle. As more people became involved in the

process, it became clear that a better process was needed to create software systems. This led to the waterfall model of software development.

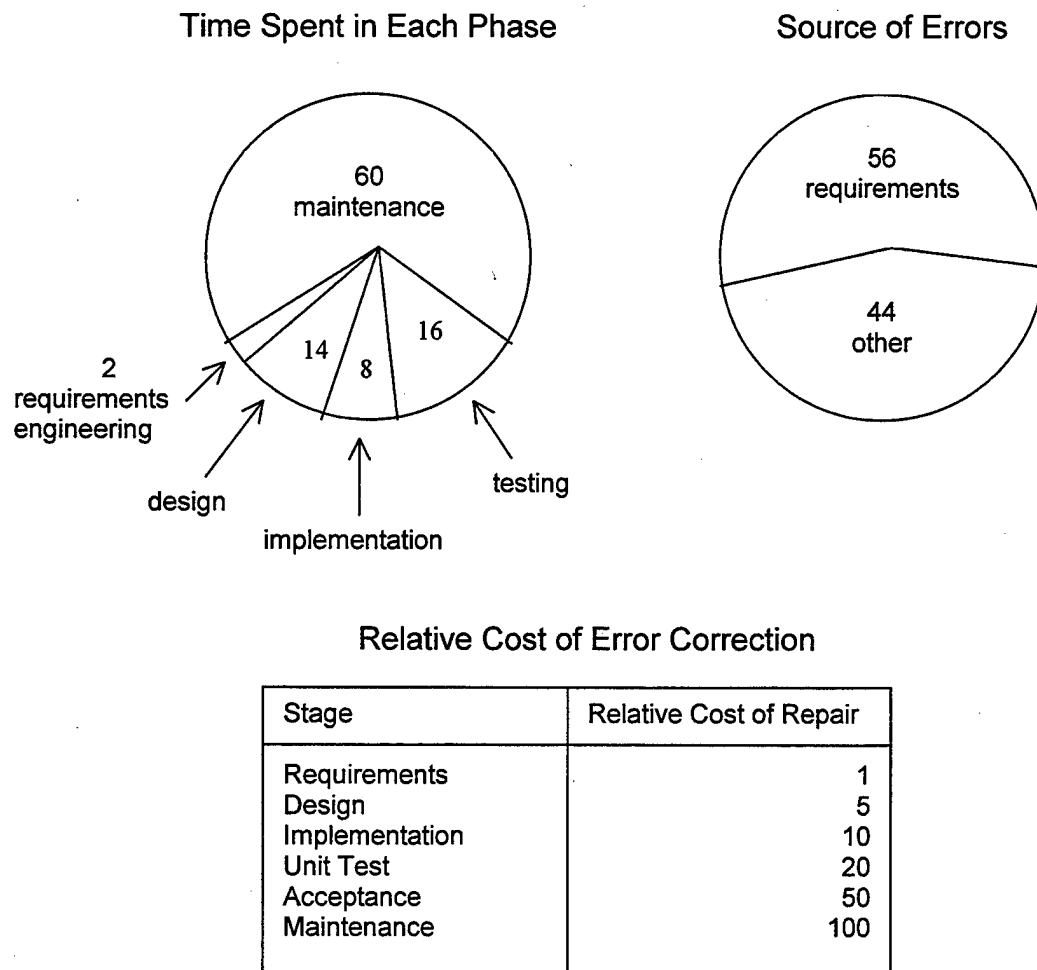
### **1. The Waterfall Method**

The waterfall model, defined by Royce [ROYC76] and refined by Boehm [BOEH76], introduced some rigor and discipline to the development process. No longer was code written in isolation from the bigger picture of what the system should actually do and then tested to see where it didn't work. Instead, the process was partitioned into well-defined development phases. Now there would be a comprehensive investigation to determine the requirements of the system, followed by design, implementation and finally testing before a system was released. In theory, this systematization of the process of producing software systems would result in improved productivity and better systems. To a large degree it succeeded in achieving the desired results. However, as systems continued to swell in size and complexity, it became clear that there existed some serious shortcomings with the waterfall method as well.

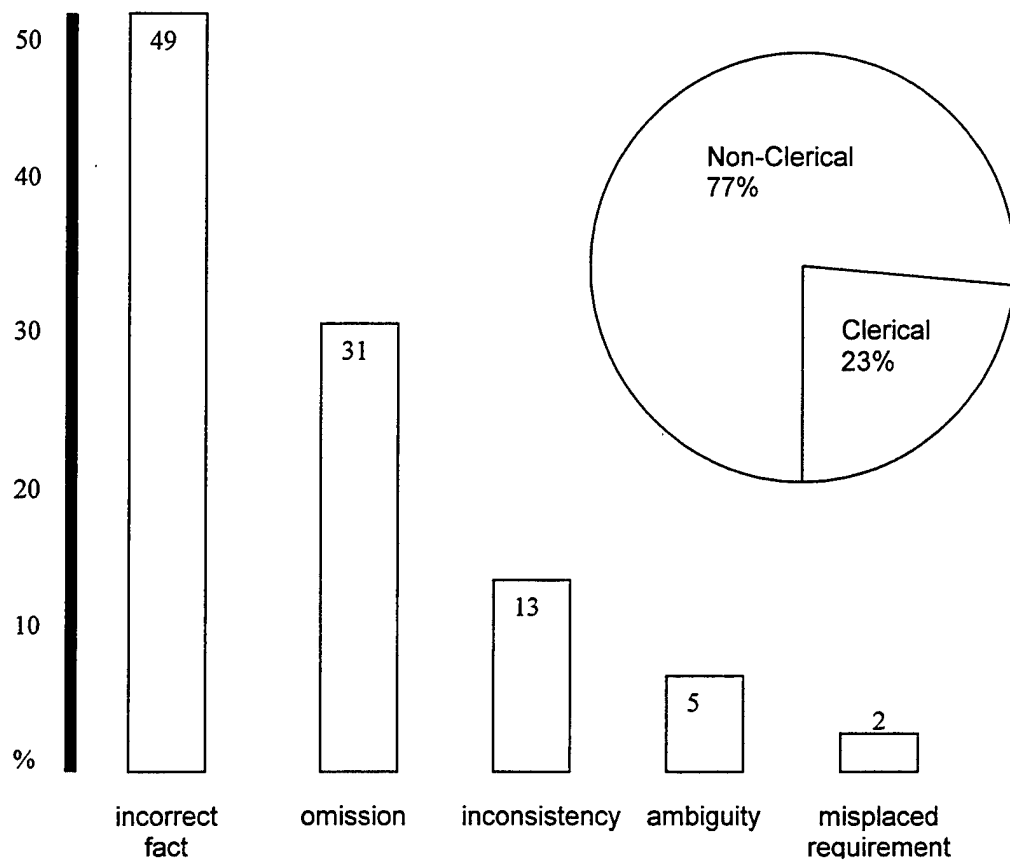
Foremost amongst these is the need to establish the requirements for the systems at the beginning of the development process. This is true for several reasons. Customers may not know exactly what they want the final system to really do or they may not clearly define their needs to the developers. Also, requirements can change due to alterations in the environment that the system will operate in, even while the system is being developed. Also, during system development, requirements can emerge that were overlooked or unanticipated initially. Whatever the reasons for changing, incomplete or missing requirements, one thing was for certain. Inaccurate requirements, left unchecked, propagated throughout the entire system and resulted in software systems that were late, over budget and performed poorly. Sometimes entire



projects were abandoned as beyond salvaging, as it became clear that they would never perform as expected. As seen in Figures 2.1 and 2.2, most of the problems experienced by systems result from inadequacies with the requirements and the later these deficiencies are corrected, the more expensive the fix becomes.



**Figure 2.1 The Importance of Early Requirement Validation From [WOOD92]**



**Figure 2.2 Types of Errors in Requirements From [WOOD92]**

The dilemma confronted by software developers with the waterfall model is aptly summed up in a report of the Defense Science Board Task Force on Military Software [DSB87].

We believe that users cannot, with any amount of effort and wisdom, accurately describe the operational requirements for a substantial software system without testing by real operators in an operational environment, and iteration on the specification. The systems built today are just too complex for the mind of man to foresee all the ramifications purely by the exercise of the analytic imagination.

This inability to completely anticipate all the requirements of system being developed lead to the creation of the spiral model [BOEH86] for software engineering.

## **2. The Spiral Model**

The spiral model, shown in Figure 2.3, moved the software development process from being documentation driven to one that is risk driven. In other words, instead of concentrating on completing the current phase in the development process before moving on to the next phase, you start by identifying the areas with the highest risk and as you develop a strategy for dealing with each one you then tackle the next lower area of risk. In essence you have a series of mini-waterfall evolutions but the power of the spiral method is that the requirements do not have to all be known before you start. Instead, you are defining them as you "spiral" through iteration after iteration, solving the hardest part of the system first and working your way to the areas of relatively less risk. Prototyping certainly is not required to execute the spiral model of development but it can dramatically improve the efficiency of the model. If the problem domain of the system being developed is low risk and well understood, it is possible to use what essentially amounts to a waterfall method of development. However, this is usually true only in uninteresting or trivial systems. In most cases a system is being created for an area that is not well understood, either by user or developers or maybe even both. Especially in the Department of Defense, this is often the case since many of the systems being developed are at the cutting edge of technology or beyond.

This is when the power of prototyping becomes clear. By allowing developers to present a mock-up of a system early in development, it is possible to get user feedback in a much more timely manner. This capability makes the spiral method of development

much more effective since developers can now produce prototypes throughout the iterative development process and use the responses from the ultimate end user to continually refine the requirements. Thus when a final product is delivered, it is much more likely that the system will function as expected. This avoids a great deal of maintenance effort and expense that normally goes into modifying a delivered system in order to get it to do what the user really wanted in the first place.

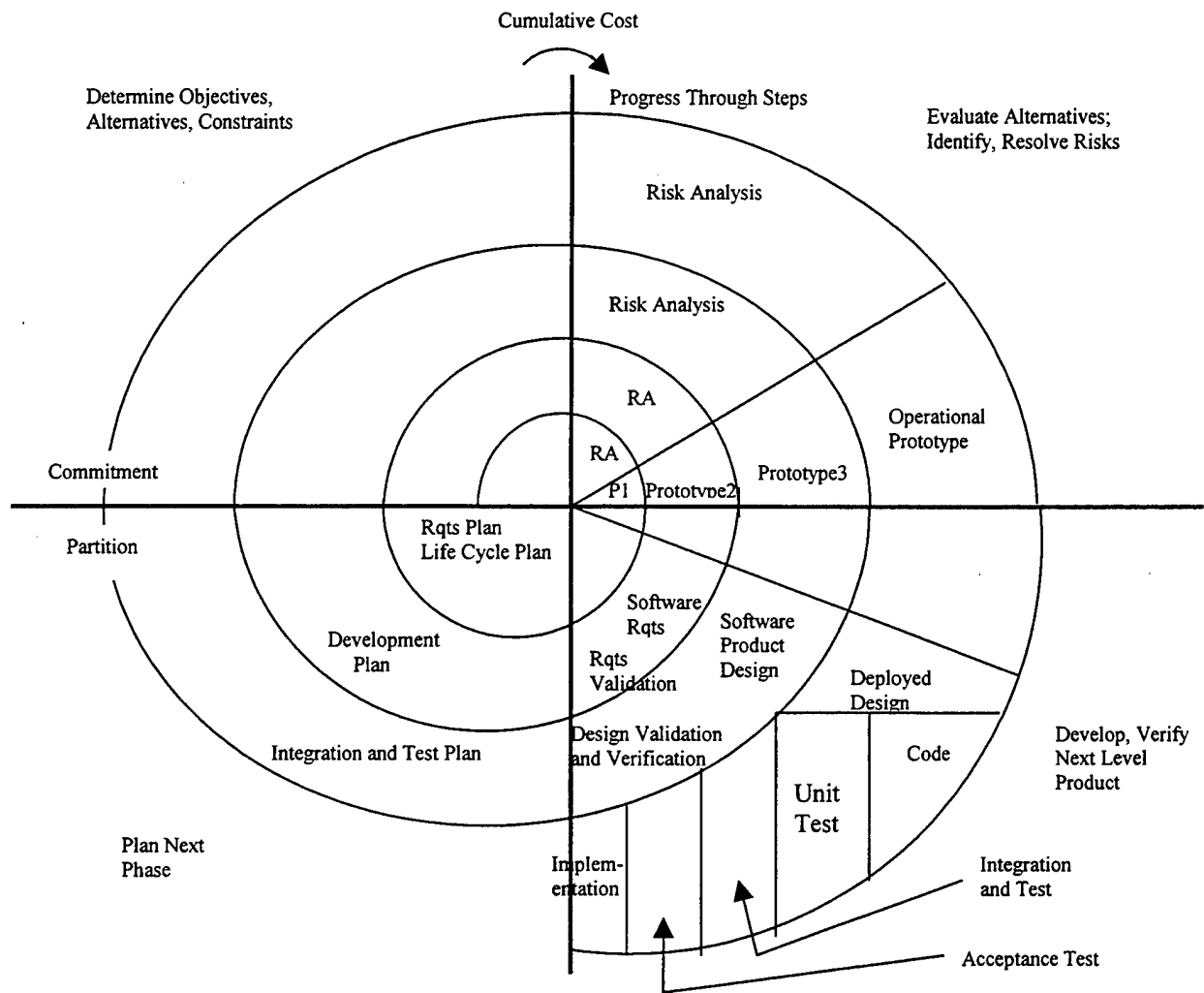


Figure 2.3 Spiral Model From [BOEH86]

This leads to consideration about what the ultimate role of the prototype is within the development process. While there are many variations and differing names, there are two basic purposes for prototypes, both within and outside the spiral model. Prototypes can be either exploratory or implementation oriented. Exploratory, also referred to as experimental or evaluative, prototypes are used to better understand some specific requirement or limitation. The intention is that they will be used and discarded after providing the required information and insight. Implementation prototypes on the other hand will ultimately become the actual system being developed. The biggest practical difference is simply how they are created. An exploratory prototype which will be "thrown away" after being used does not have to concern itself too much with things such as efficiency, robustness, readability, etc. It must only function in a manner that allows the developers, and possibly the users, to better understand the question under consideration. The implementation oriented prototype must perform in such a manner that the code it produces is of production quality.

### **3. Summary of Prototyping Goals**

No matter which approach to prototyping a development team uses, there are some general goals for prototyping that pertain to any software system development effort. Not every goal is necessarily applicable to every prototype, but they provide a framework for how prototyping can improve any development effort.

- Help systems engineers, users and developers to better understand and communicate about the environment and specific problem requirements and to transfer design intent amongst themselves;

- Demonstrate what is actually feasible, a specific system behavior or some element of the proposed system's performance;
- Use as a mechanism for getting the users involved earlier than in traditional software development processes thus potentially creating a more useful system; and
- Get a production version, of improved quality, completed in less time than with traditional methods without prototyping.

## **B. THE EVOLUTION OF PROTOTYPING SYSTEMS**

Exactly what constitutes a prototype and how you create one has been the subject of much debate and research. While there will probably never be one definitive prototype model that everyone accepts, it is illuminating to review a sampling of the systems that have been fielded. Not every system evaluated meets the definition of a prototype but they all have contributed to the evolution of prototyping systems.

### **1. Automatic Code Generators**

One of the earliest attempts at improving the software development process was the idea of having program source code automatically generated. Any productive and useful prototyping system today will have to incorporate some automatic code generation or it will be too cumbersome to be effective. The success of current prototyping systems owes much to the work done on automatic code generation. However, automatic code generation by itself as a stand-alone development method has many shortcomings. Assuming that you are confident that the code being generated is actually correct, there is the question of what it was derived from. Often, automatic code generation amounts to not much more than instantiating a template

according to the user's specifications. The better code generation systems such as Graphical Approach to Modeling and Building Interactively a Technical System (GAMBITS) [POWE96] are still restricted to a fairly narrow domain. They start small using rigid models and keep extending the systems being developed until a useful system is produced. However, useful prototypes are not part of the process and it is still up to the developers, users and system analysts to ensure that correct requirements are implemented. Therefore they are limited outside their respective problem domain.

## **2. Requirements Specification Systems**

The goal of a requirements specification system is to capture the complete requirements for a system. Often poor communications result in missed, inaccurate or incomplete requirements and thus the specifications on which implementation will ultimately depend are flawed. A requirements specification system will not produce a prototype but the research done in this area has improved current prototyping systems as well as automatic code generating systems and executable specification languages.

## **3. Executable Specification Language**

Executable specification languages, such as PAISLey [ZAVE91] and the Jackson Development System (JSD) [ROLL91], represent the operational approach to software development. The operational approach incorporates three phases into the development process - modeling, specification and implementation. It closely resembles the automatic code development paradigm and when done iteratively also resembles the spiral method of development. The modeling though is not a mock-up or prototype of some part of a system. Rather it is a model of the relationships that will exist when the system "operates." Thus they suffer from the same shortcomings as

automatic code generating systems. Namely, the difficult task of correctly specifying requirements still exists and the system produced will be limited to a relatively narrow problem domain such as a data intensive business application or database interactions. The operational approach to software development is not adequate for the prototyping of more general behavior.

#### **4. Megaprogramming Systems**

Megaprogramming is the attempt to develop systems from existing modules. Two examples are the Common Prototyping Language (CPL), a Defense Advanced Research Project Agency initiative that became Prototyping Technology (Prototech) [KIMB92] and the earlier Rapid Prototyping to Investigate End-user Requirements (RaPIER) [WELC86]. Megaprogramming is therefore a more sophisticated form of code reuse. As such, it can be an effective way to better explore a relatively well documented problem domain or one for which work is already underway. However, it is limited as a tool for probing new requirements though. It is problematic to write the code for the modules to be combined without first having a good understanding of the domain and the requirements of the proposed system.

#### **5. Prototyping Languages**

As prototyping research gained momentum, it became clear to some that the ideal prototyping system would have at its core a language specifically developed for the task. Adapting existing languages invariably lead to shortcomings and compromises in the functionality of the prototyping system. PROTEUS [MILL90] and Durra [BARB91] were two attempts to develop a prototyping language. PROTEUS is intended for prototyping parallel and distributed environments. It contains set



theoretical notation with a small set of mechanisms for controlling parallel execution. It relies on a shared variable model of concurrency and focuses on regulating synchronization and communication. Durra is a task description language for developing distributed applications. It constructs a program by having the user specify the distributed application structure and the resources allocated to each component. The developer must still code each task and process but templates exist for creating the necessary interfaces.

The limitations on languages such as PROTEUS and Durra are that they are designed for relatively narrow domains and they cannot function without some sort of support system. There is no user interface, such as a graphical user interface (GUI), to aid developers. You cannot simply load them into your computer and start designing prototypes. So while the idea of developing languages specifically for prototyping is important for any viable prototyping system, it is still just one of several elements needed.

## **6. Computer Aided Software Engineering (CASE) Tools**

The natural result of all the research into different aspects of prototyping was the creation of CASE tools that combined many of the disparate areas of research into one synergistic mechanism for producing prototypes. CAPS and Proto [ACOS94] are two such systems. Where CAPS focuses on applications in the hard real time problem domain, Proto operates on distributed, parallel systems. Proto provides, as does CAPS, a design environment for the system analyst using the System Specification and Design Language (SSDL). The design environment includes execution support, a GUI and software reuse in order to make it a comprehensive, efficient tool for developing

prototypes. This total integration of specialized prototyping language and tailored support environment is necessary to unlock the full potential that prototyping offers to software development.

There have been attempts to create prototyping systems without the overhead of creating the specialized, integrated system described above. They attempt to model prototypes using tools based on such things as Petri nets or relational databases. However, these tools are cumbersome and ultimately limited in their application.

## **7. Prototyping Systems Summary**

The above discussion of the history of prototyping systems is not comprehensive but simply representative of the different areas of research that have contributed to the considerable body of knowledge concerning software development with prototypes. As with any technology, software development is an ever-changing entity. Currently, the iterative approach of the spiral method of software development promises the best chance of creating software systems that actually satisfy the requirements specified by the user. The key to making it effective for anything except trivial systems is the ability to rapidly produce prototypes to examine proposed system behavior. Many of the research efforts involving prototyping have contributed to the field and then fallen into disuse as their productiveness reached practical limits. CAPS has weathered the turbulence that accompanies any emerging technology and today is not only alive but a vital, leading research tool in creating prototypes for supporting software system development.

### C. CAPS DEVELOPMENT

As the importance of prototyping to determining system behavior during development became clear, it also became obvious that to be effective prototypes would have to be created quickly. The requirement for a means to rapidly prototype leads to the seemingly obvious conclusion that a computer aided tool needed to be fabricated for this effort to succeed. At the time, the most successful systems for automatically generating code were focused in narrow problem domains and relied upon templates or generic solutions. In order to achieve the type of powerful, automated prototyping system envisioned, a computer language would have to be design specifically for this purpose. That language was Prototype System Description Language (PSDL) [LUQ188b]. The objective of PSDL was to mechanically create documents that could be processed and executed at the specification level. In other words, PSDL was designed to be an executable prototyping language at the specification or design level.

As CAPS is built around PSDL, PSDL provides more than simply an executable prototyping language. Queries to search the database repository of reusable software components are based on PSDL. CAPS incorporates a GUI with which the system designer can graphically represent the desired requirements. CAPS will transform this graphical description into PSDL. Once in PSDL, the Ada components can be generated and bound together. After compilation, CAPS provides the facilities for graphically displaying the prototype. This allows system behavior to be evaluated and the necessary changes made to requirements in a timely manner. Thus, CAPS supports an iterative approach to software system development.

## **D. PROTOTYPING RISKS**

It would be unfair and unrealistic to think that prototyping does not incur some risks. It is certainly not the fabled silver bullet but simply a means for increasing the efficiency and accuracy of the system development process. It is important to understand the shortcomings of prototyping in order to take full advantage of its tremendous potential. Some of the risks associated with prototyping are:

- overly enthusiastic developers who want to continually iterate and evolve a prototype by adding additional functionality beyond that which was originally being investigated;
- classification of a prototyping effort as rapid can lead managers to think that this area can easily absorb budget cuts and shortcuts without affecting the product;
- the temptation to substitute a prototype that appears to work for a fully documented and tested final product; and
- a misunderstanding about the purpose of prototyping in the development process by users or customers can lead to acrimony with the development team and compromise the close working relationship that produces the best results.

## **E. THE FUTURE**

CAPS today is an energetic, productive research effort into the use of rapid prototyping for software development. Its contributions are many and continue to appear at an impressive rate. Distributing CAPS itself certainly will ensure its lasting viability. Perhaps, however, even more is needed if CAPS is to remain at the leading edge of this research field and eventually become a mature product widely used throughout commercial software development. Focusing only on the domain of hard

real time applications is certainly becoming a limitation. The not too distant future, actually beginning to appear already, will see the emergence of applications that combine the requirements of hard real time constraints with concurrent operation across distributed networks that are growing seemingly without bound. Failure to support such applications risks being left on the sidelines or marginalized into some software development niche.

THIS PAGE INTENTIONALLY LEFT BLANK

### **III. REQUIREMENTS ANALYSIS FOR DEPLOYING CAPS IN A DISTRIBUTED ENVIRONMENT**

Fielding a sound, usable system doesn't occur by happenstance or even simply hard work. It is the result of a great deal of careful analysis and planning long before any implementation ever begins. If the requirements are not properly detailed, you can easily end up with a system that does something very well but does not do what the customer wanted. If the architecture is not precise, the development can spin out of control during implementation as each ambiguity results in a workaround. These workarounds and individual interpretations of what the designers meant compound on each other until system cohesion begins to suffer. The goal of this chapter is to fabricate the foundation of a clear and logical requirements analysis.

#### **A. CREATING A WELL DESIGNED SYSTEM**

The goal of any system design should be the production of a system that is effective, efficient, robust and maintainable. Currently, the best way to achieve that goal is through rigorously applying the techniques of object-oriented analysis and design. Modeling is the underpinning that makes object-oriented analysis and design successful. Through modeling, everyone involved in the development process, from stakeholders to testers to managers, can come to a clear understanding and agreement about how the system will ultimately perform. Today, the state of the practice for modeling is the Unified Modeling Language (UML).

##### **1. UML Overview**

UML was designed to be as flexible as possible. In fact, it can be applied to almost any process, not just software design. It is meant to simply be a tool in the

software engineers toolkit. However, it can be a very powerful tool. Utilized to its fullest, UML can be integrated into a project from the outset while searching for the complete requirements and used through designing the architecture, implementing the desired system and finally testing the production code. It is a language that focuses on the conceptual and physical nature of a system. Thus it can be used to build the blueprints for a software system. Specifically, UML is designed for the visualizing, specifying, constructing and documenting of software artifacts [BOOC99].

**a. *Visualizing***

UML helps a system developer record and present to others the ideas he/she has. This can be a tremendously powerful way of fostering communication between the stakeholders, developers, project managers, users and any other interested party.

**b. *Specifying***

UML capitalizes on the improved communications by allowing the developer, using his/her enhanced understanding of the system objectives, to build models that are precise, unambiguous and complete.

**c. *Constructing***

The developer can directly connect the models to various programming languages. Thus, it is possible to generate programming source code in languages such as Ada, C++ and Java.



#### ***d. Documenting***

Many artifacts will be produced during a development cycle while using UML. These can form the basis of a thorough, persistent documentation effort.

#### **2. Using UML to Redesign CAPS**

As stated before, UML is only a tool in the software design process. It is not a set-in-stone, explicitly detailed process. Rather, it is up to the developer to decide how to best employ UML. UML gives you the structural pieces for creating a solid design but there are many ways to actually put the pieces together. The nature of UML though lends itself to best supporting a spiral or incremental development approach.

We view this work as the beginning effort in redesigning CAPS for the future. Our goal is to first establish the requirements for an improved, distributed CAPS. The requirements are just a description of the needs and desires for what a system should do when deployed. Although this is a simple idea, it is critically important. If the requirements are not clearly and accurately defined, all that comes afterwards in the development process can be easily compromised. We have chosen to work with the UML model in this effort because we believe it can be easily extended in follow on theses.

Our methodology in defining the requirements for the next CAPS will be to first determine the necessary system functions and attributes. Next, we will promote a better understanding of the processes involved with the application of use cases. Finally, we will complete the analysis with a proposed conceptual diagram. We strongly

believe that this will be the key to a successful design and implementation phase in the future.

## **B. METHODOLOGY SPECIFICS**

The methodology we used for determining the requirements is detailed below.

### **1. System Functions**

Simply stated, the system functions are what the system should actually do in the real world. System attributes, on the other hand, are non-functional system qualities. System functions can be placed in one of two categories - evident or hidden. The user is aware when evident functions are being performed. The performance of hidden functions is not visible to the user. Often, underlying technical services, such as saving information to a persistent storage mechanism, are hidden.

Constraints and details influence system attributes. These also fall into two categories - must and want. In reality, if a constraint is truly real, it should be classified as only must. Thus it is only attribute details that are characterized as must or want.

### **2. Use Cases**

Simply put, use cases are a narrative description of domain processes that highlight the interactions between the system being designed and external agents. External agents can be such things as people, other systems, computers or processes. According to the creators of the UML, a use case is a description of a set of sequence actions that a system performs that yields an observable result to a particular actor. A use case is used to structure the behavioral things in a model. [BOOC99]

Capturing the intended behavior of the system being designed is the goal of a use case. This facilitates easier, more complete communication end users, domain experts and developers. Use cases help draw attention to high risk areas sooner by

fostering clearer investigations of domain processes. Any discussion of implementation should be avoided in use cases. The goal is to discuss what should be done, not how to do it. However, use cases can be utilized to corroborate about a proposed architecture. Two types of use case will be of interest in reengineering CAPS. First, high-level use cases will demonstrate all the interactions between actors and the system in very general terms. It provides a broad-brush overview of what the system should accomplish. The more interesting or critical use cases will be further detailed in expanded use cases. Finally, a use case diagram will link the narrative use cases to the UML.

Use cases can help validate the rules of interactions between actors and actors or actors and the system. Exploration of the variations on scenarios can result from use case discussions. Often, nontrivial scenarios will have alternate paths of action revealed during such discussions and possibly one of them will be an improvement on the original idea. The crucial characteristic is work should always be accomplished. In other words, something should be done that's of value to an actor.

Lastly, use cases can form the basis for developing test plans. As use cases are modified, the changes needed in the test plan are clearly established. A use case gives the test developer a straightforward, plain language narrative of what is supposed to happen. As use cases can be easily extended and more detailed throughout the development process, they make it possible to easily support a spiral or incremental development methodology.

### **3. Conceptual Model**

The conceptual model illustrates meaningful concepts, i.e. ,objects, in a problem domain. Decomposition by concepts supports object oriented analysis whereas

structural analysis is decomposition by processes or functions. Developing a comprehensive set of objects is the key to successful object oriented analysis. A static structures diagram expresses the conceptual model in UML. The concepts are identified and their attributes delineated as well as their associations with other concepts. There are no methods listed though. This construction serves to emphasize the fact that concepts in the conceptual model are representing real world entities and not software components. The concepts will be derived from the use cases and functional requirements.

**C. REQUIREMENTS ANALYSIS RESULTS**

The results of the initial requirements analysis for reengineering CAPS are detailed below.

**1. Functional Requirements**

The functional requirements for a reengineered CAPS are listed in Tables 3.1 - 3.7 grouped by various user inputs, system management, network support and project management

**Table 3.1 User Input Functions-Requirements Analysis**

Ref #	Function	Function Category	Attribute	Details and Constraints	Detail Category
R1.1	Record stakeholder comments	evident	interface metaphor	form driven text format	must
R1.2	Record user comments	evident	interface metaphor	form driven text format	must
R1.3	Record requirements for the system being developed	evident	interface metaphor	form driven text format	must

**Table 3.2 User Input Functions-System Modeling/Specification**

Ref #	Function	Function Category	Attribute	Details and Constraints	Detail Category
R1.4	Model hard real time systems	evident	interface metaphor	graphical user interface for flow diagram and form metaphor input for hard real time constraints	must
R1.5	Model distributed systems	evident	interface metaphor	graphical user interface for flow diagram and form metaphor input for distributed system constraints	must
R1.6	Model concurrent systems	evident	interface metaphor	graphical user interface for flow diagram and form metaphor input for concurrent system constraints	must
R1.7	Model any combination of hard real time, concurrent or distributed system	evident	interface metaphor	graphical user interface for flow diagram and form metaphor input for hard real time, concurrent and/or distributed system constraints	must

**Table 3.3 User Input Functions-Prototype Development**

Ref #	Function	Function Category	Attribute	Details and Constraints	Detail Category
R1.8	Translate PSDL code into 3 <sup>rd</sup> generation object oriented language source code	evident	response time	source code will be generated in less than 2 minutes	want
R1.9	Edit prototype source code	evident	response time	source code will be available in less than 10 seconds	want
R1.10	Compile software modules in a prototype	evident	response time	object code will be generated in less than 2 minutes	want
R1.11	Link compiled modules in a prototype	evident	response time	linking will be completed in less than 2 minutes	want
R1.12	Execute prototype	evident	platform	prototype must be able to execute on UNIX and Windows95/98/NT	must

**Table 3.3 User Input Functions-Prototype Development (continued)**

R1.13	Statically check hard real-time timing constraints for system prototype	evident	response time	complete timing validation in less than 1 minute	want
R1.14	Dynamically check hard real-time timing constraints for system prototype	evident	fault tolerance	identify violations of timing constraints while executing	must
R1.15	Dynamically report network collisions for distributed system prototype	evident	fault tolerance	identify details of simulated network collisions while executing	must
R1.16	Dynamically record synchronization details for system prototype	evident	fault tolerance	identify details of simulated network timing while executing	must
R1.17	Save component to local persistent storage	hidden	response time	save should execute in less than 1 minute	want
R1.18	Save project to local persistent storage	hidden	response time	save should execute in less than 1 minute	want
R1.19	Commit component to persistent storage	hidden	response time	commit should execute in less than 1 minute	want
R1.20	Commit project to persistent storage	hidden	response time	commit should execute in less than 1 minute	want

**Table 3.4 User Input Functions-Reuse**

Ref #	Function	Function Category	Attribute	Details and Constraints	Detail Category
R1.21	Select software module from reuse database	evident	shared component access	provide matches in less than 1 minutes	want
R1.22	Save software modules to reuse database	hidden	shared component access	provide Software Librarian controls for reuse database	must
R1.23	Delete software modules from reuse database	hidden	shared component access	provide Software Librarian controls for reuse database	must

**Table 3.5 System Administration**

Ref #	Function	Function Category	Attribute	Details and Constraints	Detail Category
R2.1	Allow Software Librarian to add new users to access list for software component database	hidden	scalability	allow enough users to support very large system development	must
R2.2	Allow Software Librarian to delete new users from access list for software component database	hidden	scalability	allow enough users to support very large system development	must
R2.3	Allow project manager to grant file access to specific users	hidden	security	only allow authorized user to access a project	must
R2.4	Allow project manager to revoke file access to specific users	hidden	security	only allow authorized user to access a project	must
R2.5	Recover from a local system failure	hidden	fault tolerance	Restore system back to a safe state after a local failure	must
R2.6	Recover from a global application failure	hidden	fault tolerance	Restore system back to a safe state after system wide application failure	must
R2.7	Recover from a local application failure	hidden	fault tolerance	Restore system back to a safe state after a local application failure	must

**Table 3.6 Network Support**

Ref #	Function	Function Category	Attribute	Details and Constraints	Detail Category
R3.1	Transmit module to a remote site	hidden	shared component access	securely get work from one site to another site	must
R3.2	Download module from a remote site	hidden	shared component access	securely get work from one site to another site	must
R3.3	Detect errors in transmission of modules	hidden	fault tolerance	ensure integrity of modules moving over network	must
R3.4	Generate alert for detected errors in transmission	hidden	fault tolerance	ensure integrity of modules moving over network	must

**Table 3.6 Network Support (continued)**

R3.5	Recover from a global system failure	hidden	fault tolerance	Restore system back to a safe state after system wide failure	must
R3.6	Acknowledge error-free transmission of module	Hidden	fault tolerance	ensure integrity of modules moving over network	must

**Table 3.7 Project Management**

Ref #	Function	Function Category	Attribute	Details and Constraints	Detail Category
R4.1	User must log in securely in order to use system	evident	response time	less than 10 seconds	want
R4.2	Create new project	evident	response time	less than 5 seconds	want
R4.3	Load existing project	evident	response time	less than 10 seconds	want
R4.4	Record version for all software artifacts	hidden	ease of use	system must track and record order in which artifacts entered	must
R4.5	Add component to project	evident	response time	component should included in project in less than 1 minute	want
R4.6	Merge components from different sources into one component	evident	ease of use	system must automatically merge components, tracking versions	must
R4.7	Merge components from different sources into one project	evident	ease of use	system must automatically merge components, tracking versions	must
R4.8	Alert project manager to conflicts during merge operations	evident	version control	system must identify version conflicts and inform the project manager	must
			response time	Project manager should be alerted in less than 2 minutes	want



## 2. High-level Use Cases

The high-level use case are detailed below.

Use Case: U1. Start-up  
Actors: System Administrator  
Type: primary  
Overview: System Administrator logs onto host system and initiates CAPS.  
Cross Reference: Functions: R2.5, R2.6, R2.7, R3.5

Use Case: U2. Log in  
Actors: User  
Type: primary  
Overview: After CAPS is initiated, a user must log into CAPS in order to determine what project and reuse database privileges the user owns.  
Cross Reference: Functions: R2.3, R4.1

Use Case: U3. Open project  
Actors: User  
Type: primary  
Overview: After logging into CAPS, user can either open a new project or select an existing project and open it, if he has the proper authorization from the project manager.  
Cross Reference: Functions: R2.3, R4.2, R4.3

**Use Case:** U4. Modify prototype

**Actors:** User

**Type:** primary

**Overview:** After opening a project, the user will be able to make changes to the graphical interface of the control flow diagram or to the text boxes associated with either objects or data streams in the diagram. These changes will be automatically reflected in the PSDL code. The user can record user comments, stakeholder comments and requirements for the system being prototyped. Additionally, the user can directly access the PSDL code and change it manually.

**Cross Reference:** Functions: R1.1, R1.2, R1.3, R1.4, R1.5, R1.6, R1.7, R1.21, R4.4, R4.5

**Use Case:** U5. Retrieve component from reuse database

**Actors:** User

**Type:** primary

**Overview:** The user accesses the reuse database and inputs search parameters. CAPS responds with no match, unique match or a list of possible matches. CAPS generates an alert if there is an error in transmission and an acknowledgement if the transfer is successful.

**Cross Reference:** Functions: R1.21, R3.2, R3.4, R3.6

Use Case: U6. Save  
Actors: User  
Type: primary  
Overview: User has opened a project or component and now saves it to local persistent storage.  
Cross Reference: Functions: R1.17, R1.18

Use Case: U7. Modify GUI for displaying prototype execution  
Actors: User  
Type: primary  
Overview: User selects and then edits the files that provide functionality for displaying prototype execution.  
Cross Reference: Functions: R1.4, R1.5, R1.6, R1.7, R1.8, R1.9

Use Case: U8. Generate executable prototype  
Actors: User  
Type: primary  
Overview: The user translates the prototype in order to create source code in an implementation language from the PSDL and link it. The user will then verify through CAPS that all static hard real time constraints are met. The user will then compile the prototype source code.  
Cross Reference: Functions: R1.9, R1.10, R1.11

Use Case: U9. Execute the prototype

Actors: User

Type: primary

Overview: The user will execute the prototype. As the user test the prototype functionality, CAPS will verify that all dynamic hard real time, concurrency and network constraints are met. CAPS will allow the user to record stakeholder and/or user comments.

Cross Reference: Functions: R1.1, R1.2, R1.3, R1.12, R1.13, R1.14, R1.15, R1.16

Use Case: U10. Manage project changes

Actors: User, Project Manager

Type: primary

Overview: User (possibly more than one) will submit module(s) to the Project Manager for incorporation into the project prototype. The user may elect to return the entire prototype after making changes to only some of the modules. In this case the Project Manager must identify which modules are changed or new. If the module does not already exist in the project database, the Project Manager will merge the submitted module(s) into one module, resolving any conflicts. CAPS will assign and track a version number and insert the module into the project database, making ties to all other modules that may exist in the project database. If the module

previously existed in the project database, the Project Manager will merge all submitted modules with the existing one, resolving any conflicts. CAPS will update the version number. If an existing module is submitted to the Project Manager, CAPS will verify that all other modules that it depends on are the most up to date version. CAPS will generate an alert if there is an error in transmitting a module and an acknowledgement if there is no error.

Cross Reference: Functions: R1.19, R1.20, R3.1, R3.3, R3.4, R3.6, R4.3, R4.4, R4.6, R4.7, R4.8

Use Case: U11. Manage reuse database changes

Actors: User, Software Librarian

Type: primary

Overview: User submits a new module, or an existing one that was modified, to the Software Librarian. When the Software Librarian accepts the module for inclusion in the reuse database, a version control number is assigned or updated as necessary and the module is saved to the reuse library. If the module was an update of an existing one, all users of the old version are alerted. The Software Librarian can also delete modules.

Cross Reference: Functions: R1.22, R1.23, R3.1, R3.3, R3.4, R3.6

Use Case: U12. Add user to project access  
Actors: Project Manager  
Type: secondary  
Overview: The Project Manager grants a user access to a specific project.  
Cross Reference: Functions:R2.3

Use Case: U13. Delete user from project access  
Actors: Project Manager  
Type: secondary  
Overview: The Project Manager deletes a user's access to a specific project.  
Cross Reference: Functions: R2.4

Use Case: U14. Add user to the reuse database  
Actors: Software Librarian  
Type: secondary  
Overview: Upon receiving request, Software Librarian will add user to the reuse database access list.  
Cross Reference: Functions: R2.1

Use Case: U15. Delete user from the reuse database  
Actors: Software Librarian  
Type: secondary

Overview: Upon receiving request, Software Librarian will delete user from the reuse database access list.

Cross Reference: Functions: R2.2

Use Case: U16. Exit project

Actors: User

Type: primary

Overview: The user exits the current project. If unsaved data exist, CAPS will ask the user if they want to save the data before exiting the project.

Cross Reference: Functions: R1.18. R1.20

Use Case: U17. Log off

Actors: User

Type: primary

Overview: User request to log off from the current session of CAPS. If unsaved data exist, CAPS will ask the user if they want to save the data before logging the user off CAPS and quitting the current session of CAPS.

Cross Reference: Functions: R1.18, R1.20

### **3. Expanded Uses Cases**

The expanded use cases are detailed in Appendix A. They are: U3. Open project; U4. Modify prototype; U5. Retrieve component from reuse database; U7. Modify GUI for displaying prototype execution; U8. Generate executable prototype;

U9. Execute the prototype; U10. Manage project changes; U11. Manage reuse database changes.

#### 4. Conceptual Diagrams

The list of components used in the conceptual diagrams is shown below in Table 3.8 and the detailed subsections are shown in figures 3.1 through 3.5. The complete descriptions of the relations and concepts are included in the detailed views.

**Table 3.8 Conceptual Diagram Components**

Access List
Alert
CAPS
CAPS Graphical Interface
CAPS Menu
Comment
Compiler
Component
Constraint
Editor
Error
Executable Code
GUI Builder
Host System
List of Possible Matches
Network
Persistent Storage
Project
Project Database
Project Manager
Prototype
Prototype GUI
PSDL Code
PSDL Data Stream
PSDL Diagram
PSDL Object
Query
Remote Site
Requirement



**Table 3.8 Conceptual Diagram Components (continued)**

Reuse Library
Reuse Library Search
Scheduler
Search Parameters
Software Librarian
Source Code
Stakeholder
System being Designed
System Parameters
Text Box
Translator
User
Work Station

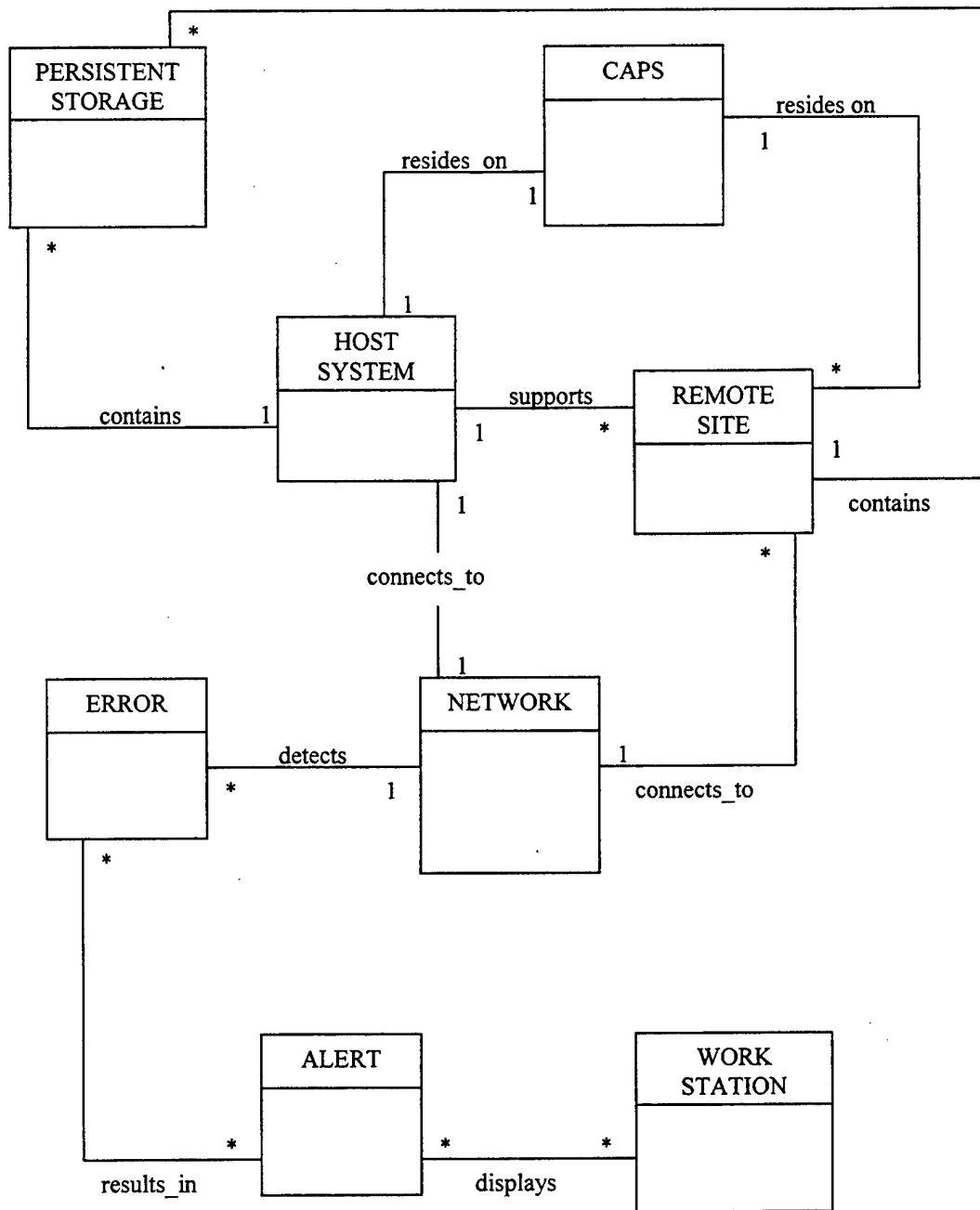


Figure 3.1 Conceptual Diagram - Network Support

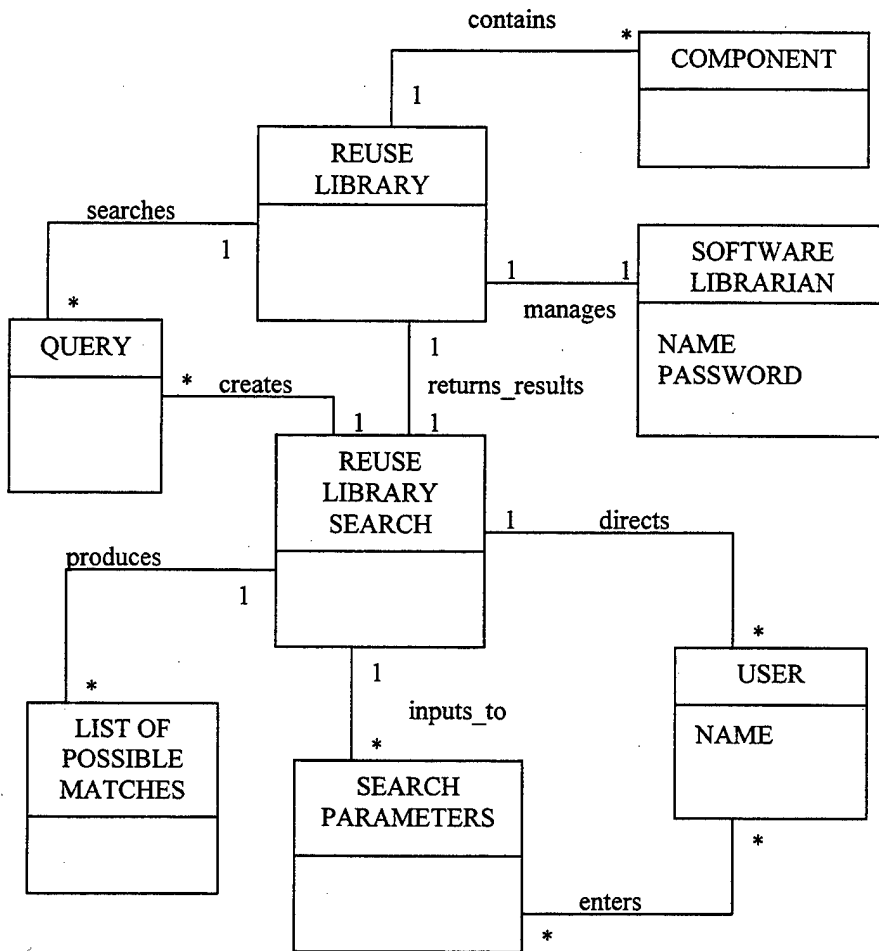
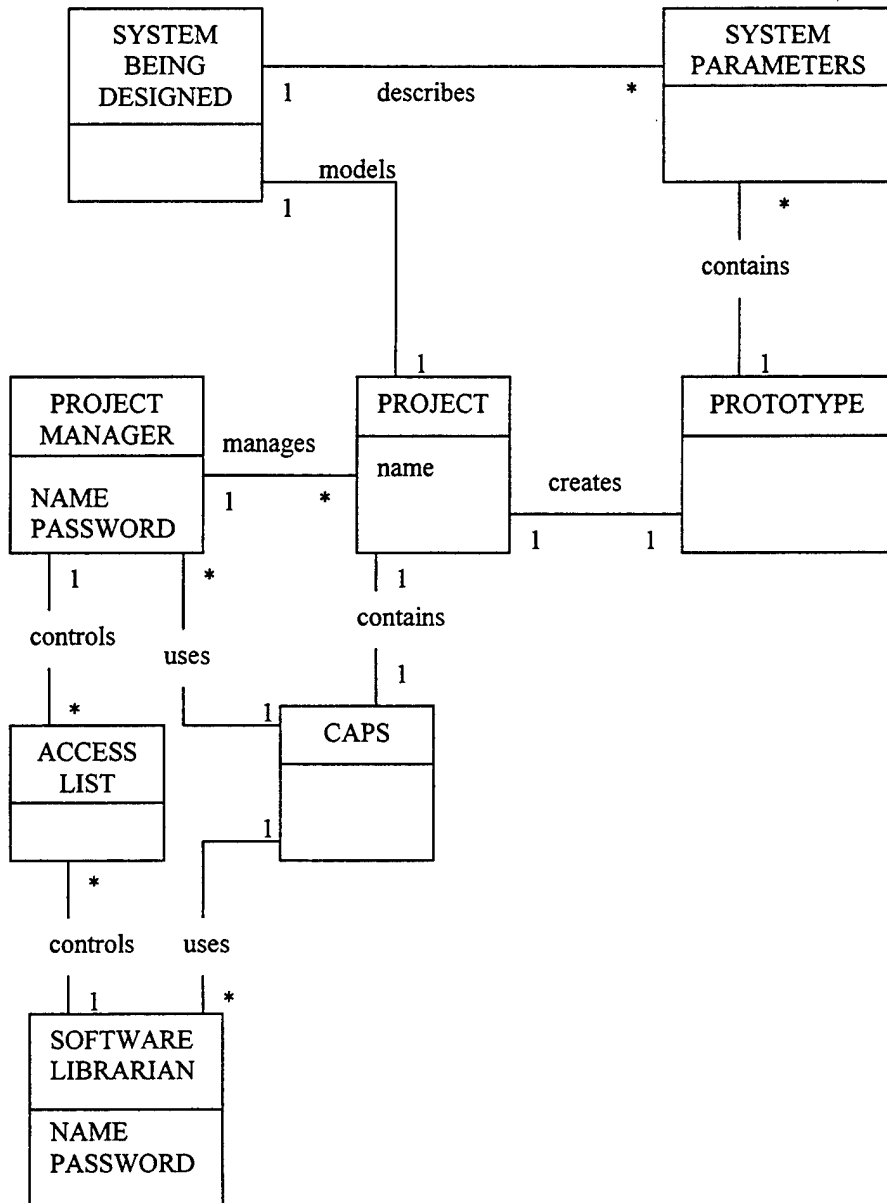


Figure 3.2 Conceptual Diagram - Reuse Support



**Figure 3.3 Conceptual Diagram - Management Support**

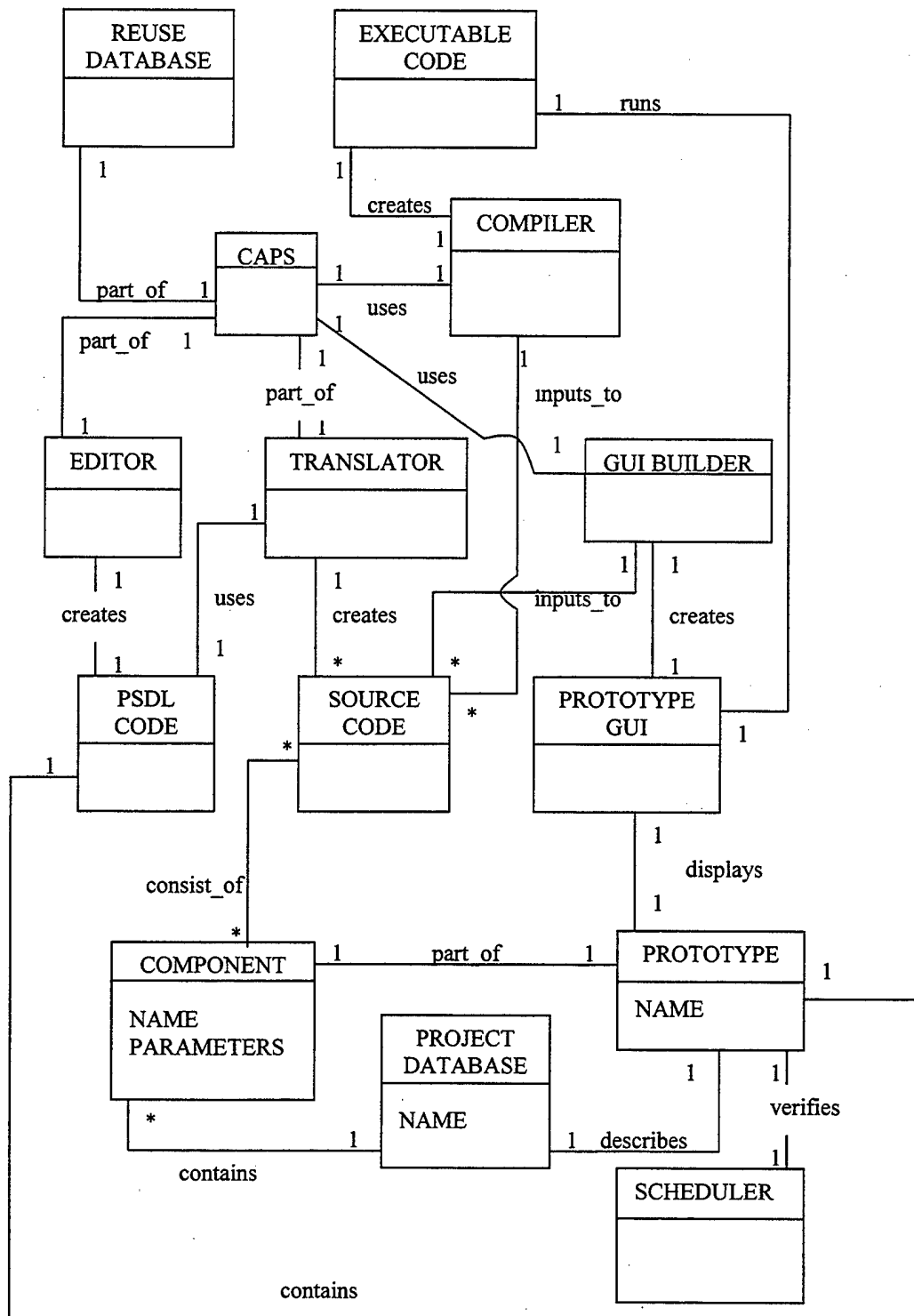


Figure 3.4 Conceptual Diagram - Execution

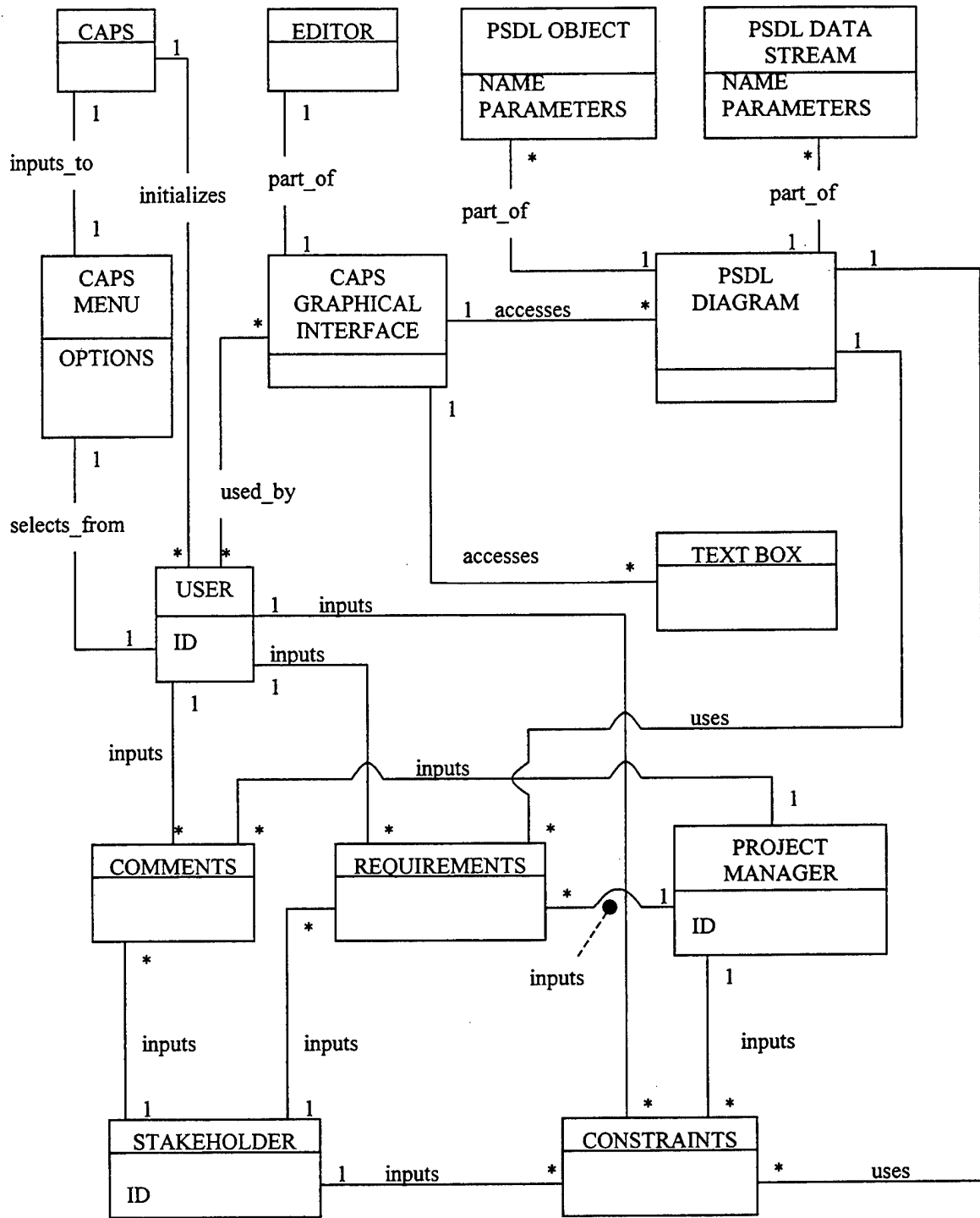


Figure 3.5 Conceptual Diagram - User Inputs

#### **IV. ARCHITECTURAL DESIGN FOR CAPS IN A DISTRIBUTED NETWORK**

The choices for how to design the architecture for a distributed CAPS fall along an unbroken continuum. At one end you have dumb client machines and a super smart server that does all the work and returns the results to the clients for display to the user. On the other end of the spectrum you have a set up that resembles most Local Area Networks (LAN) where the server simply provides a service, such as file downloads, and the client machines do all the work. The first is highly centralized and the second highly decentralized. We believe that the most optimum solution is a client-server architecture that falls in-between the extremes of this continuum. The rapid growth in the amount of bandwidth available on all kinds of networks, including the Internet, make sophisticated client-server architectures very feasible.

##### **A. DEFINING THE CAPS CLIENT-SERVER ARCHITECTURE**

Any design must take advantage of the increasing power of today's desktop personal computers (PC). By moving a significant load from the server to the client machines, you can dramatically improve scalability by reducing the amount of work the server must do and the quantity of data it must send to client machines. However, the implementation of this client-server architecture should not be limited to just one type of PC. Indeed, client machines do not necessarily have to even be a PC. They could, for instance, be a UNIX workstation. This diversity reflects the reality of most networks today. They are becoming more and more heterogeneous. The internet is the ultimate example of this heterogeneity but by no means the only example. The Heterogeneous Systems Integrator (HSI), which contains a PSDL graphical editor, written by Ilker Duranlioglu [DURA99] provides an important first step in creating a powerful CAPS

client that any truly successful distributed CAPS design will require. The HSI is an all Java implementation.

### **1. A Three Tier Client-Server Design**

A robust three tier design is the preferred way to implement a distributed CAPS in the client-server model. The three tier design offers several advantages over the older two tier model. First, it more closely resembles the object oriented paradigm practiced today. You can encapsulate the functionality of the client-server design easily in a three tiered model and as long as the interfaces between each tier remain unchanged, you can update the separate parts independently. The three tiers in a distributed CAPS are the client side, represented by the HSI, the communication server that communicates with multiple clients over a network and the 'back end' programs that do the actual work requested by the clients on the computational server. Additionally, by carefully allocating responsibilities in the design you can noticeably improve performance and latency by ensuring good load balancing. Thus, no particular node becomes a bottleneck.

### **2. Component Responsibilities**

One of the first questions to resolve in designing a distributed system, is who will be responsible for maintaining state information. This can be quite complicated and introduce large inefficiencies if it is done at one central location, i.e., on the server side of the network. Thus, we decided that the best approach was for each client to maintain their own state information. Fortunately, this does not introduce very much additional complexity to the HSI already written. The HSI already manages threads that are created by opening prototypes and editors. It is a relatively simple matter for the HSI to include state information whenever it invokes services from the server. This allows the



server to focus on facilitating requests from various clients. If the server asynchronously passes client requests off to 'back end' for processing, then it can continue to efficiently service clients. While the local clients manage their own states, one centrally located repository greatly simplifies configuration management and version control. Local clients can open prototypes from either local memory or from the server side. They can also save to either location. However, the project manager controls centrally what changes get into the master copy of a prototype. Thus when various clients submit changes to a prototype, they are saved into the user's centrally located folder and the project manager decides when and how to access them. This arrangement allows the project manager to easily maintain control over the merging process and the resulting configuration and versioning decisions. Users can only read the master prototype, and only when the project manager permits it.

## **B. IMPLEMENTING THE CAPS CLIENT-SERVER ARCHITECTURE**

There are many ways in which a distributed CAPS could be implemented. The first decision is what language, or languages, to use. The original CAPS is written largely in Ada, C and C++. Given the realities of writing distributed applications, among other considerations, we decided that Java would be a superior language with which to implement a distributed CAPS. Choosing an implementation language is only part of the total implementation decision. The other part is determining how the communications will actually take place in a distributed system. For this, we decided that the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) offered the best possible solution. Our reasons for these selections are detailed below.

## **1. The Selection of Java**

There are many reasons why Java is the language of choice when implementing a distributed application. First, the ease of migrating Java across different machines and operating systems is an enormous advantage. As discussed previously, it is almost a certainty that any distributed system deployed will by necessity be heterogeneous. Every major type of computer and operating system today has a Java Virtual Machine (JVM) written for it. This allows Java byte code written on one type of machine to execute on any machine that has a JVM. This was shown to work while implementing our proof of concept demonstration. Byte code that was created in a Windows NT PC environment was copied to a Sun Solaris Unix machine and successfully executed without any alterations. This type of portability will greatly simplify implementing a distributed CAPS. Secondly, Java was created as a language to operate on the Internet and World Wide Web (WWW). Thus it possesses many built-in capabilities that facilitate network operations. Additionally, Java has facilities for things such as multi-threading, garbage collection and error management designed into the language that appreciably reduce the difficulty of designing an application. Lastly, as will be discussed, Java is a natural fit with CORBA and the two are quickly merging in the world of network applications.

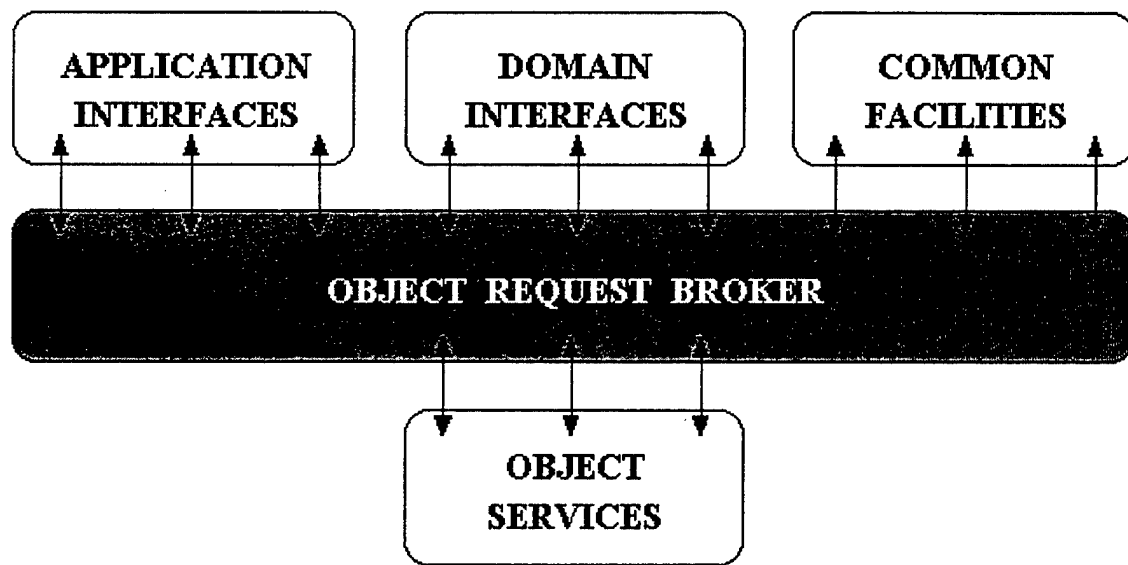
## **2. The Selection of CORBA**

CORBA provides many advantages over other competing middleware solutions. A review of CORBA will facilitate a discussion of these advantages.

a. ***CORBA Description***

As previously discussed, it is assumed that a distributed CAPS must perform in a heterogeneous environment. While heterogeneity in itself is not negative and may, in fact, be viewed as an asset to be leveraged, it does present challenges to us as software developers desiring to implement an application in a heterogeneous networked system. Heterogeneity creates the need for middleware that can enable the sharing of objects, functions and types without causing extensive software re-work for developers, or complex work-arounds for users.

The Object Management Group (OMG) was formed in 1989 to develop, adopt, and promote standards for the development and deployment of applications in distributed heterogeneous environments. Since that time, the OMG has grown to be the largest software consortium in the world, and has developed the Object Management Architecture (OMA). The OMA consists of an Object Model and a Reference Model. The Object Model defines how objects can be described, and the Reference Model, shown in Figure 4.1, deals with interactions between those objects. In the Object Model, clients issue requests for services to objects (much like a remote procedure call (RPC)). The implementations of these objects are hidden from the client. A key component of the Reference Model is the Object Request Broker (ORB), which facilitates communication between clients and objects. CORBA is the specification developed by the OMG that details the interfaces and characteristics of the ORB. In CORBA the terms "client" and "server" are not rigidly defined roles. The CAPS server could handle the request from client machines and in turn become a client itself when it invokes requests on some 'back end' implementation.



**Figure 4.1 OMG Reference Model Architecture From [SCHM99]**

In CORBA, an application consists of one or more objects that may reside on the same or different platforms. An object provides service(s) that can be “requested” by a client. Clients obtain services from an object by making “requests” that consist of an operation, the name of the object that will respond, zero or more parameters, and an optional request context. The object may or may not return results to a client, and will return an exception if an abnormal condition occurs. Object implementations may be written in a variety of languages and may exist in a variety of forms. Essentially, CORBA allows components to discover each other and interoperate on an object bus. However, CORBA does much more than just create a simple object bus, as shown in Figure 4.2. It also provides an extensive set of services for such things as creating and deleting objects, accessing them by name, putting them in persistent storage, externalizing their states and forming ad hoc relationships between objects. Thus you can design an ordinary object and then give it the specific

characteristics needed for a distributed application by using CORBA's multiple inheritance. [ORFA98]

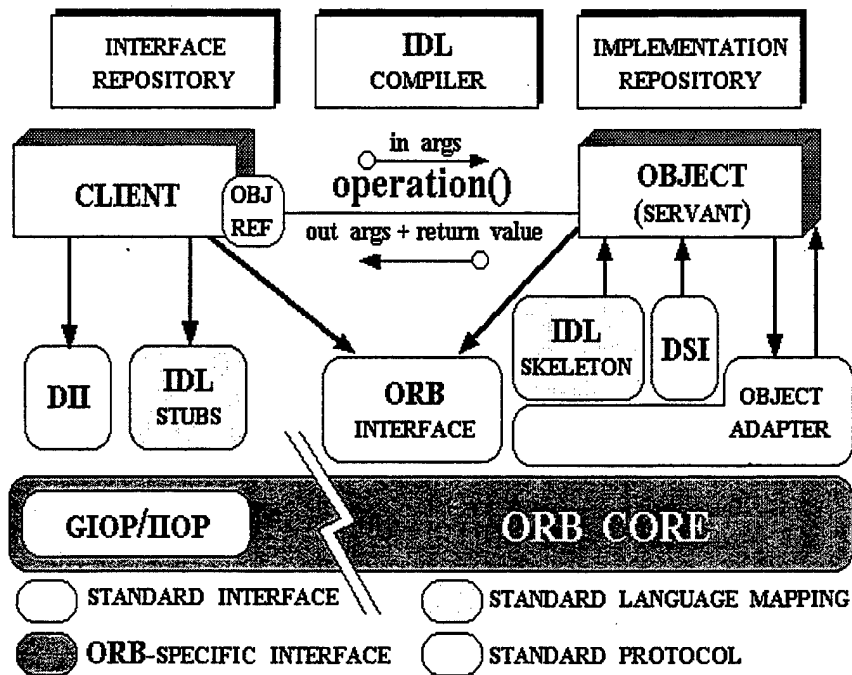


Figure 4.2 CORBA ORB Architecture From [SCHM99]

Methods can be invoked statically or dynamically. In Static Invocation, a client's request is made via interface definition language (IDL) "stubs" on the client side, and the response is handled by IDL "skeletons" on the object side. The stubs and skeletons interface with the CORBA ORB. In Static Invocation, the IDL Client Stub converts data from the client's local data representation (type) to the Common Data Representation (CDR), which is platform and language independent. On the object's platform, the Object Skeleton executes the reverse operation. In Dynamic Invocation, requests are made via Dynamic Invocation Interface. With Dynamic Invocation the developer is afforded more flexibility. In Dynamic Invocation, the Dynamic Skeleton

Interface (DSI) may take the place of the Static Invocation Object Skeleton to accomplish data conversion at run time.

CORBA supports two types of (method) invocation semantics: synchronous invocation and asynchronous invocation. Synchronous invocation is blocking. The client will invoke the method and block until it receives a response from the server (object implementation). Asynchronous invocation is non-blocking. The client will invoke a method, continue its computation, and collect results as they arrive. With non-blocking primitives, the SEND primitive returns control to the requesting program as soon as the message is copied from the user buffer to the kernel buffer. The corresponding object that executes the RECEIVE primitive signals its intention to receive a message, provides a buffer into which the message will be copied and continues to execute. [POPE98]

Through the IDL stubs, a client can use RPC-style semantics (synchronous), or by using Dynamic Invocation Interface (DII) a client can use SEND/RECEIVE semantics. Using DII allows a client to directly access the underlying request mechanisms provided by the ORB. Applications use DII to dynamically issue requests to objects without requiring IDL stubs to be linked in. The DII allows clients to make non-blocking "deferred synchronous" (separate SEND and RECEIVE operations) and one way (SEND only) calls. [POPE98]

Many industry leaders, including IBM, Novell, Borland/Visigenic, SunSoft, Netscape, Oracle and JavaSoft just to name a few, have recognized the importance of CORBA middleware in realizing the potential of heterogeneous, distributed systems[ORFA98]. In fact, CORBA is now part of the Defense Information

Infrastructure Common Operating Environment (DII COE) standard web browser, and is finding increasing use in Department of Defense applications. The utility of CORBA lies in its ability to integrate diverse applications across a variety of networks and network protocols. CORBA's language independent IDL's allow objects to be used from a variety of programming languages, including COBOL, C, C++, Ada, Smalltalk, Perl and Java. CORBA-based applications are independent of network protocols so they may be run in a distributed system over a diverse network. These attributes ensure CORBA's tremendous usefulness in a heterogeneous environment.

**b. *CORBA Advantages for a Distributed CAPS***

CORBA certainly is not the only solution for implementing an application in a distributed environment. The other prominent options include legacy solutions that predate the ORB concept such as Java sockets, Common Gateway Interface (CGI) scripts with Hypertext Transfer Protocol (HTTP) and Java Servlets. Non-CORBA ORBs are JavaSoft's Remote Method Invocations (RMI) and Microsoft's Distributed Component Object Model (DCOM). As for simple performance metrics, the fastest implementation for operating over a network is a socket using a buffered data stream. However, as we'll discuss below, this is not a realistic choice for implementing a distributed CAPS. The three ORBs, CORBA, DCOM and RMI, are very close in performance and as a group are a little less than twice as slow as a socket. Servlets are well over an order of magnitude slower than a socket and the CGI/HTTP combination is well over two orders of magnitude slower. [ORFA98]

The three legacy solutions all suffer from the fact that they operate at very low levels of abstractions. Sockets in particular are a relatively primitive model.

Sockets are 'close to the wire' in the sense that they are the lowest level of abstraction available. This accounts for their efficient performance but it also makes them difficult to program. The other solutions build upon sockets as a transport mechanism while shielding users from having to deal with the details of socket programming. CGI scripts over HTTP is the current predominant model for three tiered applications over the internet. HTTP provides simpler semantics on top of sockets which CGI scripts use to communicate between clients, servers and 'back end' resources. However, besides being incredibly slow, other disadvantages include a lack of typed parameter support and object reference persistence, a relatively low level of abstraction and poor scalability. A Servlet is a small piece of Java code loaded onto a server. It operates much as CGI but it overcomes some of the worst performance degradations by remaining in memory between request and by staying connected to 'back end' resources. As with CGI though, the lack of typed interfaces result in a proliferation of interfaces as the number of methods grows and each method must be prepared to marshal and unmarshal multiple data types.

RMI and DCOM share many of the advantages of CORBA for developing distributed applications. There are however, some significant drawbacks to both. RMI and DCOM both lack the comprehensive services and facilities that are available for CORBA. These include a Naming Service, Event Service, Property Service, Relationship Service, Lifecycle Service, Security Service and a continually growing host of CORBA facilities such as mobile agents, data interchange, workflow, firewalls and business object frameworks. The ultimate goal of CORBA facilities is to provide an IDL interface for virtually every networked service. RMI is an all Java implementation, which



could potentially become a problem at some point in the future. It is also proprietary and thus doesn't interact with other ORBs. The extremely proprietary nature of DCOM is certainly its biggest disadvantage. Microsoft only produces Windows versions of DCOM and in addition, it must run in their JVM. There are some third party attempts at porting it to other platforms, but so far these have met with limited success. Even if successfully ported, there exist deficiencies in the design of DCOM which make it inferior to CORBA. It does not follow the object oriented model since it precludes inheritance. There are work-arounds but they can be cumbersome. Also, object references are not persistent and due to the limitations of working only on Windows platforms, it is not scalable.

In light of the discussion, it becomes clear that CORBA addresses many of the issues we initially identified as pertinent to developing a distributed CAPS. With CORBA, the type of network becomes irrelevant to the operation of a distributed CAPS. Indeed, the proof of concept implementation done for this work started on a single PC and then migrated seamlessly to first an all PC environment and then to one where the client resided on a PC and the server on a Sun Solaris Unix. The next step, to operate over the internet, would be just as smooth a transition. Clearly, the heterogeneous nature of any large network becomes immaterial given the fact that every important language has an IDL mapping and every major type of operating system supports CORBA. The fact that no special hardware is required is also a plus. CORBA's extensive support for exceptions and the ability to implement user defined exceptions provide for greatly improved fault tolerance. Despite the intuitive feeling that the overhead of CORBA must be significant, it is actually shown to be quite acceptable.

With the introduction of real-time CORBA this year, latency should become even less of an issue. Lastly, the CORBA services and facilities provide extensive support for current and future growth.

## **V. A DISTRIBUTED CAPS PROOF OF CONCEPT IMPLEMENTATION**

The primary objective in the initial implementation was to demonstrate the efficacy of a distributed CAPS design using Java and CORBA. The first step in this process was completed with the HSI which was written in Java and included a PSDL editor. There was, however, little other functionality built into the initial HSI. Our goal was to create an implementation where the HSI would run on a PC and remotely invoke methods on a CAPS server located on a Sun Solaris Unix machine. The mechanism for these invocations would be CORBA. Since this was a proof of concept implementation, only selected functionality was demonstrated in order to prove the effectiveness of the Java/CORBA combination before attempting large scale development.

### **A. PRODUCT CHOICES FOR IMPLEMENTATION**

There are many possible environments for developing Java code and currently there are three major suppliers of CORBA/Java ORBs. In selecting an environment for developing Java code, the continuum runs from using the Java Developer Kit (JDK) supplied by JavaSoft and a text editor which are a very minimalist approach to full feature, comprehensive integrated development environments (IDE). Java IDEs are offered by a host of vendors. We decided to use the newest version of the JavaSoft JDK, 1.2.2. There were several reasons for this. First, the JDK is supplied free of charge. Secondly, in keeping with the walk before running philosophy, choosing the JDK gave us a simpler, more straightforward design environment. It allowed us to concentrate on creating a distributed CAPS without the distraction of all the bells and whistles that accompany the more feature laden IDEs. The documentation with the JDK

is also very comprehensive. As will be discussed below, the JDK has one additional advantage – a built-in Java ORB that is fully CORBA compliant.

The three Java ORBs available today are JavaSoft's JAVA IDL, Iona's OrbixWeb 3.0 and Borland/Visigenic's VisiBroker for Java 3.1. Much of the same reasoning for selecting JavaSoft's JDK for writing the Java code went into the selection of using JavaSoft's Java IDL for the ORB in our implementation. While far from a full feature ORB (it lacks such things as a non-volatile Naming Service and many of the add-on services) it is fully CORBA compliant. In our case, the simpler ORB was more a benefit than a disadvantage. As a distributed CAPS moves from the research stage to a production release, the work done can be easily migrated to a more full feature ORB. Also, it is very likely that JavaSoft will continue to improve the Java IDL and such a migration may not even be necessary. Java IDL is already built into the JDK environment. We did discover that some bugs exist in versions of JDK before 1.2.2 which prevented the Java IDL from operating properly. The IDL to Java compiler worked correctly but using a version earlier than 1.2.2 resulted in exceptions when attempting to connect to the ORB.

## **B. IMPLEMENTATION DECISIONS**

The HSI, as implemented by [DURA99], provided the GUI by which a user could create a prototype with the graphical editor and generate the corresponding PSDL code. It could open from and save to a local memory location or the user's allocated memory storage location on a network. The entire process was one controlled by the current instantiation of the HSI created by the user. In order to demonstrate the viability of a distributed CAPS operating in a client-server paradigm, we had to separate out some of

the functionality into distinct client and server processes. Additionally, we wanted to incorporate some of the tasks from the original stand-alone CAPS into our client-server designed one. The IDL file and the two files that implement the server are listed in Appendix B. The files generated by the IDL-to-Java compiler are listed in Appendix C. The HSI files that were modified to implement the remote calls to the server are listed in Appendix D. All three appendixes include both source files and the corresponding Javadoc generated documentation.

First, we decided that the user should be able to open an existing prototype from either the local memory or from prototypes stored on the server side of the network. The user was also given the ability to save files on the server side as well as in his/her own memory space as before. This remote save was implemented as the commit function of the HSI. We decided to incorporate execution of the translate function on the server side in order to further extend the work began in [DURA99]. When a user selects the translate function during a HSI session, the current prototype file is transferred to the server where the translate is actually invoked. The Ada files generated are stored on the server in the user's directory and a message notifying the user of a successful translate is sent to the HSI. Likewise, the user is notified after a prototype's PSDL file is successfully saved to the server side when the user invokes the commit function from the HSI. On the server side, when the translate function is invoked, a shell script is called which performs the actual translating. This is not a true three tier design but as the primary objective was to test the communication between the HSI and a remotely located server, it was acceptable. As this research matures, it may be that ultimately the server communicates with the 'back end' implementation

doing the actual work over a second CORBA object bus. In essence, the server would become the client in a client-server relationship with the 'back end' implementation.

The interfaces of the methods needed to implement this initial design were described in CORBA IDL and compiled using the IDL-to-Java compiler in JDK 1.2.2. When invoking the IDL-to-Java compiler, the command line argument `-fno-cpp` is used in order to disable the C++ preprocessor that is defaulted enabled. This preprocessing is not needed and prevented the IDL-to-Java from performing properly in the lab. The result were 15 .Java files which represented all the classes required to implement the interfaces described in the initial IDL description. The corresponding .class files were created by the JDK 1.2.2 Java compiler. The files were responsible for the actual operation of the CORBA object bus by marshalling and unmarshalling data being passed over the network between the client HSI and the server side. Additionally, they provide stub and skeleton implementations containing all the necessary information for proper communications with the ORB. They simply had to be extended and the desired functionality added, e.g. saving or translating a file, and they were ready to be compiled into .class files by the Java compiler.

Since this was a proof of concept demonstration, robustness, ease of use and efficiency were not primary considerations. Minimal error checking, such as ensuring that a prototype was opened in the HSI before invoking the translate function on the server, was incorporated. However, other than the built in mechanisms such as type checking, not much work was done in this area. Given the effectiveness of Java exceptions and the ease of creating user defined exceptions in CORBA, this will not be an impediment to future work. Likewise, three pieces of information are currently

entered on the command line when starting the HSI or the server. On starting the HSI, PROTOTYPEHOME and CAPSUser tell the system where the prototype PSDL files are located and who the current user is. In the event that PROTOTYPEHOME is not entered, the system will look in the user's home directory as a default. Again, the default is to the home directory of who ever starts the CAPS server. These could easily be incorporated into the respective GUIs in future work.

The sequence of events that brings the entire system on line are quite straightforward. First, on the machine that is hosting the server, start the naming service. Second, in another process, e.g., a separate DOS window, start the server implementation. Third, start the HSI on another machine (it can be started on the same machine in a third process). For all three, you must include the desired communication port. It must be the same for three processes. Additionally, for the client HSI you must include the internet location of the host running the server. More advanced implementations can eliminate these requirements. The following are examples of the actual semantics needed. To invoke the naming service, use "tnameserv -ORBInitialPort 1050." For the server, use "java -DCAPSJavaHome=\$CAPSHOME\caps CapsServer -ORBInitialPort 1050". Lastly, for the HSI use "java -PROTOTYPEHOME=\jdk2\caps - DCAPSUser=kreeger caps.Caps -ORBInitialHost 131.120.8.58 -ORBInitialPort 1050".

THIS PAGE INTENTIONALLY LEFT BLANK



## VI. CONCLUSIONS AND FUTURE WORK

Our initial tests are very encouraging. We were able to easily integrate the existing Java HSI running on a Windows NT machine into a client-server arrangement with the server running on a Sun Solaris Unix machine. We successfully created prototypes in the graphical editor of the HSI and saved the resulting PSDL files both to the HSI's memory and transferred it over the network, where the server stored in its memory. We successfully opened PSDL files from both locations and displayed them in the graphical editor. Additionally, we sent PSDL files over the network to the server where they were translated and the corresponding Ada files generated. Thus, there appears to be little doubt about the ability of the current CAPS to be implemented in a client-server design that functions over any size network, including ultimately the Internet. The machinery that makes this not only possible, but even a reasonable effort, is CORBA. CORBA makes it feasible to convert the current CAPS to a client-server architecture while preserving much of the existing codebase. It is straightforward to instantiate a CAPS object on the server and then send a reference object to the CAPS object to any client. The users on the client side interact with the HSI's extremely intuitive interface and whenever they need CAPS functionality, such as translate, the HSI simply invokes the method on the CAPS object reference that was received from the server. Thus much of the more complicated code that has been developed for the stand-alone CAPS can continue to be used. To this code, the origins of the request are irrelevant. As long as the proper parameters are passed in, the expected result will be produced and where the result is ultimately sent doesn't matter. This arrangement

allows the HSI to manage the user inputs to a prototype design locally. This is a very efficient manner for this type of operations. The more heavy duty, less frequently invoked functionality can reside on the server side of the network. The additional benefit of this design is that as more effective means of executing these services are implemented or additional requirements are discovered, it is a much simpler matter to update the single copy of the code located on the server side compared to trying to update code on a variety of different client machines across an entire network. As long as the interfaces remain unchanged, any alterations in one tier of the architecture will remain transparent to the other tiers.

As for the future, much remains to be done in order to fully realize the potential of a distributed CAPS. The work done thus far has clearly shown the potential that exists. However, to become a practical system for creating and managing prototyping for large software projects, a distributed CAPS must be fully functional and much more robust than at present. The next logical phase of this research should focus on two parallel tracks.

On the HSI the effort should concentrate on implementing the rest of the existing functionality for the current CAPS and on engineering production quality robustness into the HSI. The implementations of the commit and translate functions in this work provide an example of how to invoke from the HSI existing CAPS methods on a CAPS server and return the results to the HSI. Incorporating additional methods into the IDL interface definition and recompiling it are straightforward processes. This will produce the files necessary to implement the additional methods on the server side and the HSI. The second part of the effort for improving the HSI is easily done concurrently with the

first part. As additional methods are included into the HSI, every effort should be made to ensure production quality robustness. The ease of creating user defined exceptions in both Java and CORBA simplify this effort tremendously.

In parallel with the HSI work, there remain tasks to be completed on the CAPS software. First, while comprehensive theoretical work has been done on every aspect of CAPS, there remains functionality that either needs to be completed or enhanced. This is especially true in regards to the reusable database. Finishing this work in conjunction with the HSI work provides the opportunity to tightly coordinate efforts. Secondly, the CAPS method implementations should be separated out from the CAPS server. This would allow the CAPS server to be as efficient as possible and easily scale to a larger number of users. Furthermore, by isolating the actual method of implementations from the CAPS server, they can be modified and updated more easily.

In conclusion, the endeavor to deploy a distributed CAPS is both feasible and necessary. The emergence of the Java/CORBA technology supplies the means by which migrating CAPS from an isolated system to a fully functional distributed system becomes not just possible, but quite manageable. That is fortunate timing, for if CAPS is to continue to be a driving force in software engineering research it must adapt to the new realities of a networked world.

THIS PAGE INTENTIONALLY LEFT BLANK

## APPENDIX A: EXPANDED USE CASES

Selected used cases are detailed below in expanded format:

Use Case:	<b>U3. Open project</b>
Actors:	User
Type:	primary and essential
Purpose:	Allow user to open a project
Overview:	After logging into CAPS, user can select an existing project and open it, if he has the proper authorization from the project manager.  If the user has project manager privileges, they can open a new project.
Cross Reference:	Functions: R2.3, R4.2, R4.3  Use Cases: User must have completed the Start-up and Log in use cases

### Section: Main

#### Typical Course of Action

Actor Action	System Response
1. This use case begins after the user has successfully started and logged into CAPS	
2. The user chooses to open a new or existing prototype	

a. If the user chooses a new prototype, see section Open New Prototype

b. If the user chooses an existing prototype, see section Open Existing Prototype

**Section: Open New Prototype**

Typical Course of Action

Actor Action	System Response
1. The user selects the new prototype option	2. The system verifies the user has project manager level privileges.
4. The user inputs new prototype information	3. The system prompts the user for new prototype information
	5. A new prototype is created

**Section: Open Existing Prototype**

Typical Course of Action

Actor Action	System Response
1. The user selects the open existing prototype option	2. System presents the user with a list of existing prototypes within a file structure that can be navigated
3. The user highlights the desired prototype and selects to open it	4. The selected prototype is loaded at the users workstation

**Use Case:** U4. Modify prototype

**Actors:** User

**Type:** primary and essential

**Purpose:** Allow user to modify a prototype

**Overview:** After opening a prototype, the user will be able to make changes to the graphical interface of the control flow diagram or to the text boxes associated with either objects or data streams in the diagram. These changes will be automatically reflected in the PSDL code. The user can record user comments, stakeholder comments and requirements for the system being prototyped. Additionally, the user can directly access the PSDL code and change it manually.

**Cross Reference:** Functions: R1.1, R1.2, R1.3, R1.4, R1.5, R1.6, R1.7, R1.21, R4.4, R4.5

Use Cases: User must have completed the Start-up, Log in and Open Project use cases

**Section: Main**

Typical Course of Action

Actor Action

System Response

1. This use case starts after the user has opened a prototype

2. The user can make changes graphically or textually to the control flow diagram or may edit the actual PSDL code

a. If user makes graphical changes, see section Graphical Changes

b. If user makes textual changes, see section Textual Changes

c. If the user edits the PSDL code, see section Edit PSDL Code

### **Section: Graphical Changes**

#### Typical Course of Action

Actor Action	System Response
1. User selects graphical input from the menu of inputs, such as circle or line, or selects an existing graphical representation within the control flow diagram and manipulates it within the diagram	2. The changes made by the user are displayed in the diagram and the PSDL code in main memory is generated and/or modified



## Section: Textual Changes

### Typical Course of Action

Actor Action	System Response
1. The user chooses an existing graphical object from the control flow diagram and selects to add textual input to it	2. The text box for the graphical object selected is displayed
3. The user makes the desired changes to the textual input box and confirms done when the input is completed	4. The text box is no longer displayed, the changes are accepted, the PSDL code is generated and/or modified in main memory and if applicable the diagram presentation is modified

## Section: Edit PSDL Code

### Typical Course of Action

Actor Action	System Response
1. The user selects the option to directly edit the PSDL code from a CAPS menu	2. The PSDL code for the prototype is displayed within a text editor
3. The user makes changes to the PSDL code and selects save or closes the editor when done	4. The PSDL code is modified in main memory
	5. The modified PSDL code is validated for correctness before the user can exit the text editor

**Use Case:** U5. Retrieve component from reuse database

**Actors:** User

**Type:** primary and essential

**Purpose:** To retrieve reusable components from library

**Overview:** The user accesses the reuse database and inputs search parameters. CAPS responds with no match, unique match or a list of possible matches. CAPS generates an alert if there is an error in transmission and an acknowledgement if the transfer is successful.

**Cross Reference:** Functions: R1.21, R3.2, R3.4, R3.6  
 Use Cases: User must have completed the Start-up, Log in and Open Project use cases

**Section: Main**

Typical Course of Action

Actor Action	System Response
1. The user selects option from CAPS menu to retrieve component from reusable component library	2. An input box is displayed for the user to input the desired parameters of the component to be returned
3. The user inputs the desired component parameters and selects retrieve	4. The parameters are sent to the reuse component library, possible over a network to a remote site
	5. The parameters are accepted, a query formulated and the reuse component library searched

7. The user selects the desired component, if one available
6. The user is informed that there was a match, multiple possible matches or no match
8. A copy of the selected component is returned to the user
9. The user is notified that the transfer was successful

#### Alternate Courses

- Line 9. If the transfer is unsuccessful, the user is notified. The type of error is displayed, if known

**Use Case:** U7. Modify GUI for displaying prototype execution

**Actors:** User

**Type:** primary and essential

**Purpose:** To modify the GUI generated to display prototype functionality

**Overview:** User selects and then edits the files that provide functionality for displaying prototype execution

**Cross Reference:** Functions: R1.4, R1.5, R1.6, R1.7, R1.8, R1.9

Use Cases: User must have completed the Start-up, Log in and Open Project use cases and have completed the Generate Executable Prototype use case sometime previously (not necessarily the same session)

## Section: Main

### Typical Course of Action

#### Actor Action

1. This use case starts after the user has opened a prototype
2. The user can make changes to the source code to affect prototype functionality or to the prototype graphical interface
  - a. If user makes functionality changes, see section Functional Changes
  - b. If user makes graphical interface changes, see section Interface Changes

#### System Response

## Section: Functional Changes

### Typical Course of Action

#### Actor Action

1. The user selects the edit source file option from a CAPS menu

#### System Response

2. A list of normal programming source files (e.g. Ada or Java) within the current prototype is displayed

- |   |  |
|---|--|
| <p>3. The user selects the source file to edit</p> <p>5. The user makes changes to the file</p> <p>6. The user saves the file</p> <p>8. The user quits the file</p> <p>9. The user quits the editor</p> | <p>4. The selected file is opened in a text editor</p> <p>7. The file is written to persistent storage</p> <p>10. The user is queried about saving any unsaved changes, which will be saved to persistent storage and the editor is closed</p> |
|---|--|

**Section: Interface Changes**

Typical Course of Action

- | Actor Action   | System Response  |
|--|--|
| <p>1. The user selects the edit interface option from a CAPS menu</p> <p>3. The user makes the desired changes to the prototype GUI</p> <p>4. The user selects the generate code option from the GUI builder</p> <p>6. The user saves the file(s)</p> <p>8. The user quits the GUI builder</p> | <p>2. A GUI builder is invoked with the current prototype GUI opened</p> <p>5. Source code for the prototype GUI, which is controlled by the CAPS generated control source code, is automatically generated</p> <p>7. The file(s) is written to persistent storage</p> |

**Use Case:** U8. Generate Executable Prototype

**Actors:** User

**Type:** primary and essential

**Purpose:** Prepare a prototype for execution

**Overview:** The user translates the prototype in order to create source code in an implementation language from the PSDL and link it. The user will then verify through CAPS that all static hard real time constraints are met. The user will then compile the prototype source code.

**Cross Reference:** Functions: R1.9, R1.10, R1.11

Use Cases: User must have completed the Start-up, Log in and Open Project use cases

**Section: Main**

Typical Course of Action

Actor Action	System Response
1. The user selects the translate option from a CAPS menu	2. PSDL code for prototype is used to generate 3 <sup>rd</sup> generation object oriented language source code
3. The user selects the schedule option from a CAPS menu	4. All static hard real time constraints are verified as met
5. The user selects the compile option from a CAPS menu	6. All source files are compiled and executable files created

## Alternate Courses

- Line 4. Some static hard real time constraint is missed. CAPS generates an alert for the user.

**Use Case:** U9. Execute the prototype

**Actors:** User

**Type:** primary and essential

**Purpose:** Execute the prototype and perform analysis of system constraints

**Overview:** The user will execute the prototype. As the user test the prototype functionality, CAPS will verify that all dynamic hard real time, concurrency and network constraints are met. CAPS will allow the user to record stakeholder and/or user comments.

**Cross Reference:** Functions: R1.1, R1.2, R1.3, R1.12, R1.13, R1.14, R1.15, R1.16

Use Cases: User must have completed the Start-up, Log in and Open Project use cases and have completed the Generate Executable Prototype use case sometime previously (if not immediately after generating the executable prototype, you may execute a prototype that doesn't have the most recent changes)

## Section: Main

### Typical Course of Action

Actor Action	System Response
1. The user selects the execute option from a CAPS menu	2. The prototype GUI interface is generated and execution of the system being designed begins
3. The user test the functionality of the designed system with manual inputs or scripted tests	4. Dynamic hard real time constraints are verified met 5. If the designed system is multi-threaded, concurrency constraints are verified met 6. If the designed system is distributed, network constraints are verified met
7. The user selects the record comments option from a CAPS menu	8. A menu is displayed that allows the choice of selecting user or stakeholder comment inputs
9. The comments are entered and the comment entry box deselected	10. The comments are saved to persistent storage and become part of the project record

### Alternate Courses

- Line 4. Dynamic hard real time constraints are not met. CAPS generates an alert for the user.



- Line 5. Concurrency constraints are not met. CAPS generates an alert for the user.
- Line 6. Network constraints are not met. CAPS generates an alert for the user.

Use Case: **U10. Manage project changes**

Actors: User, Project Manager

Type: primary and essential

Purpose: Allow Project Manager to control configuration and version control for a project

Overview: User (possibly more than one) will submit module(s) to the Project Manager for incorporation into the project prototype. The user may elect to return the entire prototype after making changes to only some of the modules. In this case the Project Manager must identify which modules are changed or new. If the module does not already exist in the project database, the Project Manager will merge the submitted module(s) into one module, resolving any conflicts. CAPS will assign and track a version number and insert the module into the project database, making ties to all other modules that may exist in the project database. If the module previously existed in the project database, the Project Manager will merge all submitted modules with the existing one, resolving any conflicts. CAPS will update the version number. If an existing module is submitted to the Project Manager, CAPS will verify that all other modules that it depends on are the most up to date

version. CAPS will generate an alert if there is an error in transmitting a module and an acknowledgement if there is no error.

Cross Reference: Functions: R1.19, R1.20, R3.1, R3.3, R3.4, R3.6, R4.3, R4.4, R4.6, R4.7, R4.8

Use Cases: Project Manager must have completed the Start-up, Log in and Open Project use cases

**Section: Main**

Typical Course of Action

Actor Action	System Response
1. This use case begins with the Project Manager selecting the option from a CAPS menu to review modules that have been submitted for the current project	2. All modules submitted for a project that are pending action are displayed
3. Project Manager determines module update state: a. If a single module not previously in the project is submitted, see section Single New Module b. If multiple versions of a module not previously in the project are submitted, see section Multiple New Modules	

c. If a single module previously in the project is submitted, see section Single Old Module

d. If multiple versions of a module previously in the project are submitted, see section Multiple Old Modules

### Alternate Courses

- Line 2. If there was an error in receiving any module, the Project Manager is alerted

### Section: Single New Module

#### Typical Course of Action

Actor Action	System Response
1. The Project Manager selects the option to view the new module from a CAPS menu	2. The module is displayed in a text format
3. If the Project Manager decides to include the module in the project, she selects the add option from a CAPS menu	4. The Project Manager is asked where to save the file and exactly which project to include it in
5. The Project Manager specifies the location to save to and the project into which the module is to be included	6. The module is saved and included as directed

- 7. The module will be assigned a version number automatically
- 8. The module will be registered with the project automatically

**Section: Multiple New Modules**

Typical Course of Action

Actor Action	System Response
1. The Project Manager selects the option to view the new modules from a CAPS menu	2. The modules are displayed in a text format as selected
3. If the Project Manager decides to include the modules in the project, he selects the merge option from a CAPS menu	4. The Project Manager is asked which modules to merge
6. The Project Manager specifies the location to save to and the project into which the module is to be included	5. After merging the modules into a single prototype, the Project Manager is asked where to save the file and exactly which project to include it in
	7. The module is saved and included as directed
	8. The module will be assigned a version number automatically

9. The module will be registered with the project automatically

#### Alternate Courses

- Line 5. If the modules cannot be successfully merged automatically, the Project Manager is sent an alert along with information about the conflict(s)

#### Section: Single Old Module

##### Typical Course of Action

Actor Action	System Response
1. The Project Manager selects the option to view the new module from a CAPS menu	2. The module is displayed in a text format
3. If the Project Manager decides to replace the existing module in the project, she selects the merge option from a CAPS menu	4. The Project Manager is asked to specify the modules to merge
6. The Project Manager specifies the location to save to and the project into which the module is to be included	5. After merging the modules into a single prototype, the Project Manager is asked where to save the file and exactly which project to include it in
	7. The module is saved and included as directed

8. The module will be assigned an updated version number automatically

#### Alternate Courses

- Line 3. The Project Manager can elect to simply replace the existing module with the new one
- Line 5. If the modules cannot be successfully merged automatically, the Project Manager is sent an alert along with information about the conflict(s)

#### Section: Multiple Old Modules

##### Typical Course of Action

Actor Action	System Response
1. The Project Manager selects the option to view the new modules from a CAPS menu	2. The modules are displayed in a text format as selected
3. If the Project Manager decides to include the modules in the project, he selects the merge option from a CAPS menu	4. The Project Manager is asked to specify the new modules to merge into a single module
	5. After merging the modules into a single module, the Project Manager is asked which existing module to merge with the single new module

7. The Project Manager specifies the location to save to and the project into which the module is to be included
6. After merging the modules into a single prototype, the Project Manager is asked where to save the file and exactly which project to include it in
8. The module is saved and included as directed
9. The module will be assigned an updated version number automatically

#### Alternate Courses

- Line 4. The Project Manager may elect to merge the new modules and the existing module at the same time
- Line 5. If the modules cannot be successfully merged automatically, the Project Manager is sent an alert along with information about the conflict(s)
- Line 6. If the modules cannot be successfully merged automatically, the Project Manager is sent an alert along with information about the conflict(s)

Use Case:           **U11. Manage reuse database changes**

Actors:             User, Software Librarian

Type:               primary and essential

Purpose:             Allow Software Librarian to control configuration and version control  
for the reuse database

**Overview:** User submits a new module, or an existing one that was modified, to the Software Librarian. When the Software Librarian accepts the module for inclusion in the reuse database, a version control number is assigned or updated as necessary and the module is saved to the reuse library. If the module was an update of an existing one, all users of the old version are alerted. The Software Librarian can also delete modules.

**Cross Reference:** Functions: R1.22, R1.23, R3.1, R3.3, R3.4, R3.6  
Use Cases: Software Librarian must have completed the Start-up and Log in use cases

**Section: Main**

Typical Course of Action

Actor Action	System Response
1. This use case begins with the Software Librarian selecting the option from a CAPS menu to review modules that have been submitted for inclusion in the reuse database	2. All modules submitted for inclusion that are pending action are displayed
3. The Software Librarian selects the option to view the a module on the list from a CAPS menu	4. The module is displayed in a text format



5. After review, the Software Librarian selects the add option from CAPS menu.

6. The Software Librarian chooses to add the module as a new one or to replace an existing module with the new one:

a. If the module is added as a completely new one, see section Add New Module

b. If the module is replacing an existing module, see section Replace Existing Module

### **Section: Add New Module**

#### Typical Course of Action

Actor Action	System Response
1. The Software Librarian elects to add the submitted module as a new module	2. The module is added to the reuse database and a version control number is assigned

## Section: Replace Existing Module

### Typical Course of Action

Actor Action	System Response
1. The Software Librarian elects to replace an existing module with the submitted module	2. The new module replaces the existing one in the reuse database 3. The version control number is updated 4. Users of the old version are alerted that there are changes to the module they checked out

## APPENDIX B: CAPS SERVER IMPLEMENTATION SOURCE CODE

This appendix contains the IDL file that describes the methods defined in order to implement a distributed CAPS server. It also contains the source files and Javadoc generated documentation for the CAPS server.

```
/**
 * The Interface Description Language (IDL) for a Distributed CAPS
 *
 * @author Gary Kreeger
 * @version 1.0
 */

module DistributedCaps
{
    interface DistCaps
    {
        exception cantWriteFile {};
        exception cantReadFile {};

        typedef sequence<octet> prototype_file;
        typedef sequence<string> prototype_list;

        string translate (in prototype_file psdl_file, in string name,
            in string version, in string user) raises (cantWriteFile);

        prototype_list get_proto_list (in string user);

        prototype_file open_proto (in string name, in string user) raises (cantReadFile);

        string commit (in prototype_file psdl_file, in string name,
            in string version, in string user) raises (cantWriteFile);
    };
};

/**
 * The DistributedCaps Server main program. It instantiates a DisCapsImpl
 * object, starts the orb and registers the object with the orb.
 *
 * @author Gary Kreeger
 * @version 1.0
 */
```

```

import DistributedCaps.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

public class CapsServer
{ static public void main(String[] args)
  try
  {
    // Initialize the ORB
    ORB orb = ORB.init(args, null);

    // Create the Caps object
    DistCapsImpl caps = new DistCapsImpl();
    orb.connect (caps);

    //get the root naming context
    org.omg.CORBA.Object objRef = orb.resolve_initial_references ("NameService");
    NamingContext ncRef = NamingContextHelper.narrow (objRef);

    // bind the Object Reference in Naming
    NameComponent nc = new NameComponent ("DistCaps", " ");
    NameComponent path[] = {nc};
    ncRef.rebind (path, caps);

    //wait for invocations from client
    java.lang.Object sync = new java.lang.Object();
    synchronized (sync)
    {
      sync.wait();
    }
  }
  catch (Exception e)
  {
    System.err.println ("Error: " + e);
    e.printStackTrace (System.out);
  }
} //end CapsServer

/**
 * The Implementation for the distributed CAPS object.
 *
 * @author Gary Kreeger
 * @version 1.0
 */

import DistributedCaps.*;
import java.io.*;
import java.io.File;
import java.util.Vector;
import javax.swing.filechooser.FileSystemView;

public class DistCapsImpl extends _DistCapsImplBase

```

```

{
/**
 * the constructor for a distributed CAPS object
 */

DistCapsImpl()
{
    super();
    System.out.println ("Caps Object Created");
}

/**
 * Translate function for creating Ada files from a PSDL file
 *
 * @param psdl_file The PSDL file to be translated
 * @param name The name of the PSDL file
 * @param version The version number of the PSDL file being translated
 * @param user The name of the user at the client session
 *
 * @return A string to confirm the file was transferred and translated
 *
 * @throws DistributedCaps.DistCapsPackage.cantWriteFile
 */

public String translate (byte[] psdl_file, String name, String version, String user)
    throws DistributedCaps.DistCapsPackage.cantWriteFile
{
    boolean tempBool = true;
    String protoHome;
    // MTS 8/25/99
    // added local variable userHome
    String userHome = "";

    try
    {
        String CapsServerFiles = System.getProperty("CAPSJavaHome");
        if (CapsServerFiles == null) // CAPSJavaHome not set on command line
        {
            //default to the home directory
            File homeDir = FileSystemView.getFileSystemView ().getHomeDirectory ();
            protoHome = new String (homeDir + File.separator + user +
                File.separator + ".caps");

            // MTS 8/25/99
            // added code to initialize userHome
            userHome = (homeDir + File.separator + user);

            File protoDir = new File (protoHome);
            if (!protoDir.exists ())
            {
                protoDir.mkdir ();
            }
        }
        else
        {
            protoHome = new String (CapsServerFiles + File.separator + user +

```

```

        File.separator + ".caps");

// MTS 8/25/99
// added code to initialize userHome
userHome = (CapsServerFiles + File.separator + user);

File protoDir = new File (protoHome);
if (!protoDir.exists ())
{
    protoDir.mkdir ();
}
}

//ensure the file is created, read the byte array into a FileOutputStream
//and then read the FileOutputStream into the file
String proto = name;
File PSDL_Demo = new File (protoHome + File.separator + name +
    File.separator + version + File.separator + proto + ".psdl");
tempBool = PSDL_Demo.createNewFile();
FileOutputStream fos = new FileOutputStream (PSDL_Demo);
fos.write (psdl_file);
fos.close();
}
catch (Exception e)
{
    throw new DistributedCaps.DistCapsPackage.cantWriteFile();
}

// MTS 8/25/99
// replace protoHome with userHome in the following call to translate.script
// String command = "translate.script " + protoHome + " " + name + " " + version;

String command = "translate.script " + userHome + " " + name + " " + version;

try
{
    Runtime run = Runtime.getRuntime();
    run.exec(command);
}
catch (IOException ex)
{
    System.out.println (ex);
}

return "\nThe PSDL file was successfully transferred to the server\n";
} //end Translate

/**
 * Get the list of prototypes available remotely
 *
 * @param user The name of the user at the client session
 *
 * @return An array of prototype names that may be selected for opening
 */

public String [] get_proto_list(String user)

```

```

{
String [] protolist;
String CapsServerFiles = System.getProperty("CAPSJavaHome");
String protoHome;
File protoDir;

if (CapsServerFiles == null) // CAPSJavaHome not set on command line
{
File homeDir = FileSystemView.getFileSystemView ().getHomeDirectory ();
protoHome = new String (homeDir + File.separator + user +
File.separator + ".caps");
protoDir = new File (protoHome);
if (!protoDir.exists ())
{
protoDir.mkdir ();
}
}
else
{
protoHome = new String (CapsServerFiles + File.separator + user +
File.separator + ".caps");
protoDir = new File (protoHome);
if (!protoDir.exists ())
{
protoDir.mkdir ();
}
}

// vector to hold prototype names
Vector prototypeNames = new Vector (0, 2);
// array to hold list of existing files
File [] dirs = protoDir.listFiles ();

if (dirs.length == 0) // no files exist
{
return protolist = new String [0];
}
else
{
for (int ix = 0; ix < dirs.length; ix++)
{
String protoName = "";
protoName = dirs [ix].getName ();
File subDirs [] = dirs [ix].listFiles ();
for (int jx = 0; jx < subDirs.length; jx++)
{
prototypeNames.addElement (protoName.concat
(File.separator + subDirs [jx].getName ());
}
}
}

//get the vector into an object array and then convert it to a string array
Object [] temp_proto_list = prototypeNames.toArray ();
protolist = new String [temp_proto_list.length];

for (int ix = 0; ix < temp_proto_list.length; ix++)

```

```

        {
            protolist[ix] = String.valueOf(temp_proto_list[ix]);
        }

        return protolist;
    }
} //end get_proto_list

/**
 * Open the selected prototype
 *
 * @param name The name of the prototype opened
 * @param user The name of the user at the client session
 *
 * @return A byte array holding the selected prototype PSDL file
 *
 * @throws DistributedCaps.DistCapsPackage.cantReadFile
 */
public byte[] open_proto (String name, String user)
    throws DistributedCaps.DistCapsPackage.cantReadFile
{
    try
    {
        byte[] proto_file;
        String CapsServerFiles = System.getProperty("CAPSJavaHome");
        String protoHome;
        File protoDir;

        if (CapsServerFiles == null) // CAPSJavaHome not set on command line
        {
            File homeDir = FileSystemView.getFileSystemView ().getHomeDirectory ();
            protoHome = new String (homeDir + File.separator + user +
                File.separator + ".caps");
            protoDir = new File (protoHome);
            if (!protoDir.exists ())
            {
                protoDir.mkdir ();
            }
        }
        else
        {
            protoHome = new String (CapsServerFiles + File.separator + user +
                File.separator + ".caps");
            protoDir = new File (protoHome);
            if (!protoDir.exists ())
            {
                protoDir.mkdir ();
            }
        }

        if (name == null)
        {
            return proto_file = new byte[0];
        }
    }
}

```



```

else
{
    // create file object with which to manipulate the selected file
    File selectedDir = new File (protoHome + File.separator + name);
    File file = new File (selectedDir.getAbsolutePath () + File.separator +
        selectedDir.getParentFile ().getName () + ".psdl");
    if (!file.exists ())
    {
        return proto_file = new byte[0];
    }
    else
    {
        //read the opened file into a FileInputStream and then read the
        //FileInputStream in the byte array to be returned
        FileInputStream in = new FileInputStream (file);
        proto_file = new byte[in.available()];
        in.read (proto_file);
        return proto_file;
    }
}
}
}
catch (Exception e)
{
    throw new DistributedCaps.DistCapsPackage.cantReadFile();
}

} //end open_proto

/**
 * Save a prototype's PSDL file on a local client to the remote server
 *
 * @param psdl_file The PSDL file to be translated
 * @param name The name of the PSDL file
 * @param version The version number of the PSDL file being translated
 * @param user The name of the user at the client session
 *
 * @return A string to confirm the file was transferred
 *
 * @throws DistributedCaps.DistCapsPackage.cantWriteFile
 */
public String commit (byte[] psdl_file, String name, String version, String user)
    throws DistributedCaps.DistCapsPackage.cantWriteFile
{
    boolean tempBool = true;
    String protoHome = "";
    String proto = name;
    String completePath = "";

    try
    {
        String CapsServerFiles = System.getProperty("CAPSJavaHome");
        if (CapsServerFiles == null) // CAPSJavaHome not set on command line
        {

```

```

File homeDir = FileSystemView.getFileSystemView ().getHomeDirectory ();

protoHome = (homeDir + File.separator + user +
    File.separator + ".caps");
File protoDir = new File (protoHome);
if (!protoDir.exists ())
{
    protoDir.mkdir ();
}
}
else
{
    protoHome = (CapsServerFiles + File.separator + user +
        File.separator + ".caps");
    File protoDir = new File (protoHome);
    if (!protoDir.exists ())
    {
        tempBool = protoDir.mkdirs ();
    }
}

completePath = (protoHome + File.separator + name + File.separator + version);
File completeDirs = new File (completePath);
if (!completeDirs.exists ()) // ensure the correct directory exist to save to
{
    tempBool = completeDirs.mkdirs();
}

//ensure the file is created, read the byte array into a FileOutputStream
//and then read the FileOutputStream into the file
File PSDL_Demo = new File (completePath + File.separator + proto + ".psdl");
tempBool = PSDL_Demo.createNewFile();
FileOutputStream fos = new FileOutputStream (PSDL_Demo);
fos.write (psdl_file);
fos.close();
}

catch (Exception e)
{
    System.out.println (e);
    throw new DistributedCaps.DistCapsPackage.cantWriteFile();
}

return "\n\nThe PSDL file was successfully transferred to the server\n\n";
}
} //end commit

```

## Class CapsServer

java.lang.Object

|  
+---CapsServer

---

public class **CapsServer**  
extends java.lang.Object

The DistributedCaps Server main program. It instantiates a DisCapsImpl object, starts the orb and registers the object with the orb.

---

### Constructor Summary

[CapsServer](#)()

### Method Summary

static void [main](#)(java.lang.String[] args)

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Constructor Detail

#### CapsServer

public CapsServer()

### Method Detail

#### main

public static void [main](#)(java.lang.String[] args)

## Class DistCapsImpl

```

java.lang.Object
|
+--org.omg.CORBA.portable.ObjectImpl
    |
    +--org.omg.CORBA.DynamicImplementation
        |
        +--DistributedCaps._DistCapsImplBase
            |
            +--DistCapsImpl
  
```

```

public class DistCapsImpl
extends DistributedCaps._DistCapsImplBase
  
```

The Implementation for the distributed CAPS object.

See Also:

[Serialized Form](#)

### Method Summary

java.lang.String	<b><u>commit</u></b> (byte[] psdl_file, java.lang.String name, java.lang.String version, java.lang.String user) Save a prototype's PSDL file on a local client to the remote server
java.lang.String[]	<b><u>get proto list</u></b> (java.lang.String user) Get the list of prototypes available remotely
byte[]	<b><u>open proto</u></b> (java.lang.String name, java.lang.String user) Open the selected prototype
java.lang.String	<b><u>translate</u></b> (byte[] psdl_file, java.lang.String name, java.lang.String version, java.lang.String user) Translate function for creating ADA files from a PSDL file

### Methods inherited from class DistributedCaps.\_DistCapsImplBase

\_ids, invoke

## Methods inherited from class org.omg.CORBA.portable.ObjectImpl

`_create_request`, `_create_request`, `_duplicate`, `_get_delegate`, `_get_domain_managers`,  
`_get_interface_def`, `_get_policy`, `_hash`, `_invoke`, `_is_a`, `_is_equivalent`, `_is_local`,  
`_non_existent`, `_orb`, `_release`, `_releaseReply`, `_request`, `_request`,  
`_servant_postinvoke`, `_servant_preinvoke`, `_set_delegate`, `_set_policy_override`,  
`equals`, `hashCode`, `toString`

## Methods inherited from class java.lang.Object

`clone`, `finalize`, `getClass`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

## Method Detail

### translate

```
public java.lang.String translate(byte[] psdl_file,  
                                java.lang.String name,  
                                java.lang.String version,  
                                java.lang.String user)  
    throws DistributedCaps.DistCapsPackage.cantWriteFile
```

Translate function for creating ADA files from a PSDL file

#### Overrides:

translate in class DistributedCaps.\_DistCapsImplBase

#### Parameters:

`psdl_file` - The PSDL file to be translated  
`name` - The name of the PSDL file  
`version` - The version number of the PSDL file being translated  
`user` - The name of the user at the client session

#### Returns:

A string to confirm the file was transferred and translated

#### Throws:

DistributedCaps.DistCapsPackage.cantWriteFile -

---

### get\_proto\_list

```
public java.lang.String[] get_proto_list(java.lang.String user)
```

Get the list of prototypes available remotely

#### Overrides:

get\_proto\_list in class DistributedCaps.\_DistCapsImplBase

#### Parameters:

`user` - The name of the user at the client session

#### Returns:

An array of prototype names that may be selected for opening

---

## open\_proto

```
public byte[] open_proto(java.lang.String name,  
                        java.lang.String user)  
    throws DistributedCaps.DistCapsPackage.cantReadFile
```

Open the selected prototype

**Overrides:**

open\_proto in class DistributedCaps.\_DistCapsImplBase

**Parameters:**

name - The name of the prototype opened  
user - The name of the user at the client session

**Returns:**

A byte array holding the selected prototype PSDL file

**Throws:**

DistributedCaps.DistCapsPackage.cantReadFile -

---

## commit

```
public java.lang.String commit(byte[] psdl_file,  
                              java.lang.String name,  
                              java.lang.String version,  
                              java.lang.String user)  
    throws DistributedCaps.DistCapsPackage.cantWriteFile
```

Save a prototype's PSDL file on a local client to the remote server

**Overrides:**

commit in class DistributedCaps.\_DistCapsImplBase

**Parameters:**

psdl\_file - The PSDL file to be translated  
name - The name of the PSDL file  
version - The version number of the PSDL file being translated  
user - The name of the user at the client session

**Returns:**

A string to confirm the file was transferred

**Throws:**

DistributedCaps.DistCapsPackage.cantWriteFile -

---

**Class** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

---

## APPENDIX C: IDL-TO-JAVA COMPILER GENERATED SOURCE CODE

This appendix contains the source files and corresponding Javadoc generated documentation for the files created by the IDL-to-Java compiler when creating the distributed CAPS server.

```
/*
 * File: ./DISTRIBUTEDCAPS/DISTCAPS.JAVA
 * From: DISTCAPS.IDL
 * Date: Wed Aug 18 20:38:43 1999
 * By: idltojava Java IDL 1.2 Aug 18 1998 16:25:34
 */

package DistributedCaps;
public interface DistCaps
    extends org.omg.CORBA.Object, org.omg.CORBA.portable.IDLEntity {
    String translate(byte[] psdl_file, String name, String version, String user)
        throws DistributedCaps.DistCapsPackage.cantWriteFile;
    String[] get_proto_list(String user);
    byte[] open_proto(String name, String user)
        throws DistributedCaps.DistCapsPackage.cantReadFile;
    String commit(byte[] psdl_file, String name, String version, String user)
        throws DistributedCaps.DistCapsPackage.cantWriteFile;
}
```

```
/*
 * File: ./DISTRIBUTEDCAPS/_DISTCAPSIMPLBASE.JAVA
 * From: DISTCAPS.IDL
 * Date: Wed Aug 18 20:38:43 1999
 * By: idltojava Java IDL 1.2 Aug 18 1998 16:25:34
 */

package DistributedCaps;
public abstract class _DistCapsImplBase extends org.omg.CORBA.DynamicImplementation implements
DistributedCaps.DistCaps {
    // Constructor
    public _DistCapsImplBase() {
        super();
    }
    // Type strings for this class and its superclasses
    private static final String _type_ids[] = {
        "IDL:DistributedCaps/DistCaps:1.0"
    };

    public String[] _ids() { return (String[]) _type_ids.clone(); }

    private static java.util.Dictionary _methods = new java.util.Hashtable();
    static {
```

```

_methods.put("translate", new java.lang.Integer(0));
_methods.put("get_proto_list", new java.lang.Integer(1));
_methods.put("open_proto", new java.lang.Integer(2));
_methods.put("commit", new java.lang.Integer(3));
}
// DSI Dispatch call
public void invoke(org.omg.CORBA.ServerRequest r) {
    switch (((java.lang.Integer) _methods.get(r.op_name())).intValue()) {
        case 0: // DistributedCaps.DistCaps.translate
            {
                org.omg.CORBA.NVList _list = _orb().create_list(0);
                org.omg.CORBA.Any _psdl_file = _orb().create_any();
                _psdl_file.type(DistributedCaps.DistCapsPackage.prototype_fileHelper.type());
                _list.add_value("psdl_file", _psdl_file, org.omg.CORBA.ARG_IN.value);
                org.omg.CORBA.Any _name = _orb().create_any();
                _name.type(org.omg.CORBA.ORB.init().get_primitive_tc(org.omg.CORBA.TCKind.tk_string));
                _list.add_value("name", _name, org.omg.CORBA.ARG_IN.value);
                org.omg.CORBA.Any _version = _orb().create_any();
                _version.type(org.omg.CORBA.ORB.init().get_primitive_tc
                    (org.omg.CORBA.TCKind.tk_string));
                _list.add_value("version", _version, org.omg.CORBA.ARG_IN.value);
                org.omg.CORBA.Any _user = _orb().create_any();
                _user.type(org.omg.CORBA.ORB.init().get_primitive_tc(org.omg.CORBA.TCKind.tk_string));
                _list.add_value("user", _user, org.omg.CORBA.ARG_IN.value);
                r.params(_list);
                byte[] psdl_file;
                psdl_file = DistributedCaps.DistCapsPackage.prototype_fileHelper.extract(_psdl_file);
                String name;
                name = _name.extract_string();
                String version;
                version = _version.extract_string();
                String user;
                user = _user.extract_string();
                String __result;
                try {
                    __result = this.translate(psdl_file, name, version, user);
                }
                catch (DistributedCaps.DistCapsPackage.cantWriteFile e0) {
                    org.omg.CORBA.Any _except = _orb().create_any();
                    DistributedCaps.DistCapsPackage.cantWriteFileHelper.insert(_except, e0);
                    r.except(_except);
                    return;
                }
                org.omg.CORBA.Any __result = _orb().create_any();
                __result.insert_string(__result);
                r.result(__result);
            }
            break;
        case 1: // DistributedCaps.DistCaps.get_proto_list
            {
                org.omg.CORBA.NVList _list = _orb().create_list(0);
                org.omg.CORBA.Any _user = _orb().create_any();
                _user.type(org.omg.CORBA.ORB.init().get_primitive_tc(org.omg.CORBA.TCKind.tk_string));
                _list.add_value("user", _user, org.omg.CORBA.ARG_IN.value);
                r.params(_list);
                String user;
            }
    }
}

```



```

user = _user.extract_string();
String[] __result;
__result = this.get_proto_list(user);
org.omg.CORBA.Any _result = _orb().create_any();
DistributedCaps.DistCapsPackage.prototype_listHelper.insert(__result, __result);
r.result(__result);
}
break;
case 2: // DistributedCaps.DistCaps.open_proto
{
org.omg.CORBA.NVList _list = _orb().create_list(0);
org.omg.CORBA.Any _name = _orb().create_any();
_name.type(org.omg.CORBA.ORB.init().get_primitive_tc(org.omg.CORBA.TCKind.tk_string));
_list.add_value("name", _name, org.omg.CORBA.ARG_IN.value);
org.omg.CORBA.Any _user = _orb().create_any();
_user.type(org.omg.CORBA.ORB.init().get_primitive_tc(org.omg.CORBA.TCKind.tk_string));
_list.add_value("user", _user, org.omg.CORBA.ARG_IN.value);
r.params(_list);
String name;
name = _name.extract_string();
String user;
user = _user.extract_string();
byte[] __result;
try {
__result = this.open_proto(name, user);
}
catch (DistributedCaps.DistCapsPackage.cantReadFile e0) {
org.omg.CORBA.Any _except = _orb().create_any();
DistributedCaps.DistCapsPackage.cantReadFileHelper.insert(_except, e0);
r.except(_except);
return;
}
org.omg.CORBA.Any __result = _orb().create_any();
DistributedCaps.DistCapsPackage.prototype_fileHelper.insert(__result, __result);
r.result(__result);
}
break;
case 3: // DistributedCaps.DistCaps.commit
{
org.omg.CORBA.NVList _list = _orb().create_list(0);
org.omg.CORBA.Any psdl_file = _orb().create_any();
_psdl_file.type(DistributedCaps.DistCapsPackage.prototype_fileHelper.type());
_list.add_value("psdl_file", psdl_file, org.omg.CORBA.ARG_IN.value);
org.omg.CORBA.Any _name = _orb().create_any();
_name.type(org.omg.CORBA.ORB.init().get_primitive_tc(org.omg.CORBA.TCKind.tk_string));
_list.add_value("name", _name, org.omg.CORBA.ARG_IN.value);
org.omg.CORBA.Any _version = _orb().create_any();
_version.type(org.omg.CORBA.ORB.init().get_primitive_tc
(org.omg.CORBA.TCKind.tk_string));
_list.add_value("version", _version, org.omg.CORBA.ARG_IN.value);
org.omg.CORBA.Any _user = _orb().create_any();
_user.type(org.omg.CORBA.ORB.init().get_primitive_tc(org.omg.CORBA.TCKind.tk_string));
_list.add_value("user", _user, org.omg.CORBA.ARG_IN.value);
r.params(_list);
byte[] psdl_file;
psdl_file = DistributedCaps.DistCapsPackage.prototype_fileHelper.extract(_psdl_file);

```

```

String name;
name = _name.extract_string();
String version;
version = _version.extract_string();
String user;
user = _user.extract_string();
String __result;
try {
    __result = this.commit(psdl_file, name, version, user);
}
catch (DistributedCaps.DistCapsPackage.cantWriteFile e0) {
    org.omg.CORBA.Any _except = _orb().create_any();
    DistributedCaps.DistCapsPackage.cantWriteFileHelper.insert(_except, e0);
    r.except(_except);
    return;
}
org.omg.CORBA.Any __result = _orb().create_any();
__result.insert_string(__result);
r.result(__result);
}
break;
default:
    throw new org.omg.CORBA.BAD_OPERATION(0,
org.omg.CORBA.CompletionStatus.COMPLETED_MAYBE);
}
}
}

/*
 * File: ./DISTRIBUTEDCAPS/_DISTCAPSSTUB.JAVA
 * From: DISTCAPS.IDL
 * Date: Wed Aug 18 20:38:43 1999
 * By: idltojava Java IDL 1.2 Aug 18 1998 16:25:34
 */

package DistributedCaps;
public class _DistCapsStub
    extends org.omg.CORBA.portable.ObjectImpl
    implements DistributedCaps.DistCaps {

    public _DistCapsStub(org.omg.CORBA.portable.Delegate d) {
        super();
        _set_delegate(d);
    }

    private static final String _type_ids[] = {
        "IDL:DistributedCaps/DistCaps:1.0"
    };

    public String[] _ids() { return (String[]) _type_ids.clone(); }

    // IDL operations
    // Implementation of ::DistributedCaps::DistCaps::translate
    public String translate(byte[] psdl_file, String name, String version, String user)
        throws DistributedCaps.DistCapsPackage.cantWriteFile {

```

```

org.omg.CORBA.Request r = _request("translate");
r.set_return_type(org.omg.CORBA.ORB.init().get_primitive_tc(org.omg.CORBA.TCKind.tk_string));
org.omg.CORBA.Any_psd_file = r.add_in_arg();
DistributedCaps.DistCapsPackage.prototype_fileHelper.insert(_psdl_file, psdl_file);
org.omg.CORBA.Any_name = r.add_in_arg();
_name.insert_string(name);
org.omg.CORBA.Any_version = r.add_in_arg();
_version.insert_string(version);
org.omg.CORBA.Any_user = r.add_in_arg();
_user.insert_string(user);
r.exceptions().add(DistributedCaps.DistCapsPackage.cantWriteFileHelper.type());
r.invoke();
java.lang.Exception __ex = r.env().exception();
if (__ex instanceof org.omg.CORBA.UnknownUserException) {
    org.omg.CORBA.UnknownUserException __userEx = (org.omg.CORBA.UnknownUserException) __ex;
    if (__userEx.except.type().equals(DistributedCaps.DistCapsPackage.cantWriteFileHelper.type())) {
        throw DistributedCaps.DistCapsPackage.cantWriteFileHelper.extract(__userEx.except);
    }
}
String __result;
__result = r.return_value().extract_string();
return __result;
}
//      Implementation of ::DistributedCaps::DistCaps::get_proto_list
public String[] get_proto_list(String user)
{
    org.omg.CORBA.Request r = _request("get_proto_list");
    r.set_return_type(DistributedCaps.DistCapsPackage.prototype_listHelper.type());
    org.omg.CORBA.Any_user = r.add_in_arg();
    _user.insert_string(user);
    r.invoke();
    String[] __result;
    __result = DistributedCaps.DistCapsPackage.prototype_listHelper.extract(r.return_value());
    return __result;
}
//      Implementation of ::DistributedCaps::DistCaps::open_proto
public byte[] open_proto(String name, String user)
throws DistributedCaps.DistCapsPackage.cantReadFile {
    org.omg.CORBA.Request r = _request("open_proto");
    r.set_return_type(DistributedCaps.DistCapsPackage.prototype_fileHelper.type());
    org.omg.CORBA.Any_name = r.add_in_arg();
    _name.insert_string(name);
    org.omg.CORBA.Any_user = r.add_in_arg();
    _user.insert_string(user);
    r.exceptions().add(DistributedCaps.DistCapsPackage.cantReadFileHelper.type());
    r.invoke();
    java.lang.Exception __ex = r.env().exception();
    if (__ex instanceof org.omg.CORBA.UnknownUserException) {
        org.omg.CORBA.UnknownUserException __userEx = (org.omg.CORBA.UnknownUserException) __ex;
        if (__userEx.except.type().equals(DistributedCaps.DistCapsPackage.cantReadFileHelper.type())) {
            throw DistributedCaps.DistCapsPackage.cantReadFileHelper.extract(__userEx.except);
        }
    }
}
byte[] __result;
__result = DistributedCaps.DistCapsPackage.prototype_fileHelper.extract(r.return_value());
return __result;
}

```

```

}
//      Implementation of ::DistributedCaps::DistCaps::commit
public String commit(byte[] psdl_file, String name, String version, String user)
throws DistributedCaps.DistCapsPackage.cantWriteFile {
    org.omg.CORBA.Request r = _request("commit");
    r.set_return_type(org.omg.CORBA.ORB.init().get_primitive_tc(org.omg.CORBA.TCKind.tk_string));
    org.omg.CORBA.Any _psdl_file = r.add_in_arg();
    DistributedCaps.DistCapsPackage.prototype_fileHelper.insert(_psdl_file, psdl_file);
    org.omg.CORBA.Any _name = r.add_in_arg();
    _name.insert_string(name);
    org.omg.CORBA.Any _version = r.add_in_arg();
    _version.insert_string(version);
    org.omg.CORBA.Any _user = r.add_in_arg();
    _user.insert_string(user);
    r.exceptions().add(DistributedCaps.DistCapsPackage.cantWriteFileHelper.type());
    r.invoke();
    java.lang.Exception __ex = r.env().exception();
    if (__ex instanceof org.omg.CORBA.UnknownUserException) {
        org.omg.CORBA.UnknownUserException __userEx = (org.omg.CORBA.UnknownUserException) __ex;
        if (__userEx.except.type().equals(DistributedCaps.DistCapsPackage.cantWriteFileHelper.type())) {
            throw DistributedCaps.DistCapsPackage.cantWriteFileHelper.extract(__userEx.except);
        }
    }
    String __result;
    __result = r.return_value().extract_string();
    return __result;
}
};

/*
* File: ./DISTRIBUTEDCAPS/DISTCAPSHELPER.JAVA
* From: DISTCAPS.IDL
* Date: Wed Aug 18 20:38:43 1999
* By: idtojava Java IDL 1.2 Aug 18 1998 16:25:34
*/

package DistributedCaps;
public class DistCapsHelper {
    // It is useless to have instances of this class
    private DistCapsHelper() { }

    public static void write(org.omg.CORBA.portable.OutputStream out, DistributedCaps.DistCaps that) {
        out.write_Object(that);
    }
    public static DistributedCaps.DistCaps read(org.omg.CORBA.portable.InputStream in) {
        return DistributedCaps.DistCapsHelper.narrow(in.read_Object());
    }
    public static DistributedCaps.DistCaps extract(org.omg.CORBA.Any a) {
        org.omg.CORBA.portable.InputStream in = a.create_input_stream();
        return read(in);
    }
    public static void insert(org.omg.CORBA.Any a, DistributedCaps.DistCaps that) {
        org.omg.CORBA.portable.OutputStream out = a.create_output_stream();
        write(out, that);
    }
}

```

```

    a.read_value(out.create_input_stream(), type());
}
private static org.omg.CORBA.TypeCode _tc;
synchronized public static org.omg.CORBA.TypeCode type() {
    if (_tc == null)
        _tc = org.omg.CORBA.ORB.init().create_interface_tc(id(), "DistCaps");
    return _tc;
}
public static String id() {
    return "IDL:DistributedCaps/DistCaps:1.0";
}
public static DistributedCaps.DistCaps narrow(org.omg.CORBA.Object that)
    throws org.omg.CORBA.BAD_PARAM {
    if (that == null)
        return null;
    if (that instanceof DistributedCaps.DistCaps)
        return (DistributedCaps.DistCaps) that;
    if (!that.is_a(id())) {
        throw new org.omg.CORBA.BAD_PARAM();
    }
    org.omg.CORBA.portable.Delegate dup = ((org.omg.CORBA.portable.ObjectImpl)that)._get_delegate();
    DistributedCaps.DistCaps result = new DistributedCaps._DistCapsStub(dup);
    return result;
}
}
}

```

```

/*
 * File: ./DISTRIBUTEDCAPS/DISTCAPSHOLDER.JAVA
 * From: DISTCAPS.IDL
 * Date: Wed Aug 18 20:38:43 1999
 * By: idltojava Java IDL 1.2 Aug 18 1998 16:25:34
 */

```

```

package DistributedCaps;
public final class DistCapsHolder
    implements org.omg.CORBA.portable.Streamable {
    // instance variable
    public DistributedCaps.DistCaps value;
    // constructors
    public DistCapsHolder() {
        this(null);
    }
    public DistCapsHolder(DistributedCaps.DistCaps __arg) {
        value = __arg;
    }

    public void _write(org.omg.CORBA.portable.OutputStream out) {
        DistributedCaps.DistCapsHelper.write(out, value);
    }

    public void _read(org.omg.CORBA.portable.InputStream in) {
        value = DistributedCaps.DistCapsHelper.read(in);
    }

    public org.omg.CORBA.TypeCode _type() {

```

```
    return DistributedCaps.DistCapsHelper.type();  
  }  
}
```

DistributedCaps

## Interface DistCaps

All Known Implementing Classes:

[DistCapsImplBase](#), [DistCapsStub](#)public interface **DistCaps**extends [org.omg.CORBA.Object](#), [org.omg.CORBA.portable.IDLEntity](#)

### Method Summary

java.lang.String	<a href="#">commit</a> (byte[] psdl_file, java.lang.String name, java.lang.String version, java.lang.String user)
java.lang.String[]	<a href="#">get_proto_list</a> (java.lang.String user)
byte[]	<a href="#">open_proto</a> (java.lang.String name, java.lang.String user)
java.lang.String	<a href="#">translate</a> (byte[] psdl_file, java.lang.String name, java.lang.String version, java.lang.String user)

### Methods inherited from interface [org.omg.CORBA.Object](#)

[\\_create\\_request](#), [\\_create\\_request](#), [\\_duplicate](#), [\\_get\\_domain\\_managers](#),  
[\\_get\\_interface\\_def](#), [\\_get\\_policy](#), [\\_hash](#), [\\_is\\_a](#), [\\_is\\_equivalent](#), [\\_non\\_existent](#),  
[\\_release](#), [\\_request](#), [\\_set\\_policy\\_override](#)

### Method Detail

#### translate

```
public java.lang.String translate(byte[] psdl_file,
                                java.lang.String name,
                                java.lang.String version,
                                java.lang.String user)
    throws DistributedCaps.DistCapsPackage.cantWriteFile
```

## get\_proto\_list

```
public java.lang.String[] get_proto_list(java.lang.String user)
```

---

## open\_proto

```
public byte[] open_proto(java.lang.String name,  
                        java.lang.String user)  
    throws DistributedCaps.DistCapsPackage.cantReadFile
```

---

## commit

```
public java.lang.String commit(byte[] psdl_file,  
                              java.lang.String name,  
                              java.lang.String version,  
                              java.lang.String user)  
    throws DistributedCaps.DistCapsPackage.cantWriteFile
```

---

### **Class** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

---



## **Class** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

DistributedCaps

## Class DistCapsImplBase

java.lang.Object

```
|
+--org.omg.CORBA.portable.ObjectImpl
    |
    +--org.omg.CORBA.DynamicImplementation
        |
        +--DistributedCaps._DistCapsImplBase
```

public abstract class DistCapsImplBase  
extends org.omg.CORBA.DynamicImplementation  
implements [DistCaps](#)

See Also:

[Serialized Form](#)

## Constructor Summary

[DistCapsImplBase](#) ()

## Method Summary

java.lang.String[]	<a href="#">ids</a> ()
void	<a href="#">invoke</a> (org.omg.CORBA.ServerRequest r)

## Methods inherited from class org.omg.CORBA.portable.ObjectImpl

[\\_create\\_request](#), [\\_create\\_request](#), [\\_duplicate](#), [\\_get\\_delegate](#), [\\_get\\_domain\\_managers](#),  
[\\_get\\_interface\\_def](#), [\\_get\\_policy](#), [\\_hash](#), [\\_invoke](#), [\\_is\\_a](#), [\\_is\\_equivalent](#), [\\_is\\_local](#),  
[\\_non\\_existent](#), [\\_orb](#), [\\_release](#), [\\_releaseReply](#), [\\_request](#), [\\_request](#),  
[\\_servant\\_postinvoke](#), [\\_servant\\_preinvoke](#), [\\_set\\_delegate](#), [\\_set\\_policy\\_override](#),  
[equals](#), [hashCode](#), [toString](#)

## Methods inherited from class java.lang.Object

[clone](#), [finalize](#), [getClass](#), [notify](#), [notifyAll](#), [wait](#), [wait](#), [wait](#)

## Constructor Detail

### DistCapsImplBase

```
public DistCapsImplBase()
```

## Method Detail

### ids

```
public java.lang.String[] ids()
```

**Overrides:**

ids in class org.omg.CORBA.portable.ObjectImpl

---

### invoke

```
public void invoke(org.omg.CORBA.ServerRequest r)
```

**Overrides:**

invoke in class org.omg.CORBA.DynamicImplementation

---

## **Class** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

---

## **Class** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

DistributedCaps

## Class DistCapsStub

```
java.lang.Object
|
+--org.omg.CORBA.portable.ObjectImpl
|
+--DistributedCaps._DistCapsStub
```

```
public class DistCapsStub
extends org.omg.CORBA.portable.ObjectImpl
implements DistCaps
```

See Also:

[Serialized Form](#)

### Constructor Summary

[DistCapsStub](#)(org.omg.CORBA.portable.Delegate d)

### Method Summary

java.lang.String[]	<a href="#">ids</a> ()
java.lang.String	<a href="#">commit</a> (byte[] psdl_file, java.lang.String name, java.lang.String version, java.lang.String user)
java.lang.String[]	<a href="#">get proto list</a> (java.lang.String user)
byte[]	<a href="#">open proto</a> (java.lang.String name, java.lang.String user)
java.lang.String	<a href="#">translate</a> (byte[] psdl_file, java.lang.String name, java.lang.String version, java.lang.String user)

## Methods inherited from class org.omg.CORBA.portable.ObjectImpl

`_create_request, _create_request, _duplicate, _get_delegate, _get_domain_managers, _get_interface_def, _get_policy, _hash, _invoke, _is_a, _is_equivalent, _is_local, _non_existent, _orb, _release, _releaseReply, _request, _request, _servant_postinvoke, _servant_preinvoke, _set_delegate, _set_policy_override, equals, hashCode, toString`

## Methods inherited from class java.lang.Object

`clone, finalize, getClass, notify, notifyAll, wait, wait, wait`

## Constructor Detail

### \_DistCapsStub

```
public _DistCapsStub(org.omg.CORBA.portable.Delegate d)
```

## Method Detail

### \_ids

```
public java.lang.String[] _ids()
```

#### Overrides:

\_ids in class org.omg.CORBA.portable.ObjectImpl

---

### translate

```
public java.lang.String translate(byte[] psdl_file,  
                                   java.lang.String name,  
                                   java.lang.String version,  
                                   java.lang.String user)  
    throws DistributedCaps.DistCapsPackage.cantWriteFile
```

#### Specified by:

translate in interface DistCaps

---

### get\_proto\_list

```
public java.lang.String[] get_proto_list(java.lang.String user)
```

#### Specified by:

get\_proto\_list in interface DistCaps

---

## open\_proto

```
public byte[] open_proto(java.lang.String name,  
                        java.lang.String user)  
    throws DistributedCaps.DistCapsPackage.cantReadFile
```

### Specified by:

[open\\_proto](#) in interface [DistCaps](#)

---

## commit

```
public java.lang.String commit(byte[] psdl_file,  
                              java.lang.String name,  
                              java.lang.String version,  
                              java.lang.String user)  
    throws DistributedCaps.DistCapsPackage.cantWriteFile
```

### Specified by:

[commit](#) in interface [DistCaps](#)

---

## **Class** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

---

DistributedCaps

## Class DistCapsHelper

java.lang.Object

|  
+--DistributedCaps.DistCapsHelper

```
public class DistCapsHelper
extends java.lang.Object
```

### Method Summary

static <a href="#">DistCaps</a>	<a href="#">extract</a> (org.omg.CORBA.Any a)
static java.lang.String	<a href="#">id</a> ()
static void	<a href="#">insert</a> (org.omg.CORBA.Any a, <a href="#">DistCaps</a> that)
static <a href="#">DistCaps</a>	<a href="#">narrow</a> (org.omg.CORBA.Object that)
static <a href="#">DistCaps</a>	<a href="#">read</a> (org.omg.CORBA.portable.InputStream in)
static org.omg.CORBA.TypeCode	<a href="#">type</a> ()
static void	<a href="#">write</a> (org.omg.CORBA.portable.OutputStream out, <a href="#">DistCaps</a> that)

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

### Method Detail

**write**

```
public static void write(org.omg.CORBA.portable.OutputStream out,
                        DistCaps that)
```

---

## read

```
public static DistCaps read(org.omg.CORBA.portable.InputStream in)
```

---

## extract

```
public static DistCaps extract(org.omg.CORBA.Any a)
```

---

## insert

```
public static void insert(org.omg.CORBA.Any a,
                          DistCaps that)
```

---

## type

```
public static org.omg.CORBA.TypeCode type()
```

---

## id

```
public static java.lang.String id()
```

---

## narrow

```
public static DistCaps narrow(org.omg.CORBA.Object that)
    throws org.omg.CORBA.BAD_PARAM
```

---

### **Class** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[SUMMARY](#): [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

[DETAIL](#): [FIELD](#) | [CONSTR](#) | [METHOD](#)

---

DistributedCaps

## Class DistCapsHolder

java.lang.Object

|--DistributedCaps.DistCapsHolder

```
public final class DistCapsHolder
extends java.lang.Object
implements org.omg.CORBA.portable.Streamable
```

### Field Summary

<a href="#">DistCaps</a>	<a href="#">value</a>
--------------------------	-----------------------

### Constructor Summary

[DistCapsHolder\(\)](#)[DistCapsHolder\(DistCaps \\_\\_arg\)](#)

### Method Summary

void	<a href="#">read</a> (org.omg.CORBA.portable.InputStream in)
------	--

org.omg.CORBA.TypeCode	<a href="#">type</a> ()
------------------------	-------------------------

void	<a href="#">write</a> (org.omg.CORBA.portable.OutputStream out)
------	---

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



## Field Detail

### value

```
public DistCaps value
```

## Constructor Detail

### DistCapsHolder

```
public DistCapsHolder()
```

### DistCapsHolder

```
public DistCapsHolder(DistCaps __arg)
```

## Method Detail

### write

```
public void write(org.omg.CORBA.portable.OutputStream out)
```

**Specified by:**

write in interface org.omg.CORBA.portable.Streamable

### read

```
public void read(org.omg.CORBA.portable.InputStream in)
```

**Specified by:**

read in interface org.omg.CORBA.portable.Streamable

### type

```
public org.omg.CORBA.TypeCode type()
```

**Specified by:**

type in interface org.omg.CORBA.portable.Streamable

## **Class** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

THIS PAGE INTENTIONALLY LEFT BLANK

## APPENDIX D: MODIFIED HSI SOURCE CODE

This appendix contains the HSI source files that were modified from [DURA99] and the corresponding Javadoc generated documentation. Changes from the original source code are bolded.

```
/**
 * The main CAPS window.
 *
 * @author Ilker DURANLIOGLU, modified by Gary Kreeger
 * @version 1.1
 */

package caps.CAPSMMain;

import java.awt.*;
import javax.swing.*;
import java.io.File;
import caps.Builder.PsdlBuilder;
import caps.Psdl.Vertex;
import caps.Psdl.DataTypes;
import caps.GraphEditor.Editor;
import java.awt.event.*;
import java.util.Vector;
import java.util.Enumeration;
import DistributedCaps.*;

public class CAPSMMainWindow extends JFrame {

    /**
     * The width of the frame.
     */

    private final int WIDTH = 400;

    /**
     * The height of the frame.
     */

    private final int HEIGHT = 150;

    /**
     * The File that contains the PSDL prototype.
     */

    private File prototype;
```

```

/**
 * The object reference to the CAPS server.
 */

private DistCaps capsRef;

/**
 * The Vector that holds references to the open prototypes
 */

private static Vector openPrototypes;

//THE FOLLOWING ATTRIBUTES WERE SUGGESTED BY PROFESSOR SHING

/**
 * The private attribute for holding protoName.
 */

private static String protoName;

/**
 * The private attribute for holding protoVersion.
 */

private static String protoVersion;

/**
 * The private attribute for holding capsUser.
 */

private static String capsUser;

/**
 * The constructor for this class.
 *
 * @param objRef The reference to the CAPS object on the server
 */

public CAPSMainWindow (DistCaps objRef)
{
    super ("HSI Designer Mode");    // The title of the frame.

    prototype = null;

    capsRef = objRef; //reference to server object

    capsUser = System.getProperty("CAPSUser"); // session user

    openPrototypes = new Vector (0, 2);

```

```

    initialize ();
}

/**
 * Initializes the CAPS main window.
 */

public void initialize ()
{
    setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
    addWindowListener (new ExitCAPSMain (this));

    //Places the frame in the upper-right corner of the screen

    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    setLocation(screenSize.width - (WIDTH + WIDTH / 2), HEIGHT / 2);

    setResizable (false);

    setJMenuBar (new CAPSMainMenuBar (this));

    JPanel panel = new JPanel ();

    JLabel capsLabel = new JLabel ("Heterogeneous System Integrator");
    capsLabel.setFont (new Font ("Courier", Font.BOLD, 17));
    JLabel imageLabel = new JLabel (new ImageIcon ("caps/Images/caps.gif"));

    panel.add (Box.createHorizontalStrut (5));
    panel.add (imageLabel);
    panel.add (Box.createHorizontalStrut (5));
    panel.add (capsLabel);
    panel.add (Box.createHorizontalStrut (5));

    getContentPane ().add (panel);

    pack ();

    setVisible (true);
}

/**
 * Sets the prototype file to the argument.
 *
 * @param f The File that contains the PSDL prototype.
 */

public void setPrototype (File f)
{
    prototype = f;
}

```

```

/**
 * Sets the label to the string message passed in.
 *
 * @param msg The message to be displayed.
 */

public void setLabel (String msg)
{
    JOptionPane.showMessageDialog (this, msg, "Information Message",
        JOptionPane.INFORMATION_MESSAGE);
}

/**
 * Retrieves the curent prototype file.
 *
 * @return the File that contains the PSDL prototype.
 */

public File getPrototype ()
{
    return prototype;
}

/**
 * Retrieves the curent reference to the caps server.
 *
 * @return the reference to the caps server object.
 */

public DistCaps getCapsRef ()
{
    return capsRef;
}

/**
 * Returns the vector that holds the open prototype files.
 *
 * @return the vector that contains the open prototype files.
 */

public Vector getOpenPrototypes ()
{
    return openPrototypes;
}

/**
 * Sets the prototype name to the argument
 *
 * @param name The string that contains the prototype's name
 */

public void setProtoName (String name)

```

```

{
    protoName = name;
}

/**
 * Sets the prototype version to the argument
 *
 * @param version The string that contains the prototype's version
 */

public void setProtoVersion (String version)
{
    protoVersion = version;
}

/**
 * Gets the prototype name
 *
 */
public String getProtoName ()
{
    return protoName;
}

/**
 * Gets the prototype version
 *
 */

public String getProtoVersion ()
{
    return protoVersion;
}

/**
 * Gets the capsUser
 *
 */

public String getCapsUser ()
{
    return capsUser;
}

/**
 * Opens the graphics editor to edit a prototype.
 */

public void editPrototype ()
{
    if (prototype == null) { // No prototype is selected to open
        JOptionPane.showMessageDialog (this, "No prototype is selected to edit.",

```

```

                "Error Message", JOptionPane.ERROR_MESSAGE);
    }
    else if (!isPrototypeChanged ()) { // Attempt to edit the same prototype.
        JOptionPane.showMessageDialog (this, new String ("Prototype " + prototype.getName () +
            " is already open."),
            "Error Message", JOptionPane.ERROR_MESSAGE);
    }
    else {
        PsdlBuilder.disable_tracing (); // Disable debug messages
        Vertex root = null;
        root = PsdlBuilder.buildPrototype (prototype);
        if (root == null) {
            root = new Vertex (0, 0, null, false); // If this is a new prototype
            String name = prototype.getName (); // Prototype name is the same as
            root.setLabel (name.substring (0, name.length () - 5)); // the file name
        }
        DataTypes types = new DataTypes ();
        types.buildTypes (prototype);
        Editor e = new Editor (prototype, root, types);
        new Thread (e).start ();
        openPrototypes.addElement (e);
    }
}

/**
 * Checks whether or not the current prototype file is already used by
 * a PSDL Editor.
 *
 * @return true if selected prototype is not already open
 */

public boolean isPrototypeChanged ()
{
    for (Enumeration enum = openPrototypes.elements (); enum.hasMoreElements ();) {
        Editor e = (Editor) enum.nextElement ();
        if (prototype.equals (e.getPrototypeFile ()))
            return false;
    }
    return true;
}

/**
 * Removes one element from the openPrototypes vector.
 *
 * @param e the editor that is going to be removed from the vector.
 */

public static void removeEditor (Editor e)
{
    openPrototypes.removeElement (e);
}

/**

```



```

* Checks if the status of any of the open prototypes is 'saveRequired'.
* Prompts the user to save the prototype.
*
* @return true if none of the prototypes need saving.
*/

public boolean isOpenPrototypeSaved ()
{
    boolean flag = true;
    Editor e;
    label :
    for (Enumeration enum = openPrototypes.elements ();enum.hasMoreElements ();) {
        e = (Editor) enum.nextElement ();
        if (e.isSaveRequired ()) {
            int ix = JOptionPane.showConfirmDialog (this, new String ("Save changes to the prototype " +
                e.getRoot ().getLabel () + "?"));
            if (ix == JOptionPane.CANCEL_OPTION) {
                flag = false;
                break label;
            }
            else if (ix == JOptionPane.YES_OPTION)
                e.savePrototype ();
        }
    }
    return flag;
}

} // End of the class CAPSMainWindow

/**
 * This class holds the 'Exec Support' menu items.
 *
 * @author Ilker DURANLIOGLU, modified by Gary Kreeger
 *
 * @version 1.1
 */

package caps.CAPSMain;

import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.*;
import java.io.IOException;
import javax.swing.*;
import java.io.*;
import DistributedCaps.*;

public class ExecSupportMenu extends JMenu implements ActionListener {

    /**
     * Initiates the 'Translate' event
     */
    private JMenuItem translateMenuItem = new JMenuItem ("Translate");

```

```

/**
 * Initiates the 'Schedule' event
 */
private JMenuItem scheduleMenuItem = new JMenuItem ("Schedule");

/**
 * Initiates the 'Compile' event
 */
private JMenuItem compileMenuItem = new JMenuItem ("Compile");

/**
 * Initiates the 'Execute' event
 */
private JMenuItem executeMenuItem = new JMenuItem ("Execute");

/**
 * The main window which owns this menu.
 */
protected CAPSMainWindow ownerWindow;

/**
 * Constructor for this class.
 *
 * @param owner CAPSMainWindow that owns the menu
 */
public ExecSupportMenu (CAPSMainWindow owner)
{
    super ("Exec Support");

    ownerWindow = owner;

    add (translateMenuItem);
    add (scheduleMenuItem);
    add (compileMenuItem);
    add (executeMenuItem);

    //Register the action listeners

    translateMenuItem.addActionListener (this);
    scheduleMenuItem.addActionListener (this);
    compileMenuItem.addActionListener (this);
    executeMenuItem.addActionListener (this);
} // end of ExecSupportMenu constructor

/**
 * Action event handler for the menu events.
 *
 * @param e The action event that is created by selecting a menu item
 */
public void actionPerformed(ActionEvent e)
{

```

```

if (e.getSource () == translateMenuItem) {

    if (ownerWindow.getPrototype() == null) // No prototype is selected to open
    {
        JOptionPane.showMessageDialog (ownerWindow,
            "No prototype is selected to edit.", "Error Message", JOptionPane.ERROR_MESSAGE);
    }
    else
    {
        try
        {
            File proto = ownerWindow.getPrototype();

            FileInputStream in = new FileInputStream (proto);
            byte[] fileBuf = new byte[in.available()];
            in.read (fileBuf);

            DistCaps capsRef = ownerWindow.getCapsRef();

            String returnMsg = capsRef.translate(fileBuf, ownerWindow.getProtoName(),
                ownerWindow.getProtoVersion(), ownerWindow.getCapsUser());

            ownerWindow.setLabel (returnMsg);
            System.out.println (returnMsg);

        }
        catch (Exception e1)
        {
            System.err.println ("Error: " + e1);
            e1.printStackTrace (System.out);
        }
    }
}
else if (e.getSource () == scheduleMenuItem) {
    System.out.println ("Schedule has not been implemented yet");
}
else if (e.getSource () == compileMenuItem) {
    System.out.println ("Compile has not been implemented yet");
}
else if (e.getSource () == executeMenuItem) {
    System.out.println ("Executing telnet");
    try {
        Runtime run = Runtime.getRuntime ();
        run.exec ("telnet.exe");
    } catch (IOException ex) {
        System.out.println (ex);
    }
}
} // end of actionPerformed

} // End of the class ExecSupportMenu

```

```

/**
 * This class holds the 'Prototype' menu items.
 *
 * @author Ilker DURANLIOGLU, modified by Gary Kreeger
 *
 * @version 1.1
 */

package caps.CAPSMain;

import javax.swing.*;
import javax.swing.filechooser.FileSystemView;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.*;
import java.util.Vector;
import DistributedCaps.*;

public class PrototypeMenu extends JMenu implements ActionListener {

    /**
     * Initiates the 'New' event
     */

    private JMenuItem newMenuItem = new JMenuItem ("New");

    /**
     * Initiates the 'Open' event
     */

    private JMenuItem openMenuItem = new JMenuItem ("Open");

    /**
     * Initiates the 'Commit Work' event
     */

    private JMenuItem commitWorkMenuItem = new JMenuItem ("Commit Work");

    /**
     * Initiates the 'Retrieve From DDB' event
     */

    private JMenuItem retrieveMenuItem = new JMenuItem ("Retrieve From DDB");

    /**
     * Initiates the 'Quit' event
     */

    private JMenuItem quitMenuItem = new JMenuItem ("Quit");

```

```

/**
 * The main window which owns this menu.
 */

protected CAPSMainWindow ownerWindow;

/**
 * Constructor for this class.
 *
 * @param owner The main window which has created this menu.
 */

public PrototypeMenu (CAPSMainWindow owner)
{
    super ("Prototype");

    ownerWindow = owner;

    add (newMenuItem);
    add (openMenuItem);
    add (commitWorkMenuItem);
    add (retrieveMenuItem);
    add (quitMenuItem);

    //Register the action listeners

    newMenuItem.addActionListener (this);
    openMenuItem.addActionListener (this);
    commitWorkMenuItem.addActionListener (this);
    retrieveMenuItem.addActionListener (this);
    quitMenuItem.addActionListener (this);
} // end PrototypeMenu constructor

/**
 * Action event handler for the menu events.
 *
 * @param e The action event that is created by selecting a menu item from this menu
 */

public void actionPerformed(ActionEvent e)
{
    if (e.getSource () == newMenuItem) {
        processNewMenuItem ();
    }
    else if (e.getSource () == openMenuItem) {
        processOpenMenuItem ();
    }
    else if (e.getSource () == commitWorkMenuItem) {
        processCommitWorkMenuItem();
    }
    else if (e.getSource () == retrieveMenuItem) {
        System.out.println ("Retrieve has not yet been implemented");
    }
}

```

```

    }
    else if (e.getSource () == quitMenuItem) {
        // Exit the program if all of the prototypes are saved.
        if (ownerWindow.isOpenPrototypeSaved ())
            System.exit (0);
    }
} //end of actionPerformed

/**
 * Handles the event which is caused by selecting the 'New' menu item.
 */

public void processNewMenuItem ()
{
    // The system property for the home prototype directory.
    String protoHome = System.getProperty ("PROTOTYPEHOME");
    File protoDir;
    if (protoHome == null) { // If it is not set as a command line argument
        File homeDir = FileSystemView.getFileSystemView ().getHomeDirectory ();
        protoHome = new String (homeDir + File.separator + ".caps");
        protoDir = new File (protoHome);
        if (!protoDir.exists ())
            protoDir.mkdir ();
    }
    else {
        protoDir = new File (protoHome);
        if (!protoDir.exists ())
            protoDir.mkdir ();
    }

    String proto = JOptionPane.showInputDialog (ownerWindow,
        "Enter Prototype Name : ", "New", JOptionPane.PLAIN_MESSAGE);
    if (proto == null)
        return;

    String version = JOptionPane.showInputDialog (ownerWindow,
        "Prototype Version Information : ", "New", JOptionPane.PLAIN_MESSAGE);
    {
        String name = proto;

        File file = new File (protoHome + File.separator + proto + File.separator + version +
            File.separator + name + ".psd");
        if (file.exists ()) {
            int selected = JOptionPane.showConfirmDialog (ownerWindow, "Selected prototype file already exists.\n" +
                "Do you want to overwrite it ?");
            if (selected == JOptionPane.YES_OPTION) {
                try {
                    file.delete ();
                    file.createNewFile ();
                } catch (java.io.IOException ex) {
                    System.out.println (ex);
                }
                ownerWindow.setPrototype (file);
            }
        }
    }
}

```

```

else {
    try {
        File dir = file.getParentFile ().getParentFile ();
        dir.mkdir ();
        File vers = file.getParentFile ();
        vers.mkdir ();
        file.createNewFile ();
    } catch (java.io.IOException ex) {
        System.out.println (ex);
    }
    ownerWindow.setPrototype (file);
    ownerWindow.setProtoName (proto);
    ownerWindow.setProtoVersion (version);
}
}
} // end of processNewMenuItem

```

```

/**
 * Handles the event which is caused by selecting the 'Open' menu item.
 */

```

```

public void processOpenMenuItem ()
{
    Object[] possibleValues = { "Local", "Remote"};
    String selectedValue = (String) JOptionPane.showInputDialog(null,
        "Please choose the prototype source", "Input",
        JOptionPane.INFORMATION_MESSAGE, null, possibleValues, possibleValues[0]);

    if (selectedValue == "Local")
    {
        String protoHome = System.getProperty ("PROTOTYPEHOME");
        File protoDir;
        if (protoHome == null) // If it is not set as a command line argument
        {
            File homeDir = FileSystemView.getFileSystemView ().getHomeDirectory ();
            protoHome = new String (homeDir + File.separator + ".caps");
            protoDir = new File (protoHome);
            if (!protoDir.exists ())
            {
                protoDir.mkdir ();
            }
        }
        else
        {
            protoDir = new File (protoHome);
            if (!protoDir.exists ())
                protoDir.mkdir ();
        }

        Vector prototypeNames = new Vector (0, 2);
        File [] dirs = protoDir.listFiles ();
        String protoName = "";

        if (dirs.length == 0)
        {

```

```

JOptionPane.showMessageDialog (ownerWindow, "No prototype is found to open",
                                "Error Message", JOptionPane.ERROR_MESSAGE);
}
else
{
for (int ix = 0; ix < dirs.length; ix++)
{
protoName = dirs [ix].getName ();
File subDirs [] = dirs [ix].listFiles ();
for (int jx = 0; jx < subDirs.length; jx++)
{
prototypeNames.addElement (protoName.concat
                             (File.separator + subDirs [jx].getName ()));
}
}
}

Object [] protos = prototypeNames.toArray ();
String selected = (String) JOptionPane.showInputDialog (ownerWindow, "Select a prototype : ",
                                                         "Open", JOptionPane.INFORMATION_MESSAGE, null, protos, protos [0]);

if (selected != null)
{
File selectedDir = new File (protoHome + File.separator + selected);
File file = new File (selectedDir.getAbsolutePath () + File.separator +
                     selectedDir.getParentFile ().getName () + ".psdl");
if (!file.exists ())
{
JOptionPane.showMessageDialog (ownerWindow,
                                "The selected prototype file cannot be opened",
                                "Error Message", JOptionPane.ERROR_MESSAGE);
}
ownerWindow.setPrototype (file);
ownerWindow.setProtoName (selectedDir.getParentFile ().getName ());
ownerWindow.setProtoVersion (selectedDir.getName ());
}
}
else // open prototype from remote source
{
DistCaps capsRef = ownerWindow.getCapsRef();

//get the list of available prototypes
String [] proto_list = capsRef.get_proto_list(ownerWindow.getCapsUser());

byte[] proto_file;
String selected = (String) JOptionPane.showInputDialog (ownerWindow,
                                                         "Select a prototype : ", "Open",
                                                         JOptionPane.INFORMATION_MESSAGE, null,
                                                         proto_list, proto_list [0]);

try
{
//open the selected file from the remote server
proto_file = capsRef.open_proto (selected, ownerWindow.getCapsUser());

if (proto_file == null)

```



```

{
    JOptionPane.showMessageDialog (ownerWindow,
        "The selected prototype file cannot be opened",
        "Error Message", JOptionPane.ERROR_MESSAGE);
}
else
{
    try //convert the selected file from a byte array back to a file
    {
        String protoHome = System.getProperty ("PROTOTYPEHOME");
        File protoDir;
        if (protoHome == null) // If it is not set as a command line argument
        {
            File homeDir = FileSystemView.getFileSystemView ().getHomeDirectory ();
            protoHome = new String (homeDir + File.separator + ".caps");
            protoDir = new File (protoHome);
            if (!protoDir.exists ())
            {
                protoDir.mkdirs ();
            }
        }
        else
        {
            protoDir = new File (protoHome);
            if (!protoDir.exists ())
            {
                protoDir.mkdirs ();
            }
        }

        File selectedDir = new File (protoHome + File.separator + selected);
        if (!selectedDir.exists ())
        {
            selectedDir.mkdirs ();
        }
        else
        {
            ownerWindow.setLabel ("The remote file will overwrite an existing local one");
        }

        File file = new File (selectedDir.getAbsolutePath () + File.separator +
            selectedDir.getParentFile ().getName () + ".psdl");
        boolean tempBoolean = file.createNewFile();

        FileOutputStream fos = new FileOutputStream (file);
        fos.write (proto_file);
        fos.close();
        ownerWindow.setPrototype (file);
        ownerWindow.setProtoName (selectedDir.getParentFile ().getName ());
        ownerWindow.setProtoVersion (selectedDir.getName ());
    }
    catch (Exception e)
    {
        JOptionPane.showMessageDialog (ownerWindow,
            "The selected prototype file was retrieved but cannot be opened",
            "Error Message", JOptionPane.ERROR_MESSAGE);
    }
}

```



```
{
    ownerWindow.setLabel ("You must save the prototype before committing it");
}
}

setCursor (new Cursor(Cursor.DEFAULT_CURSOR));

} //end of processCommitWorkMenuItem

} // End of the class PrototypeMenu
```

caps.CAPSMMain

## Class CAPSMMainWindow

```
java.lang.Object
|
+--java.awt.Component
|
+--java.awt.Container
|
+--java.awt.Window
|
+--java.awt.Frame
|
+--javax.swing.JFrame
|
+--caps.CAPSMMain.CAPSMMainWindow
```

```
public class CAPSMMainWindow
extends javax.swing.JFrame
```

The main CAPS window.

**See Also:**

[Serialized Form](#)

**Inner classes inherited from class javax.swing.JFrame**

`javax.swing.JFrame.AccessibleJFrame`

**Inner classes inherited from class java.awt.Component**

`java.awt.Component.AWTTreeLock`

## Field Summary

<code>private DistributedCaps.DistCaps</code>	<u>capsRef</u> The object reference to the CAPS server.
<code>private static java.lang.String</code>	<u>capsUser</u> The private attribute for holding capsUser.
<code>private int</code>	<u>HEIGHT</u> The height of the frame.
<code>private static java.util.Vector</code>	<u>openPrototypes</u> The Vector that holds references to the open prototypes
<code>private static java.lang.String</code>	<u>protoName</u> The private attribute for holding protoName.
<code>private java.io.File</code>	<u>prototype</u> The File that contains the PSDL prototype.
<code>private static java.lang.String</code>	<u>protoVersion</u> The private attribute for holding protoVersion.
<code>private int</code>	<u>WIDTH</u> The width of the frame.

### Fields inherited from class javax.swing.JFrame

accessibleContext, defaultCloseOperation, rootPane, rootPaneCheckingEnabled

### Fields inherited from class java.awt.Frame

base, CROSSHAIR\_CURSOR, DEFAULT\_CURSOR, E\_RESIZE\_CURSOR, frameSerializedDataVersion, HAND\_CURSOR, icon, ICONIFIED, mbManagement, menuBar, MOVE\_CURSOR, N\_RESIZE\_CURSOR, nameCounter, NE\_RESIZE\_CURSOR, NORMAL, NW\_RESIZE\_CURSOR, ownedWindows, resizable, S\_RESIZE\_CURSOR, SE\_RESIZE\_CURSOR, serialVersionUID, state, SW\_RESIZE\_CURSOR, TEXT\_CURSOR, title, W\_RESIZE\_CURSOR, WAIT\_CURSOR, weakThis

### Fields inherited from class java.awt.Window

active, base, focusMgr, inputContext, nameCounter, OPENED, ownedWindowList, serialVersionUID, state, warningString, weakThis, windowListener, windowSerializedDataVersion

### Fields inherited from class java.awt.Container

component, containerListener, containerSerializedDataVersion, dispatcher, layoutMgr, maxSize, ncomponents, serialVersionUID

## Fields inherited from class java.awt.Component

actionListenerK, adjustmentListenerK, appContext, assert, background, BOTTOM\_ALIGNMENT, CENTER\_ALIGNMENT, changeSupport, componentListener, componentListenerK, componentOrientation, componentSerializedDataVersion, containerListenerK, cursor, dropTarget, enabled, eventMask, focusListener, focusListenerK, font, foreground, hasFocus, height, incRate, inputMethodListener, inputMethodListenerK, isInc, isPacked, itemListenerK, keyListener, keyListenerK, LEFT\_ALIGNMENT, locale, LOCK, minSize, mouseListener, mouseListenerK, mouseMotionListener, mouseMotionListenerK, name, nameExplicitlySet, newEventsOnly, ownedWindowK, parent, peer, peerFont, popups, prefSize, RIGHT\_ALIGNMENT, serialVersionUID, textListenerK, TOP\_ALIGNMENT, valid, visible, width, windowListenerK, x, y

## Constructor Summary

CAPSMainWindow(DistributedCaps.DistCaps objRef)

The constructor for this class.

## Method Summary

void	<u>editPrototype</u> () Opens the graphics editor to edit a prototype.
DistributedCaps.DistCaps	<u>getCapsRef</u> () Retrieves the current reference to the caps server.
java.lang.String	<u>getCapsUser</u> () Gets the capsUser
java.util.Vector	<u>getOpenPrototypes</u> () Returns the vector that holds the open prototype files.
java.lang.String	<u>getProtoName</u> () Gets the prototype name
java.io.File	<u>getPrototype</u> () Retrieves the current prototype file.
java.lang.String	<u>getProtoVersion</u> () Gets the prototype version
void	<u>initialize</u> () Initializes the CAPS main window.
boolean	<u>isOpenPrototypeSaved</u> () Checks if the status of any of the open prototypes is 'saveRequired'.
boolean	<u>isPrototypeChanged</u> () Checks whether or not the current prototype file is already used by a PSDL Editor.
static void	<u>removeEditor</u> (caps.GraphEditor.Editor e) Removes one element from the openPrototypes vector.
void	<u>setLabel</u> (java.lang.String msg) Sets the label to the string message passed in.
void	<u>setProtoName</u> (java.lang.String name)

	Sets the prototype name to the argument
void	<u>setPrototype</u> (java.io.File f) Sets the prototype file to the argument.
void	<u>setProtoVersion</u> (java.lang.String version) Sets the prototype version to the argument

### Methods inherited from class javax.swing.JFrame

addImpl, createRootPane, createRootPaneException, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getJMenuBar, getLayeredPane, getRootPane, isRootPaneCheckingEnabled, paramString, processKeyEvent, processWindowEvent, remove, setContentPane, setDefaultCloseOperation, setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, update

### Methods inherited from class java.awt.Frame

, addNotify, addToFrameList, constructComponentName, finalize, getCursorType, getFrames, getIconImage, getMenuBar, getState, getTitle, initIDs, isResizable, postProcessKeyEvent, readObject, remove, removeFromFrameList, removeNotify, setCursor, setIconImage, setMenuBar, setResizable, setState, setTitle, writeObject

### Methods inherited from class java.awt.Window

addOwnedWindow, addWindowListener, applyResourceBundle, applyResourceBundle, connectOwnedWindow, dispatchEventImpl, dispose, eventEnabled, getFocusOwner, getInputContext, getLocale, getOwnedWindows, getOwner, getToolkit, getWarningString, hide, isActive, isShowing, nextFocus, ownedInit, pack, postEvent, postWindowEvent, preProcessKeyEvent, processEvent, removeOwnedWindow, removeWindowListener, setCursor, setFocusOwner, setWarningString, show, toBack, toFront, transferFocus

### Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, applyOrientation, countComponents, deliverEvent, dispatchEventToSelf, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents\_NoClientCode, getComponents, getCursorTarget, getInsets, getLayout, getMaximumSize, getMinimumSize, getMouseEventTarget, getPreferredSize, getWindow, insets, invalidate, invalidateTree, isAncestorOf, layout, lightweightPrint, list, list, locate, minimumSize, paint, paintComponents, postsOldMouseEvent, preferredSize, print, printComponents, printOneComponent, processContainerEvent, proxyEnableEvents, proxyRequestFocus, remove, removeAll, removeContainerListener, setFont, updateCursor, validate, validateTree

## Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addPropertyChangeListener, addPropertyChangeListener, areInputMethodsEnabled, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont\_NoClientCode, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getInputMethodRequests, getIntrinsicCursor, getLocation, getLocation, getLocationOnScreen, getName, getNativeContainer, getParent\_NoClientCode, getParent, getPeer, getSize, getSize, getToolkitImpl, getTreeLock, getWidth, getWindowForObject, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isDisplayable, isDoubleBuffered, isEnabled, isEnabledImpl, isFocusTraversable, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processInputMethodEvent, processMouseEvent, processMouseMotionEvent, removeComponentListener, removeFocusListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setForeground, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus

## Methods inherited from class java.lang.Object

clone, equals, getClass, hashCode, notify, notifyAll, registerNatives, wait, wait, wait

## Field Detail

### WIDTH

private final int WIDTH

The width of the frame.

### HEIGHT

private final int HEIGHT

The height of the frame.

### prototype

private java.io.File prototype

The File that contains the PSDL prototype.



---

## capsRef

private DistributedCaps.DistCaps capsRef

The object reference to the CAPS server.

---

## protoName

private static java.lang.String protoName

The private attribute for holding protoName.

---

## protoVersion

private static java.lang.String protoVersion

The private attribute for holding protoVersion.

---

## capsUser

private static java.lang.String capsUser

The private attribute for holding capsUser.

---

## openPrototypes

private static java.util.Vector openPrototypes

The Vector that holds references to the open prototypes

---

## Constructor Detail

### CAPSMainWindow

public CAPSMainWindow(DistributedCaps.DistCaps objRef)

The constructor for this class.

**Parameters:**

objRef - The reference to the CAPS object on the server

## Method Detail

### initialize

```
public void initialize()
```

Initializes the CAPS main window.

---

### setPrototype

```
public void setPrototype(java.io.File f)
```

Sets the prototype file to the argument.

**Parameters:**

f - The File that contains the PSDL prototype.

---

### setLabel

```
public void setLabel(java.lang.String msg)
```

Sets the label to the string message passed in.

**Parameters:**

msg - The message to be displayed.

---

### getPrototype

```
public java.io.File getPrototype()
```

Retrieves the current prototype file.

**Returns:**

the File that contains the PSDL prototype.

---

### getCapsRef

```
public DistributedCaps.DistCaps getCapsRef()
```

Retrieves the current reference to the caps server.

**Returns:**

the reference to the caps server object.

---

### getOpenPrototypes

```
public java.util.Vector getOpenPrototypes()
```

Returns the vector that holds the open prototype files.

**Returns:**

the vector that contains the open prototype files.

---

### **setProtoName**

```
public void setProtoName(java.lang.String name)
```

Sets the prototype name to the argument

**Parameters:**

name - The string that contains the prototype's name

---

### **setProtoVersion**

```
public void setProtoVersion(java.lang.String version)
```

Sets the prototype version to the argument

**Parameters:**

version - The string that contains the prototype's version

---

### **getProtoName**

```
public java.lang.String getProtoName()
```

Gets the prototype name

---

### **getProtoVersion**

```
public java.lang.String getProtoVersion()
```

Gets the prototype version

---

### **getCapsUser**

```
public java.lang.String getCapsUser()
```

Gets the capsUser

---

## **editPrototype**

```
public void editPrototype()
```

Opens the graphics editor to edit a prototype.

---

## **isPrototypeChanged**

```
public boolean isPrototypeChanged()
```

Checks whether or not the current prototype file is already used by a PSDL Editor.

**Returns:**

true if selected prototype is not already open

---

## **removeEditor**

```
public static void removeEditor(caps.GraphEditor.Editor e)
```

Removes one element from the openPrototypes vector.

**Parameters:**

e - the editor that is going to be removed from the vector.

---

## **isOpenPrototypeSaved**

```
public boolean isOpenPrototypeSaved()
```

Checks if the status of any of the open prototypes is 'saveRequired'. Prompts the user to save the prototype.

**Returns:**

true if none of the prototypes need saving.

---

### **Class** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

---

caps.CAPSMain

## Class ExecSupportMenu

java.lang.Object

|  
+---java.awt.Component|  
+---java.awt.Container|  
+---javax.swing.JComponent|  
+---javax.swing.AbstractButton|  
+---javax.swing.JMenuItem|  
+---javax.swing.JMenu|  
+---caps.CAPSMain.ExecSupportMenu

```
public class ExecSupportMenu
```

```
extends javax.swing.JMenu
```

```
implements java.awt.event.ActionListener
```

This class holds the 'Exec Support' menu items.

**See Also:**[Serialized Form](#)**Inner classes inherited from class javax.swing.JMenu**

```
javax.swing.JMenu.AccessibleJMenu, javax.swing.JMenu.MenuChangeListener,  
javax.swing.JMenu.WinListener
```

**Inner classes inherited from class javax.swing.JMenuItem**

```
javax.swing.JMenuItem.AccessibleJMenuItem
```

**Inner classes inherited from class javax.swing.AbstractButton**

```
javax.swing.AbstractButton.AccessibleAbstractButton,  
javax.swing.AbstractButton.ButtonChangeListener
```

### Inner classes inherited from class javax.swing.JComponent

javax.swing.JComponent.AccessibleJComponent, javax.swing.JComponent.IntVector, javax.swing.JComponent.KeyboardBinding, javax.swing.JComponent.KeyboardState

### Inner classes inherited from class java.awt.Component

java.awt.Component.AWTTreeLock

## Field Summary

private javax.swing.JMenuItem	<u>compileMenuItem</u> Initiates the 'Compile' event
private javax.swing.JMenuItem	<u>executeMenuItem</u> Initiates the 'Execute' event
protected caps.CAPSMain.CAPSMainWindow	<u>ownerWindow</u> The main window which owns this menu.
private javax.swing.JMenuItem	<u>scheduleMenuItem</u> Initiates the 'Schedule' event
private javax.swing.JMenuItem	<u>translateMenuItem</u> Initiates the 'Translate' event

### Fields inherited from class javax.swing.JMenu

delay, listenerRegistry, menuChangeListener, menuEvent, popupListener, popupMenu, uiClassID

### Fields inherited from class javax.swing.JMenuItem

accelerator, uiClassID

### Fields inherited from class javax.swing.AbstractButton

actionListener, BORDER\_PAINTED\_CHANGED\_PROPERTY, changeEvent, changeListener, CONTENT\_AREA\_FILLED\_CHANGED\_PROPERTY, contentAreaFilled, defaultIcon, defaultMargin, DISABLED\_ICON\_CHANGED\_PROPERTY, DISABLED\_SELECTED\_ICON\_CHANGED\_PROPERTY, disabledIcon, disabledSelectedIcon, FOCUS\_PAINTED\_CHANGED\_PROPERTY, HORIZONTAL\_ALIGNMENT\_CHANGED\_PROPERTY, HORIZONTAL\_TEXT\_POSITION\_CHANGED\_PROPERTY, horizontalAlignment, horizontalTextPosition, ICON\_CHANGED\_PROPERTY, itemListener, margin, MARGIN\_CHANGED\_PROPERTY, MNEMONIC\_CHANGED\_PROPERTY, model, MODEL\_CHANGED\_PROPERTY, paintBorder, paintFocus, PRESSED\_ICON\_CHANGED\_PROPERTY, pressedIcon, ROLLOVER\_ENABLED\_CHANGED\_PROPERTY, ROLLOVER\_ICON\_CHANGED\_PROPERTY, ROLLOVER\_SELECTED\_ICON\_CHANGED\_PROPERTY, rolloverEnabled, rolloverIcon, rolloverSelectedIcon, SELECTED\_ICON\_CHANGED\_PROPERTY, selectedIcon, text, TEXT\_CHANGED\_PROPERTY, VERTICAL\_ALIGNMENT\_CHANGED\_PROPERTY, VERTICAL\_TEXT\_POSITION\_CHANGED\_PROPERTY, verticalAlignment, verticalTextPosition

### Fields inherited from class javax.swing.JComponent

`_bounds`, `accessibleContext`, `alignmentX`, `alignmentY`, `ANCESTOR_USING_BUFFER`, `ancestorNotifier`, `autoscroller`, `border`, `changeSupport`, `clientProperties`, `flags`, `HAS_FOCUS`, `IS_DOUBLE_BUFFERED`, `IS_OPAQUE`, `IS_PAINTING_TILE`, `KEYBOARD_BINDINGS_KEY`, `listenerList`, `maximumSize`, `minimumSize`, `NEXT_FOCUS`, `paintImmediatelyClip`, `paintingChild`, `preferredSize`, `readObjectCallbacks`, `REQUEST_FOCUS_DISABLED`, `tmpRect`, `TOOL_TIP_TEXT_KEY`, `ui`, `uiClassID`, `UNDEFINED_CONDITION`, `vetoableChangeSupport`, `WHEN_ANCESTOR_OF_FOCUSED_COMPONENT`, `WHEN_FOCUSED`, `WHEN_IN_FOCUSED_WINDOW`

### Fields inherited from class java.awt.Container

`component`, `containerListener`, `containerSerializedDataVersion`, `dispatcher`, `layoutMgr`, `maxSize`, `ncomponents`, `serialVersionUID`

### Fields inherited from class java.awt.Component

`actionListenerK`, `adjustmentListenerK`, `appContext`, `assert`, `background`, `BOTTOM_ALIGNMENT`, `CENTER_ALIGNMENT`, `changeSupport`, `componentListener`, `componentListenerK`, `componentOrientation`, `componentSerializedDataVersion`, `containerListenerK`, `cursor`, `dropTarget`, `enabled`, `eventMask`, `focusListener`, `focusListenerK`, `font`, `foreground`, `hasFocus`, `height`, `incRate`, `inputMethodListener`, `inputMethodListenerK`, `isInc`, `isPacked`, `itemListenerK`, `keyListener`, `keyListenerK`, `LEFT_ALIGNMENT`, `locale`, `LOCK`, `minSize`, `mouseListener`, `mouseListenerK`, `mouseMotionListener`, `mouseMotionListenerK`, `name`, `nameExplicitlySet`, `newEventsOnly`, `ownedWindowK`, `parent`, `peer`, `peerFont`, `popups`, `prefSize`, `RIGHT_ALIGNMENT`, `serialVersionUID`, `textListenerK`, `TOP_ALIGNMENT`, `valid`, `visible`, `width`, `windowListenerK`, `x`, `y`

## Constructor Summary

**ExecSupportMenu**(caps.CAPSMMain.CAPSMMainWindow owner)

Constructor for this class.

## Method Summary

`void` **actionPerformed**(java.awt.event.ActionEvent e)  
Action event handler for the menu events.

### Methods inherited from class javax.swing.JMenu

`,` `add`, `add`, `add`, `add`, `addMenuListener`, `addSeparator`, `buildMenuElementArray`, `clearListenerRegistry`, `createActionChangeListener`, `createMenuChangeListener`, `createWinListener`, `doClick`, `ensurePopupMenuCreated`, `fireMenuCanceled`, `fireMenuDeselected`, `fireMenuSelected`, `getAccessibleContext`, `getComponent`, `getDelay`, `getItem`, `getItemCount`, `getMenuComponent`, `getMenuComponentCount`, `getMenuComponents`, `getPopupMenu`, `getPopupMenuOrigin`, `getSubElements`, `getUIClassID`, `insert`, `insert`, `insert`, `insertSeparator`, `isMenuComponent`, `isPopupMenuVisible`, `isSelected`, `isTearOff`, `isTopLevelMenu`, `menuSelectionChanged`, `paramString`, `processKeyEvent`, `registerMenuItemForAction`, `remove`, `remove`, `remove`, `removeAll`, `removeMenuListener`, `setAccelerator`, `setDelay`, `setMenuLocation`, `setModel`, `setPopupMenuVisible`, `setSelected`, `translateToPopupMenu`, `translateToPopupMenu`, `unregisterMenuItemForAction`, `updateUI`, `writeObject`





## Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, addImpl, applyOrientation, countComponents, deliverEvent, dispatchEventImpl, dispatchEventToSelf, doLayout, eventEnabled, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents\_NoClientCode, getComponents, getCursorTarget, getLayout, getMouseEventTarget, getWindow, initIDs, insets, invalidate, invalidateTree, isAncestorOf, layout, lightweightPrint, list, list, locate, minimumSize, nextFocus, paintComponents, postProcessKeyEvent, postsOldMouseEvent, preferredSize, preProcessKeyEvent, print, printComponents, printOneComponent, processContainerEvent, processEvent, proxyEnableEvents, proxyRequestFocus, removeContainerListener, setCursor, setFocusOwner, setLayout, transferFocus, updateCursor, validate, validateTree

## Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, areInputMethodsEnabled, bounds, checkImage, checkImage, coalesceEvents, constructComponentName, contains, createImage, createImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, getBackground, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont\_NoClientCode, getFont, getFontMetrics, getForeground, getInputContext, getInputMethodRequests, getIntrinsicCursor, getLocale, getLocation, getLocationOnScreen, getName, getNativeContainer, getParent\_NoClientCode, getParent, getPeer, getSize, getToolkit, getToolkitImpl, getTreeLock, getWindowForObject, gotFocus, handleEvent, hide, imageUpdate, inside, isDisplayable, isEnabled, isEnabledImpl, isLightweight, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, printAll, processComponentEvent, processInputMethodEvent, processMouseEvent, remove, removeComponentListener, removeFocusListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setDropTarget, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus

## Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, registerNatives, wait, wait, wait

## Field Detail

### translateMenuItem

```
private javax.swing.JMenuItem translateMenuItem
```

Initiates the 'Translate' event

### scheduleMenuItem

```
private javax.swing.JMenuItem scheduleMenuItem
```

Initiates the 'Schedule' event

---

### **compileMenuItem**

```
private javax.swing.JMenuItem compileMenuItem
```

Initiates the 'Compile' event

---

### **executeMenuItem**

```
private javax.swing.JMenuItem executeMenuItem
```

Initiates the 'Execute' event

---

### **ownerWindow**

```
protected caps.CAPSMain.CAPSMainWindow ownerWindow
```

The main window which owns this menu.

---

## **Constructor Detail**

### **ExecSupportMenu**

```
public ExecSupportMenu(caps.CAPSMain.CAPSMainWindow owner)
```

Constructor for this class.

**Parameters:**

`owner` - CAPSMainWindow that owns the menu

## **Method Detail**

### **actionPerformed**

```
public void actionPerformed(java.awt.event.ActionEvent e)
```

Action event handler for the menu events.

**Specified by:**

`actionPerformed` in interface `java.awt.event.ActionListener`

**Parameters:**

`e` - The action event that is created by selecting a menu item

---

caps.CAPSMain

## Class PrototypeMenu

```
java.lang.Object
|
+--java.awt.Component
   |
   +--java.awt.Container
      |
      +--javax.swing.JComponent
         |
         +--javax.swing.AbstractButton
            |
            +--javax.swing.JMenuItem
               |
               +--javax.swing.JMenu
                  |
                  +--caps.CAPSMain.PrototypeMenu
```

```
public class PrototypeMenu
extends javax.swing.JMenu
implements java.awt.event.ActionListener
```

This class holds the 'Prototype' menu items.

**See Also:**

[Serialized Form](#)

**Inner classes inherited from class javax.swing.JMenu**

```
javax.swing.JMenu.AccessibleJMenu, javax.swing.JMenu.MenuChangeListener,
javax.swing.JMenu.WinListener
```

**Inner classes inherited from class javax.swing.JMenuItem**

```
javax.swing.JMenuItem.AccessibleJMenuItem
```

**Inner classes inherited from class javax.swing.AbstractButton**

```
javax.swing.AbstractButton.AccessibleAbstractButton,
javax.swing.AbstractButton.ButtonChangeListener
```

### Inner classes inherited from class javax.swing.JComponent

javax.swing.JComponent.AccessibleJComponent, javax.swing.JComponent.IntVector, javax.swing.JComponent.KeyboardBinding, javax.swing.JComponent.KeyboardState

### Inner classes inherited from class java.awt.Component

java.awt.Component.AWTTreeLock

## Field Summary

private javax.swing.JMenuItem	<u>commitWorkMenuItem</u> Initiates the 'Commit Work' event
private javax.swing.JMenuItem	<u>newMenuItem</u> Initiates the 'New' event
private javax.swing.JMenuItem	<u>openMenuItem</u> Initiates the 'Open' event
protected caps.CAPSMain.CAPSMainWindow	<u>ownerWindow</u> The main window which owns this menu.
private javax.swing.JMenuItem	<u>quitMenuItem</u> Initiates the 'Quit' event
private javax.swing.JMenuItem	<u>retrieveMenuItem</u> Initiates the 'Retrieve From DDB' event

### Fields inherited from class javax.swing.JMenu

delay, listenerRegistry, menuChangeListener, menuEvent, popupListener, popupMenu, uiClassID

### Fields inherited from class javax.swing.JMenuItem

accelerator, uiClassID

### Fields inherited from class javax.swing.AbstractButton

actionListener, BORDER\_PAINTED\_CHANGED\_PROPERTY, changeEvent, changeListener, CONTENT\_AREA\_FILLED\_CHANGED\_PROPERTY, contentAreaFilled, defaultIcon, defaultMargin, DISABLED\_ICON\_CHANGED\_PROPERTY, DISABLED\_SELECTED\_ICON\_CHANGED\_PROPERTY, disabledIcon, disabledSelectedIcon, FOCUS\_PAINTED\_CHANGED\_PROPERTY, HORIZONTAL\_ALIGNMENT\_CHANGED\_PROPERTY, HORIZONTAL\_TEXT\_POSITION\_CHANGED\_PROPERTY, horizontalAlignment, horizontalTextPosition, ICON\_CHANGED\_PROPERTY, itemListener, margin, MARGIN\_CHANGED\_PROPERTY, MNEMONIC\_CHANGED\_PROPERTY, model, MODEL\_CHANGED\_PROPERTY, paintBorder, paintFocus, PRESSED\_ICON\_CHANGED\_PROPERTY, pressedIcon, ROLLOVER\_ENABLED\_CHANGED\_PROPERTY, ROLLOVER\_ICON\_CHANGED\_PROPERTY, ROLLOVER\_SELECTED\_ICON\_CHANGED\_PROPERTY, rolloverEnabled, rolloverIcon, rolloverSelectedIcon, SELECTED\_ICON\_CHANGED\_PROPERTY, selectedIcon, text, TEXT\_CHANGED\_PROPERTY, VERTICAL\_ALIGNMENT\_CHANGED\_PROPERTY, VERTICAL\_TEXT\_POSITION\_CHANGED\_PROPERTY, verticalAlignment, verticalTextPosition

### Fields inherited from class javax.swing.JComponent

\_bounds, accessibleContext, alignmentX, alignmentY, ANCESTOR\_USING\_BUFFER, ancestorNotifier, autoscroller, border, changeSupport, clientProperties, flags, HAS\_FOCUS, IS\_DOUBLE\_BUFFERED, IS\_OPAQUE, IS\_PAINTING\_TILE, KEYBOARD\_BINDINGS\_KEY, listenerList, maximumSize, minimumSize, NEXT\_FOCUS, paintImmediatelyClip, paintingChild, preferredSize, readObjectCallbacks, REQUEST\_FOCUS\_DISABLED, tmpRect, TOOL\_TIP\_TEXT\_KEY, ui, uiClassID, UNDEFINED\_CONDITION, vetoableChangeSupport, WHEN\_ANCESTOR\_OF\_FOCUSED\_COMPONENT, WHEN\_FOCUSED, WHEN\_IN\_FOCUSED\_WINDOW

### Fields inherited from class java.awt.Container

component, containerListener, containerSerializedDataVersion, dispatcher, layoutMgr, maxSize, ncomponents, serialVersionUID

### Fields inherited from class java.awt.Component

actionListenerK, adjustmentListenerK, appContext, assert, background, BOTTOM\_ALIGNMENT, CENTER\_ALIGNMENT, changeSupport, componentListener, componentListenerK, componentOrientation, componentSerializedDataVersion, containerListenerK, cursor, dropTarget, enabled, eventMask, focusListener, focusListenerK, font, foreground, hasFocus, height, incRate, inputMethodListener, inputMethodListenerK, isInc, isPacked, itemListenerK, keyListener, keyListenerK, LEFT\_ALIGNMENT, locale, LOCK, minSize, mouseListener, mouseListenerK, mouseMotionListener, mouseMotionListenerK, name, nameExplicitlySet, newEventsOnly, ownedWindowK, parent, peer, peerFont, popups, prefSize, RIGHT\_ALIGNMENT, serialVersionUID, textListenerK, TOP\_ALIGNMENT, valid, visible, width, windowListenerK, x, y

## Constructor Summary

PrototypeMenu(caps.CAPSMMain.CAPSMMainWindow owner)

Constructor for this class.

## Method Summary

void	<u>actionPerformed</u> (java.awt.event.ActionEvent e) Action event handler for the menu events.
void	<u>processCommitWorkMenuItem</u> () Handles the event which is caused by selecting the 'Commit' menu item.
void	<u>processNewMenuItem</u> () Handles the event which is caused by selecting the 'New' menu item.
void	<u>processOpenMenuItem</u> () Handles the event which is caused by selecting the 'Open' menu item.

### Methods inherited from class javax.swing.JMenu

, add, add, add, add, add, addMenuListener, addSeparator, buildMenuElementArray, clearListenerRegistry, createActionChangeListener, createMenuChangeListener, createWinListener, doClick, ensurePopupMenuCreated, fireMenuCanceled, fireMenuDeselected, fireMenuSelected, getAccessibleContext, getComponent, getDelay, getItem, getItemCount, getMenuComponent, getMenuComponentCount, getMenuComponents, getPopupMenu, getPopupMenuOrigin, getSubElements, getUIClassID, insert, insert, insert, insertSeparator, isMenuComponent, isPopupMenuVisible, isSelected, isTearOff, isTopLevelMenu, menuSelectionChanged, paramString, processKeyEvent, registerMenuItemForAction, remove, remove, remove, removeAll, removeMenuListener, setAccelerator, setDelay, setMenuLocation, setModel, setPopupMenuVisible, setSelected, translateToPopupMenu, translateToPopupMenu, unregisterMenuItemForAction, updateUI, writeObject

### Methods inherited from class javax.swing.JMenuItem

addMenuDragMouseListener, addMenuKeyListener, alwaysOnTop, fireMenuDragMouseDragged, fireMenuDragMouseEntered, fireMenuDragMouseExited, fireMenuDragMouseReleased, fireMenuKeyPressed, fireMenuKeyReleased, fireMenuKeyTyped, getAccelerator, init, isArmed, processKeyEvent, processMenuDragMouseEvent, processMenuKeyEvent, processMouseEvent, readObject, removeMenuDragMouseListener, removeMenuKeyListener, setArmed, setEnabled, setUI

### Methods inherited from class javax.swing.AbstractButton

addActionListener, addChangeListener, addItemListener, checkHorizontalKey, checkVerticalKey, createActionListener, createChangeListener, createItemListener, doClick, fireActionPerformed, fireItemStateChanged, fireStateChanged, getActionCommand, getDisabledIcon, getDisabledSelectedIcon, getHorizontalAlignment, getHorizontalTextPosition, getIcon, getLabel, getMargin, getMnemonic, getModel, getPressedIcon, getRolloverIcon, getRolloverSelectedIcon, getSelectedIcon, getSelectedObjects, getText, getUI, getVerticalAlignment, getVerticalTextPosition, isBorderPainted, isContentAreaFilled, isFocusPainted, isRolloverEnabled, paintBorder, removeActionListener, removeChangeListener, removeItemListener, setActionCommand, setBorderPainted, setContentAreaFilled, setDisabledIcon, setDisabledSelectedIcon, setFocusPainted, setHorizontalAlignment, setHorizontalTextPosition, setIcon, setLabel, setMargin, setMnemonic, setMnemonic, setPressedIcon, setRolloverEnabled, setRolloverIcon, setRolloverSelectedIcon, setSelectedIcon, setText, setUI, setVerticalAlignment, setVerticalTextPosition



## Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, areInputMethodsEnabled, bounds, checkImage, checkImage, coalesceEvents, constructComponentName, contains, createImage, createImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, getBackground, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont\_NoClientCode, getFont, getFontMetrics, getForeground, getInputContext, getInputMethodRequests, getIntrinsicCursor, getLocale, getLocation, getLocationOnScreen, getName, getNativeContainer, getParent\_NoClientCode, getParent, getPeer, getSize, getToolkit, getToolkitImpl, getTreeLock, getWindowForObject, gotFocus, handleEvent, hide, imageUpdate, inside, isDisplayable, isEnabled, isEnabledImpl, isLightweight, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, printAll, processComponentEvent, processInputMethodEvent, processMouseEvent, remove, removeComponentListener, removeFocusListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setDropTarget, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus

## Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, registerNatives, wait, wait, wait

## Field Detail

### **newMenuItem**

private javax.swing.JMenuItem **newMenuItem**

Initiates the 'New' event

---

### **openMenuItem**

private javax.swing.JMenuItem **openMenuItem**

Initiates the 'Open' event

---

### **commitWorkMenuItem**

private javax.swing.JMenuItem **commitWorkMenuItem**

Initiates the 'Commit Work' event

---



## retrieveMenuItem

```
private javax.swing.JMenuItem retrieveMenuItem
```

Initiates the 'Retrieve From DDB' event

---

## quitMenuItem

```
private javax.swing.JMenuItem quitMenuItem
```

Initiates the 'Quit' event

---

## ownerWindow

```
protected caps.CAPSMMain.CAPSMMainWindow ownerWindow
```

The main window which owns this menu.

## Constructor Detail

### PrototypeMenu

```
public PrototypeMenu(caps.CAPSMMain.CAPSMMainWindow owner)
```

Constructor for this class.

**Parameters:**

`owner` - The main window which has created this menu.

## Method Detail

### actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent e)
```

Action event handler for the menu events.

**Specified by:**

`actionPerformed` in interface `java.awt.event.ActionListener`

**Parameters:**

`e` - The action event that is created by selecting a menu item from this menu

---

### processNewMenuItem

```
public void processNewMenuItem()
```

Handles the event which is caused by selecting the 'New' menu item.

---

### **processOpenMenuItem**

```
public void processOpenMenuItem()
```

Handles the event which is caused by selecting the 'Open' menu item.

---

### **processCommitWorkMenuItem**

```
public void processCommitWorkMenuItem()
```

Handles the event which is caused by selecting the 'Commit' menu item.

---

### **Class** [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

[FRAMES](#) [NO FRAMES](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

---

## APPENDIX E: ACRONYMS

CAPS	Computer Aided Prototyping System
CASE	Computer Aided Software Engineering
CDR	Common Data Representation
CGI	Common Gateway Interface
CORBA	Common Object Request Broker Architecture
CPL	Common Prototyping Language
DCOM	Distributed Component Object Model
DII	Dynamic Invocation Interface
DII COE	Defense Information Infrastructure Common Operating Environment
DOD	Department of Defense
DSI	Dynamic Skeleton Interface
GAMBITS	Graphical Approach to Modeling and Building Interactively a Technical System
GUI	graphical user interface
HIS	Heterogeneous Systems Integrator
HTTP	Hypertext Transfer Protocol
IDE	integrated development environments
IDL	Interface Definition Language
JDK	Java Developer Kit
JDS	Jackson Development System
JVM	Java Virtual Machine
LAN	local area network
OMA	Object Management Architecture
OMG	Object Management Group
ORB	Object Request Broker
PC	personal computer
PROTOTECH	Prototyping Technology
PSDL	Prototype System Design Language
RaPIER	Rapid Prototyping to Investigate End-user Requirements
RMI	Remote Method Invocations
RPC	remote procedure call
SSDL	System Specification and Design Language
UML	Unified Modeling Language
WAN	wide area network
WWW	World Wide Web

THIS PAGE INTENTIONALLY LEFT BLANK

## LIST OF REFERENCES

- ACOS94 International Software Systems Incorporated, Report 5581, *Proto Code Generation Techniques*, final report, by Ramon Acosta, pp. 1-27, May 1994
- BARB91 Carnegie-Mellon University Software Engineering Institute, Report CMU/SEI-91-TR-21, *Durra: An Integrated Approach to Software Specification, Modeling, and Rapid Prototyping*, by Mario Barbacci, Dennis Doubleday, Charles Weinstock and Randal Lichota, pp. 1-15, September 1991
- BOEH76 B.W. Boehm, "Software Engineering," *IEEE Transactions on Computers*, v.C-25, no. 12, pp. 1226-1241, 1976
- BOEH86 B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *ACM SIGSOFT Software Engineering Notes*, v. 11, no. 8, pp. 14-24, 1986
- BOOC99 Booch, G., Rumbough, J., and Jacobson, I., *The Unified Modeling Language Guide*, 1<sup>st</sup> ed., Addison Wesley Longman, Inc., 1999
- CHAV95 ILLGEN Simulation Technologies Incorporated, Report IST95-R-085, *The Commercialization of a Rapid Prototyping Tool for Simulating Command and Control Applications*, by Steven Chavin, Valdis Berzins, and Larry Dworkin, p.4, 29 June 1995
- DAMP92 Naval Postgraduate School Computer Science Department, Report NPS-CS-92-014, *A Model for Merging Software Prototypes*, by David Dampier, pp.1-4, 23 September 1992
- DSB87 U.S. Department of Defense, Office of the Undersecretary of Defense for Acquisition, *Report of the Defense Science Board Task Force on Military Software*, p. 34, Government Printing Office, Washington, D.C., 1987
- DURA99 Duranlioglu, Ilker, *Implementation of a Portable PSDL Editor for the Heterogeneous Systems Integrator*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1999
- KIMB92 Honeywell Systems and Research Center, Report CS-C92-002, *Honeywell ProtoTech. Phase 1*, by John Kimball, Tim King, Aaron Larson, Chris Miller and Jon Ward, pp.3-9, June 1992
- LUQI88a Luqi and M. Ketabchi, "A Computer-Aided Prototyping System", *IEEE Software*, pp. 66-72, March 1988
- LUQI88b Luqi, V. Berzins, and R. Yeh, "A Prototyping Language for Real-Time Software", *IEEE Transactions on Software Engineering*, v. 14, no. 10, pp. 1409-1423, October 1988
- LUQI96 Luqi, "System Engineering and Computer Aided Prototyping", *Journal of Systems Integration, special issue on Computer Aided Prototyping*, v. 6, no. 1, pp.15-17, 1996

- MILL90 North Carolina University at Chapel Hill Computer Science Department, Report TR90-041, *Prototyping Parallel and Distributed Programs in Proteus*, by Peter Mills, Lars Nyland, Jan Prins, John Reif, and Robert Wagner, pp.1-5, October 1990
- ORFA98 Orfali,R. and Harkey, D., *Client/Server Programming with Java and CORBA*, 2<sup>nd</sup> ed., John Wiley& Sons, Inc., 1998
- POPE98 Pope, Alan, *The CORBA Reference Guide*, 1<sup>st</sup> ed., Addison Wesley Longman, Inc., 1998
- POWE96 Powell, D., "GAMBITS- From Requirements Capture to C++ Code Generation," *IEE Colloquium on Jackson System Development (JSD) – The Original Object Oriented Method*, no. 1996/20, pp.1-3, 1996
- PROT96 Jelica Protic, Milo Tomasevic, and Veljko Milutinovic, "Distributed Shared Memory: Concepts and Systems," *IEEE Parallel and Distributed Technology*, v. 4, no. 2, pp. 63-77, 1996
- ROLL91 Rollo, A.L., "Jackson System Development", *IEE Colloquium on Introduction to Software Design Methodologies*, pp.301-313, 1991
- ROYC76 W.W. Royce, "Managing the Development of Large Software Systems," *Proceedings in WESCON*, 1976
- SCHM99 Schmidt, Dave, "Overview of CORBA," [<http://www.cs.wustl.edu/~schmidt/corba-overview.html>], August 1999
- WOOD92 Carnegie-Mellon University Software Engineering Institute, Report CMU/SEI-92-TR-13, *A Classification and Bibliography of Software Prototyping*, by David Wood and Kyo Kang, pp. 1-14, October 1992
- WELC86 International Software Systems Incorporated, Report TR-T101/85/0640-5, *Rapid Prototyping System*, by Terry Welch, pp. 1-50, 15 December 1986
- ZAVE91 Pamela Zave, "An Insider's Evaluation of PAISLey," *IEEE Transactions on Software Engineering*, v. 17, no. 3, pp. 212-225, March 1991

## INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2  
8725 John J. Kingman Road, Ste 0944  
Ft. Belvoir, Virginia 22060-6218
2. Dudley Knox Library .....2  
Naval Postgraduate School  
411 Dyer Rd.  
Monterey, California 93943-5101
3. Dr Dan Boger, Chairman, Code CS.....1  
Computer Science Department  
Naval Postgraduate School  
Monterey, CA 93940-5000
4. Dr Man-Tak Shing.....1  
Computer Science Department, Code CS/Sh  
Naval Postgraduate School  
Monterey, California 93943-5100
5. Dr Luqi.....1  
Computer Science Department, Code CS/Lq  
Naval Postgraduate School  
Monterey, California 93943-5100
6. CDR Gary Kreeger.....1  
1011 Leahy Road  
Monterey, CA 93940
7. LCDR Chris Eagle.....1  
Computer Science Department, Code CS/Ce  
Naval Postgraduate School  
Monterey, CA 93943-5100