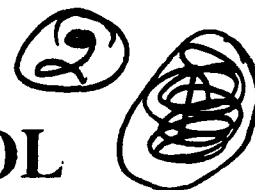


AD-A246 081



NAVAL POSTGRADUATE SCHOOL
Monterey, California



DTIC
ELECTE
FEB 20 1992
S D

THESIS

A REUSABLE COMPONENT RETRIEVAL SYSTEM
FOR
PROTOTYPING

by

John Kelly McDowell

September 1991

Thesis Advisor:

Dr. Luqi

Approved for public release; distribution is unlimited.

92 2 12 157

92-03720

REPORT DOCUMENTATION PAGE

1. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		16. RESTRICTIVE MARKINGS	
2. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited	
4. CLASSIFICATION/DOWNGRADING SCHEDULE		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6. NAME OF PERFORMING ORGANIZATION Computer Science Dept. Naval Postgraduate School		7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000		7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	
8. NAME OF FUNDING/SPONSORING ORGANIZATION National Science Foundation		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER CCR-9058453	
10. ADDRESS (City, State, and ZIP Code) 1100 G Street, N.W. Washington, D.C., 20550		10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. PROJECT NO. TASK NO. WORK UNIT ACCESSION NO.	

11. TITLE (Include Security Classification)
REUSABLE COMPONENT RETRIEVAL SYSTEM FOR PROTOTYPING

12. PERSONAL AUTHOR(S)
Dowell, John Kelly.

13. TYPE OF REPORT Master's Thesis	13b. TIME COVERED FROM 09/90 TO 09/91	14. DATE OF REPORT (Year, Month, Day) September 1991	15. PAGE COUNT 343
---------------------------------------	--	---	-----------------------

16. 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 United States Government.

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Reusable Software Components, Rapid Prototyping
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Prototyping is an important software development method to rapidly construct software, validate and refine requirements, and check consistency of proposed software designs. This thesis describes the design and implementation of a CASE tool to be used in conjunction with the Computer Aided Prototyping System (CAPS) which retrieves and prepares reusable components for use in PSDL (Prototype System Description Language) prototypes. Reusable components and their PSDL specifications are stored in a software base.

Components can be retrieved from the software base via its Object-Oriented Data Base Management System (OODBMS) using PSDL formulate queries. All of the PSDL specifications for the reusable components are normalized and stored in the software base to support efficient search based on a given query PSDL specification for a software component. The search process is based on both syntactic and semantic matches between the query and stored components.

Our software base has been designed to be easily configured to support storage and retrieval of reusable components in any programming language with the initial configuration for Ada components.

A window based user interface was also implemented to allow easy access to the software base via the CAPS user interface as well as alone use.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL		22b. TELEPHONE (Include Area Code) (408) 646-2735	22c. OFFICE SYMBOL CS/lq

Approved for public release; distribution is unlimited

A Reusable Component Retrieval System for Prototyping

by

John Kelly McDowell
Lieutenant, United States Navy

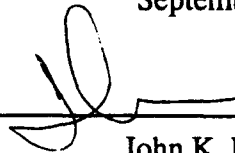
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

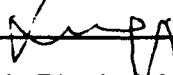
NAVAL POSTGRADUATE SCHOOL
September, 1991

Author:



John K. McDowell

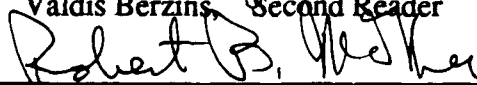
Approved By:



Luqi, Thesis Advisor



Valdis Berzins, Second Reader



Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

Prototyping is an important software development method to rapidly construct software, validate and refine requirements, and check the consistency of proposed software designs. This thesis describes the design and implementation of a CASE tool to be used in conjunction with the Computer Aided Prototyping System (CAPS) which retrieves and prepares reusable components for use in PSDL (Prototype System Description Language) prototypes. Reusable components and their PSDL specifications are stored in a software base.

Components can be retrieved from the software base via its Object-Oriented Data Base Management System (OODBMS) using PSDL to formulate queries. All of the PSDL specifications for the reusable components are normalized and stored in the software base to support efficient search based on a given query PSDL specification for a software component. The search process is based on both syntactic and semantic matches between the query and stored components.

Our software base has been designed to be easily configured to support storage and retrieval of reusable components in any programming language with the initial configuration for Ada components.

A window based user interface was also implemented to allow easy access to the software base via the CAPS user interface as well as stand alone use.

TABLE OF CONTENTS

I. INTRODUCTION	1
A. THE SOFTWARE CRISIS	2
B. STRUCTURED ANALYSIS	3
C. RAPID PROTOTYPING	6
D. THE COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)	8
E. GOALS OF THIS THESIS	9
II. REUSABLE COMPONENT LIBRARIES.	10
A. RETRIEVAL METHODS	11
1. Browsers.	11
2. Informal Specifications	12
B. KEYWORD SEARCH	12
C. MULTI-ATTRIBUTE SEARCH	13
1. Natural Language Interfaces.	14
2. Formal Specification	14
D. REVIEW OF CURRENT SYSTEMS	15
1. Draco	15
2. Rapid	16
3. Operation Support System	17
4. The Reusable Software Library	17
5. Common Ada Missile Packages	18
6. Software Reuse At Hewlett-Packard.	19
III. CAPS AND PSDL.	20
A. USING CAPS TO BUILD EXECUTABLE PROTOTYPES	20
B. USING REUSABLE COMPONENTS IN CAPS.	23

IV. SOFTWARE BASE IMPLEMENTATION	25
A. REQUIREMENTS	25
B. ONTOS DATABASE MANAGEMENT SYSTEM	26
C. SEGREGATION OF REUSABLE COMPONENT DOMAINS	29
D. STORAGE OF UNCONSTRAINED TEXT OBJECTS	30
E. BROWSING THE SOFTWARE BASE	32
1. Named Look Up Of Components	33
2. Keyword Querying	33
F. QUERY BY SPECIFICATION	35
1. Syntactic Matching	38
2. Operator Component Library	41
3. Abstract Data Type Library	42
G. DATA STREAM TYPE MATCHING	44
H. ABSTRACT REPRESENTATION OF SOFTWARE BASE	46
COMPONENTS	
I. INTEGRATING RETRIEVED COMPONENTS INTO CAPS.	48
J. SOFTWARE BASE INTERFACE	51
1. Command Line Interface	51
2. Graphical User Interface	52
V. CONCLUSIONS AND FUTURE RESEARCH	53
A. ADDING COMPONENTS TO THE SOFTWARE BASE	53
1. Component Testing	53
2. Component Implementation Restrictions	53
3. Writing Formal Specifications Of Existing Components	54
B. DELETING AND UPDATING COMPONENTS.	54
C. EFFICIENCY	55
D. SOFTWARE BASE SYSTEM IMPLEMENTATION LANGUAGE	55
LIST OF REFERENCES	57

APPENDIX A - USING PSDL TO SPECIFY REUSABLE	60
COMPONENTS	
APPENDIX B - C++ SOURCE CODE FOR SOFTWARE BASE . . .	69
APPENDIX C - PARSER GENERATION INPUT FILES	.210
APPENDIX D - INTEGRATING ADA COMPONENTS INTO . . .	.228
CAPS	
APPENDIX E - COMMAND LINE INTERFACE	.238
SPECIFICATION	
APPENDIX F - SOFTWARE BASE GRAPHICAL USER	.246
INTERFACE MANUAL	
APPENDIX G - SOFTWARE BASE GRAPHICAL USER	.259
INTERFACE SOURCE CODE	
BIBLIOGRAPHY	.326
INITIAL DISTRIBUTION LIST.	.327

ACKNOWLEDGEMENT

I would like to thank Professor Luqi and Professor Berzins for all of their support throughout this project. Their words of wisdom and encouragement got me through those difficult periods when this project seemed hopeless.

I thank Albert Wong and the rest of the NPS Computer Science Department technical staff for supporting my software needs throughout the past two years.

I would also like to thank Captain Robert Steigerwald, U.S. Air Force, for all of his assistance in the completion of the project and in the preparation of this thesis.

A very special thank you to my wife Naida and two sons, Jonathan and Brandon. Without their sacrifice over the past two years, I could not have completed this research.

Accession For	
NTIS CRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution /	
Availability Codes	
Dist	Availability / or Special
A-1	

I. INTRODUCTION

This thesis addresses the issues related to the design and implementation of an automated reusable software component retrieval system. The purpose of the system is to support the Computer Aided Prototyping System (CAPS) [Ref 1]. CAPS is an ongoing software engineering research project at the Naval Postgraduate School. The reusable component retrieval system is a critical component in the CAPS tool set.

This chapter provides an introduction to computer aided prototyping and the need for automated retrieval of software components. Chapter II details the current state of the art in component reuse and reusable component libraries. Chapter IV is an overview of CAPS and its specification language, Prototyping System Description Language (PSDL), used for specifying reusable components. Chapter IV presents the design and implementation of the software base for CAPS. Chapter V contains the conclusions of this research and recommendations for future research. Appendix A details the usage of PSDL to specify reusable components. Appendix B contains the source code for the software base system. Appendix C is the source code for the generation of the PSDL parser. Appendix D presents an example of how to integrate reusable Ada components into a CAPS prototype. Appendix E provides the specification for the software base command line interface. Appendix F is a users manual for the software base graphical user interface. Appendix G is the source code for the software base graphical user interface.

A. THE SOFTWARE CRISIS

Creating software for hard real-time and embedded computer systems is a complex process. The complexity of this task has created a situation where the demand for these systems currently exceeds the ability of the software industry to develop them.

The United States Department of Defense (DOD) is the world's largest user of embedded computer systems. In the mid 1970's the gap between the growing demand for high quality software and industry's inability to meet that demand caused the DOD to investigate potential solutions to the problem. The DOD concluded that the problems "...appear in the form of software that is non-responsive to user needs, unreliable, excessively expensive, untimely, inflexible, difficult to maintain, and not reusable." [Ref. 2:p41]

One of the results of this investigation was the development of the Ada programming language. Although Ada provides some capabilities to overcome the shortcomings noted by the DOD additional software tools are still required if the software gap is to be closed. These tools are especially needed in the areas of requirements analysis and refinement, software validation, and software testing. The design and implementation of tools in these areas continues to be a major focus of software engineering research.

B. STRUCTURED ANALYSIS

A widely used design methodology that attempted to address the issues in software development is Structured Analysis [Ref 3:p78]. Structured analysis breaks the creation of software systems into distinct areas or steps, which is essentially a variation of the "waterfall" model in DOD-STD-2167A [Ref 4:p10].

Its first step is requirements analysis. During this step the actual needs and external interfaces of the system are identified and recorded. The second step is functional specification of the system. Functional specification uses the requirements from the first step to specify the proposed external functionality of the software system. The third step in structured analysis is system design. During the design step the internal aspects of the system are specified. This design is then used in the fourth step, which is system implementation. After the system has been implemented the system is tested and delivered. After initial system delivery the software enters its maintenance phase. Maintenance of the system follows the same basic approach with each change going through requirements analysis, design, and finally implementation.

If at any time during this approach an inconsistency is identified the process reverts back to the appropriate step to correct the problem. This method of software development has been called the "waterfall approach" because the system goes from one step to the other as though going down a waterfall. Figure 1 is a graphic representation of classical structured analysis.

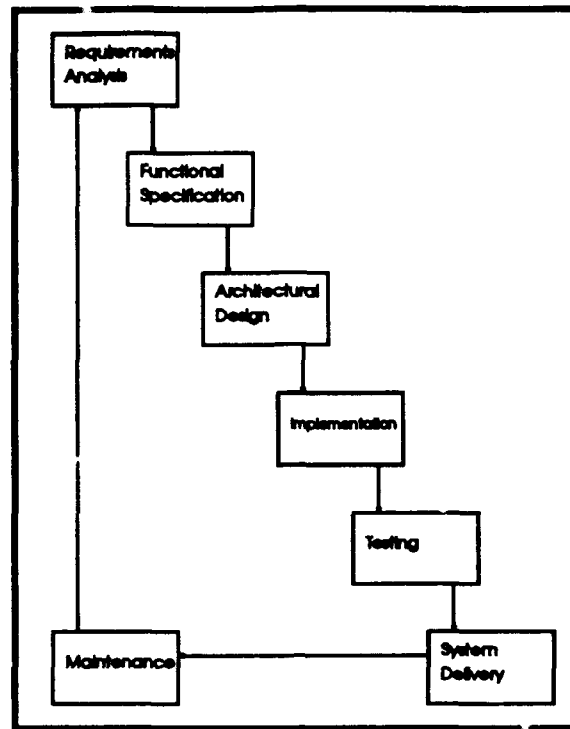


Figure 1 - Classical Structured Analysis

This approach has been modified [Ref 3] to make the steps in the model less distinct and allow more parallel effort. The steps were made less distinct because in practice it was found that information discovered in later steps sometimes required earlier steps to be modified. The ability to complete steps in parallel is possible because some functions lower in the cycle can be performed while higher level items are being completed. This parallel activity can greatly enhance the efficiency of the overall effort and allows for more feedback between steps.

In order for this approach to work there must be a means of communicating the results of each step other than plain English. This is because English prose can be ambiguous and it is very difficult to verify the consistency of a written document.

Several representations were developed in order to convey this information. These representations include Data Flow Diagrams (DFD's), Context Diagrams (CD's), Entity Relationship Diagrams (E/R Diagrams), State Transition Diagrams (STD's), and Data Dictionaries (DD). These representations make it possible for the developers to communicate with each other the behavior of a proposed system.

One of the deficiencies of structured analysis is that the English prose used to describe portions of the requirements of the system often is not precise enough to define critical subsystems. This is very important where failure of a real-time or embedded system to meet a given requirement could result in injury or death. Most military software systems and medical systems fall into this category because failure can result in life threatening circumstances.

Several high level specification languages have been developed to solve these problems. These languages support formal specifications of critical portions of the system. From these formal specifications the system can be verified to achieve the required functionality. One such specification language is SPEC [Ref 5: p82]. SPEC can be used to rigorously specify critical sections to avoid ambiguity.

Communication between a software development team and the system's end users is difficult using structured analysis. This communication is vital since in most cases the software development team is unfamiliar with the domain of the proposed system while the domain experts (end users) are unfamiliar with many of the representations used in the development of the software system.

Although DFD's, CD's, E/R diagrams, STD's, DD's, and formal specification languages are very useful for communication between software engineers, many end users are unfamiliar and/or uncomfortable with them. This results in potential mistakes or misconceptions between the developers and the users in the early stages of a project that can go undetected until the first executable version of the system has been completed. The development of this first executable version of a system can consume so much of a projects allotted development time or budget that it may be too late or costly to make any major modifications. This results in the end user being forced to accept a system that does not meet their original expectations.

C. RAPID PROTOTYPING

One promising area of software engineering research that addresses this problem is rapid prototyping [Ref 6]. A prototype is an executable model of a proposed software system, usually including a software simulation of the system's hardware external interfaces. The prototype accurately reflects chosen aspects of the system including display formats, correctness of computations, and real-time constraints.

Prototyping attempts to solve the communications problem by rapidly developing a prototype of the proposed system from available information and then using that prototype to communicate to the user[Ref 7]. The executable prototype presents a view of the system that the user is most familiar with. This allows the user to provide feedback to the design team that can be used to update the prototype. This feedback

process continues until the user is satisfied that the prototype accurately describes the needs of the system. At this point the prototype system itself acts as one of specification tools for the final system.

There are two problem areas in the acceptance of prototyping as the preferred software development technique. The first is that the process of creating and modifying prototypes must be rapid enough to avoid the same resource consumption pitfalls of classical structured analysis. The second difficulty is that typically prototype systems are used only as a guideline for the final system. Yourdon states "when the modeling is finished, the programs will be thrown away and replaced with REAL programs."

[Ref 3:p98] If all of the prototype system is completely discarded it becomes questionable as to how effective the prototyping approach is at reducing software system development cost and time.

Without addressing these two issues prototyping could actually increase the overall development time and cost of the system as compared to structured analysis. Clearly to fulfill the promise of rapid prototyping it is necessary to overcome these difficulties.

The solution to the first problem is the development of computer aided prototyping systems that enable the rapid development of executable prototypes. To achieve this rapid evolution of executable prototypes it is necessary to achieve a very high rate of code reuse instead of creating the entire prototype from scratch [Ref 8]. The solution to the second problem is to implement the prototype with code of sufficient quality that

only those modules that require performance enhancements need be re-implemented to produce the final system.

With this in mind it is clear that for rapid prototyping to be of maximum benefit, reusable component libraries containing many high quality components coupled with powerful query techniques to identify components for reuse are mandatory. The remainder of this thesis discusses CAPS and the development of a reusable component software base that fulfills these requirements.

D. THE COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)

The Computer Aided Prototyping system is designed as a rapid prototyping system for hard real-time systems. CAPS prototypes a system through translation of the high level specification language Prototyping System Description Language (PSDL) into Ada code along with the incorporation of atomic Ada reusable components [Ref 9]. The concept of using both specification translation and atomic component composition makes CAPS a unique prototyping tool. PSDL is unique in that it provides a rich set of real-time constraints to enable the prototyping of hard real-time systems, and automatic translation of these timing constraints into Ada tasking information.

The use of composition of atomic components allows for the use of high quality reusable Ada components to provide the majority of the code to implement the prototype. PSDL is used to specify the interface and functionality of the atomic components to make automated searches of a reusable component library feasible.

The software base subsystem of CAPS has been designed to allow the user to create a PSDL specification of a necessary component and then perform an automated search of the library for preexisting candidate implementations of the specification. Automation of the search of the software base is critical because the software base must be able to grow indefinitely without significantly degrading the users ability to locate components for reuse. As the software base grows larger fewer components will need to be manually coded, thus achieving a system that provides greater power with time.

E. GOALS OF THIS THESIS

The goal of this thesis is to describe the design and implementation of a software base system for the CAPS prototyping environment. The theoretical foundations for component matching, the development of algorithms to take advantage of these concepts, the design of a database structure that enables the efficient implementation of the entire system, and a description of how to obtain the maximum benefit of using this system are discussed.

II. REUSABLE COMPONENT LIBRARIES

As the gap between the demand for software systems and the software industry's ability to meet it became obvious, so did the need to reuse existing software components. Many retrieval systems have been proposed and implemented that address this issue. Several of these systems are discussed in this section.

In order to compare and contrast these retrieval systems it is necessary to develop a metric by which the systems can be evaluated. How well an information system performs is based on the nature of the objects that are returned for a given query. The two most useful measures of performance for a retrieval system are precision and recall [Ref 10]. Precision is defined as the ratio between the number of relevant components retrieved and the total number of components retrieved. Recall is defined as the ratio between the number of relevant components retrieved and the number of relevant components in the database. Precision and recall are both maximal when they equal 1.

There is a tradeoff between precision and recall. It is easy to have a system maximize one but not the other. If the system returns all objects in the database than recall will always be 1 but precision will be very low. On the other hand if a query only yields one relevant component than precision would be 1 but recall would be low.

In order to obtain maximum reuse of existing software components in a given collection of components, queries on that collection should have a recall value of 1. Without a recall of 1, components that could be reused will be missed. The system also

needs a high degree of precision because it is possible to spend more time manually searching through the results of a component query with low precision than it would take to implement the component manually.

A. RETRIEVAL METHODS

Almost all of the tools developed to assist in reusing software components use one (or more) of three different approaches for retrieval of components; browsers, informal specifications, or formal specifications. For this reason a general overview of these retrieval methods is presented followed by a discussion of some existing tools that use these methods.

1. Browsers

A browser is a tool for looking through a collection of software components. The interface for a browser can range from simple text through complex graphical user interfaces. The goal of all such systems is to allow the user to direct a search through the available components.

The advantage of a browsing system is that the user is given complete control over the retrieval process. This can be important for users who are familiar with the content of the software collection and want the ability to quickly traverse the structure of the collection to find components that they know are in the collection.

The first disadvantage of the system is that it has very low precision. The user may have to look at all of the components to find the one that is desired. Because of the

manual nature of the search, as the software collection grows the time the user spends browsing also increases.

The second disadvantage of browsing is that the system relies on the users knowledge of the structure of software collection. Without such knowledge a user will have difficulty in directing a search to retrieve a desired reusable component.

The third disadvantage of this type of a system is that unless the user finds exactly what they are searching for, there is no clear termination point for the search until every component has been reviewed.

2. Informal Specifications

This technique requires the user to describe or list some attributes of the component that they are looking for. This informal description is then used to direct the user to the appropriate components. Examples of some *common attributes* are *keywords* and *natural language interfaces*.

B. KEYWORD SEARCH

A keyword searching mechanism requires the user to specify a list of words relevant to the component being sought. The keywords the user chooses can be drawn from a known system vocabulary (controlled vocabulary) or they can be unconstrained (uncontrolled vocabulary). In the case of uncontrolled vocabulary synonym tables are often used to normalize the keyword selections into a known vocabulary.

The advantage of the keyword query is that it is conceptually simple and reduces the number of components that the user must review. Because of this simplicity many of the software component retrieval mechanisms reviewed in the next section employ some aspects of this technique.

There are two basic disadvantages to this approach. The first one is that the precision and recall of the system depend on how many keywords are used for the query. Using only one keyword typically will result in a very large number of components (high recall, low precision). Using too many keywords could miss possible candidate components (high precision, low recall).

The second disadvantage is that the user must be familiar with the structure of the keyword categories that are being used by the collection administrator to achieve maximum benefit from the system. Without such knowledge a user can easily miss potential candidates that match their needs.

C. MULTI-ATTRIBUTE SEARCH

A multi-attribute search is really just an extension of the keyword concept. Instead of using only keywords for forming a query, other attributes of the search component can be used as well. These attributes includes the class of component (procedure, function, package, etc.), the number and type of parameters used, its domain of use, etc.

The advantage of this type of system is that by using more than just keywords the search can be more selective. All of the attributes taken together make up a classification scheme that provides more information than keywords alone.

The disadvantage to this type of system is that the collection administrator must identify the attributes for stored components and the user must identify the attributes of the component that is desired. If the user succeeds in filling in the same attribute values as the administrator will a query be successful, otherwise the query mechanism must be capable of identifying when two attributes are "close" to being the same.

1. Natural Language Interfaces

Natural language interfaces for information retrieval is a growing field of computer science research. An advantage of this system is the ease in which a user describes a desired component.

The difficulty in this approach is that due to the broad semantics of the English language, implementation of these systems have had to constrain the language used to form a query. As the number of constraints on the query language grows the system begins to be more like a multi-attribute system.

2. Formal Specifications

The use of formal specifications to direct a reusable component query can be very beneficial. Because specifications systems such as SPEC and OBJ3 [Ref 11] are based on predicate calculus they are free from ambiguity. This means that formal

specifications can be transformed into normal representations without changing their meaning using logic and term rewriting rules.

Matching of specifications allows queries that achieve both high precision (formal specifications enable conclusive demonstrations that particular components do meet the requirements in a query) and high recall (through term rewriting it is possible to allow candidates with appropriate functionality to be located even if the author of the component did not anticipate the components being utilized in this context).

The primary disadvantage of this approach is that writing formal specifications for components is difficult and requires software engineers with advanced skills. Another disadvantage is that automated matching of formal specifications can be time consuming.

D. REVIEW OF CURRENT SYSTEMS

1. Draco

The Draco project was born in the early 1980's at the University of California, Irvine. The Draco approach to software reuse is essentially a multi-attribute query system. Software components are organized into problem areas or domains. Queries are constructed by the formulation of a tuple of attributes that best characterizes a particular domain. Each domain uses a different set of attributes for its queries. This type of classification of components has been called faceted classification [Ref 12].

In evaluating the effectiveness of faceted classification Draco researchers compared it to a system using no classification scheme. Using faceted classification the number of components retrieved was reduced by more than 50% yet the precision of the queries was 100%.

The advantages of faceted classification are that it is conceptually simple for users and relatively easy to implement. Because of this, the concept has been borrowed to implement the retrieval methods in both RAPID and OSS (See sections B.2 and B.3).

One of the disadvantages of this type of system is that semantically similar components may be missed because their attribute definitions are different. Draco addresses this issue by maintaining a measure of conceptual closeness for the term lists of each attribute. This allows unsuccessful searches to be tried again using an alternate but similar term for one of its attributes.

Another disadvantage of this system is that components in other domains that may be useful are easily missed. This puts the burden on the user to ensure that they have selected an appropriate domain for their search.

2. Rapid

The RAPID (Reusable Ada Packages for Information System Development) project is an ongoing effort in the Department of Defense. The objective of RAPID is to provide software engineers with quick access to reusable Ada packages in the information systems domain. The system performs reusable component classification, storage and retrieval.

RAPID uses a faceted classification scheme to organize and retrieve components and thus uses multi-attribute searches [Ref 13]. The system is currently being beta tested but no measures of performance or quality assessments are available yet.

3. Operation Support System

The Operation Support System (OSS) is an ongoing project aimed at developing an integrated software engineering environment. The system is being developed at the Naval Ocean Systems Center. One of the goals of the project is to establish a Naval software library of reusable software components.

The current prototype library subsystem allows for component retrieval using faceted classification (See section B.1), keywords, or simple textual browsing. The components currently stored in the library are for command, control, and communications and intelligence (C³I) systems. Due to the early stages of this project no information is available on the performance characteristics of the system.

4. The Reusable Software Library

The Reusable Software Library is a system design by Intermetrics to make software reuse an integral part of the software development process. Components in this system are stored in a database with attribute values that provide the basis for a search. There are two methods available to search for a component. These methods are based on multi-attribute and natural language searches.

The multi-attribute search provides a menu driven system in which the user selects the attributes desired for the search. Alternatively the user may express the query in a natural language form such as "I want a stack package for Integers." The system parses this natural language input for keywords and forms a multi-attribute query from it.

The designers of the system report [Ref 14] that the natural language front end is considerably easier to use but the search speed is significantly slower. No additional measures of performance were provided.

5. Common Ada Missile Packages

The Common Ada Missile Packages (CAMP) project is an effort sponsored by the Department of Defense to create a software engineering system and reusable software library of components. The system is directed toward software for missile systems and uses Ada as the source language for its reusable components.

The main part of the reusable component system is the Parts Engineering System (PES) catalog. The PES catalog is similar to a card catalog for books. The catalog system, used by both software engineers and domain engineers, is written in Ada and provides a menu driven interface for storing, modifying, and retrieving components (parts). Queries to this system are of the multi-attribute type.

Users select a set of attributes to search for from a predetermined finite list of values. The system then queries on each of these attributes one at a time. The results of these queries can be chained together to achieve a multi-attribute query. The CAMP documentation [Ref 15] does not assess the performance of the PES Catalog system.

6. Software Reuse At Hewlett-Packard

Hewlett-Packard recognizes the need to make software reuse an integral part of the software development process [Ref 16]. A reusable component retrieval system is currently under development to help achieve this goal. The system will have a hyper-text browsing facility as well as using informal specifications to locate reusable components.

III. CAPS AND PSDL

The Computer Aided Prototyping System (CAPS), with its accompanying specification language the Prototyping System Description Language (PSDL), is an ongoing software engineering project in the Naval Postgraduate School computer science department. CAPS is a set of software tools designed to automate the process of prototyping real-time software systems [Ref 17].

A. USING CAPS TO BUILD EXECUTABLE PROTOTYPES

The basic building blocks for a prototype in CAPS are *operators*, *types*, and *streams*. The software system being prototyped is modeled as an OPERATOR whose input and output streams correspond to the external interfaces of the system. For prototyping purposes, CAPS uses operators for software simulation of external entities as well. Based on this, the top level Data Flow Diagram (DFD) for the prototyped system is composed of an operator that represents the proposed system itself, one operator for each external entity, and the external data streams in and out of the proposed system. This top level DFD is the decomposition of a single operator that represents a closed system composed of the proposed software and all external systems that interact with the software. Figure 2 is an example of a top level DFD for a prototyped software system.

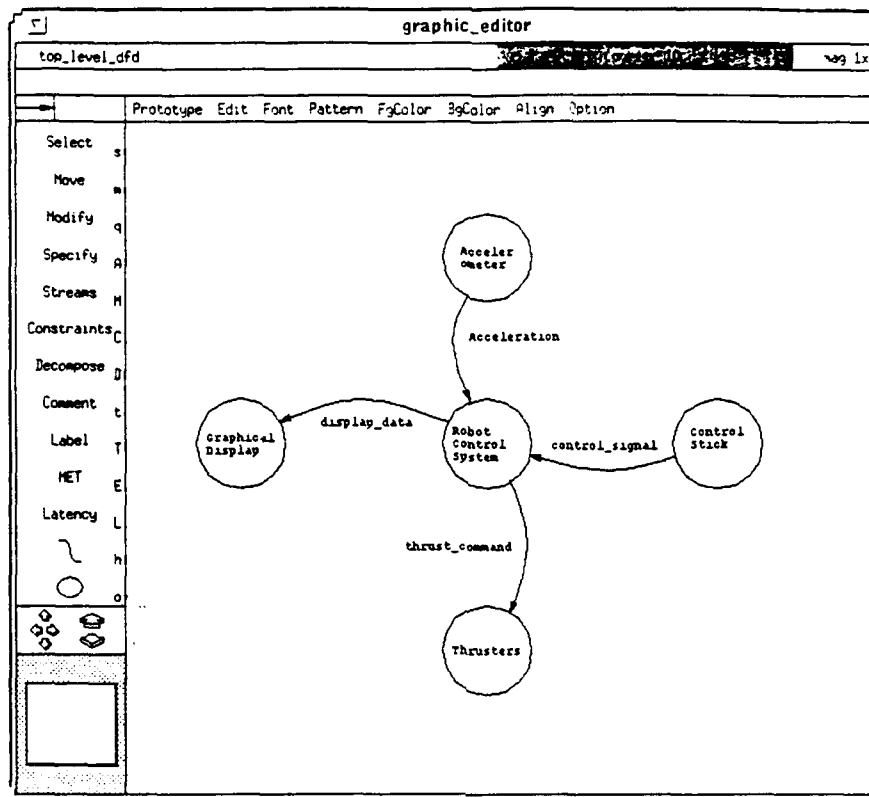


Figure - 2 CAPS Top Level DFD For Robot Prototype

CAPS prototypes are expressed in the Prototyping System Description Language (PSDL). PSDL is based on a graph model for real-time system:

$$G=(V,E,T(V),C(V))$$

where G is the graph that represents a prototype, V is the set of vertices in the graph where each vertex represents an operator in the prototype, E is the set of directed edges in the graph where each edge represents a data stream, $T(V)$ is the set of timing constraints that are imposed on the vertex set V , and $C(V)$ is the set of control constraints placed on the vertex set V [Ref. 18].

Decomposition of a prototype is achieved by implementing each of its composite operators with a graph. Each new graph G' is a more detailed representation of one of

the nodes in its parent graph G. Decomposition of operators continues in this fashion until each operator has been fully decomposed.

In order to make a prototype specified in PSDL executable, it is necessary to provide programming language implementations for all leaf operators. The current version of CAPS requires that all leaf operators be implemented in the Ada programming language. Future versions of CAPS will be capable of supporting other programming languages as well.

Each data stream in CAPS carries an instance of an abstract data type. The abstract data type for each stream is defined as a PSDL TYPE component. This definition includes all of the OPERATORS that can operate on that data type. A PSDL *type's operators* can also be graphically decomposed in the same manner as a prototype's *operators*. To make a prototype executable, all of the PSDL *type's leaf operators* must be implemented in Ada.

In the current version of CAPS the designer uses a graphical editor to design and decompose the prototype's operators [Ref 19]. Future versions of CAPS will also allow for graphical design of abstract data types.

By specifying prototypes in this manner CAPS can rapidly build an executable real-time prototype for user validation. Any deficiencies that the validation process identifies can be applied to the prototype and a new executable generated. This process can be repeated until the prototype meets all of the users needs.

B. USING REUSABLE COMPONENTS IN CAPS

To achieve maximum benefit as a rapid prototyping system it is necessary for CAPS to achieve a high rate of component reuse in the implementation of leaf operators. The software base described in this thesis has been designed to support this goal.

After the user has specified a needed *operator* or *type*, they have the opportunity to use that specification as a query to the software base to look for a potential match. If one is found it can be included in the prototype. If not the user has the choice of decomposing the component further or implementing the component manually.

In addition to automatic component retrieval facilities, the software base contains a keyword browsing feature to assist the designer in finding components in the software base to be used for manual implementation.

PSDL "was designed to serve as an executable prototyping language at the specification and design level." [Ref 17, p26] The grammar for the PSDL interface specification is not biased toward a particular programming language but rather is general enough to allow it to be extended to support any programming language.

Because of this general design it is necessary to add some pre-defined abstract data types with specific interpretations related to software reuse to PSDL (not the grammar itself), in order to adequately specify a component for automated retrieval. These extensions include a methodology for describing type inheritance and distinguishing

between different types of generic parameters. A description of how PSDL was extended to support reusable components in Ada is included in Appendix A.

IV. SOFTWARE BASE IMPLEMENTATION

A. REQUIREMENTS

The CAPS software base must perform four basic tasks [Ref 20]. Figure 3 depicts these tasks. Text file storage is a mechanism to store and retrieve portions of a reusable component. Component browsing refers to giving the user the ability to locate and view components in a manner other than by PSDL query. The ability to query the software base by PSDL specification gives the system the retrieval characteristics desired in this prototyping system. Component integration into CAPS is required once a reusable component is located so that the execution support system can produce an executable prototype.

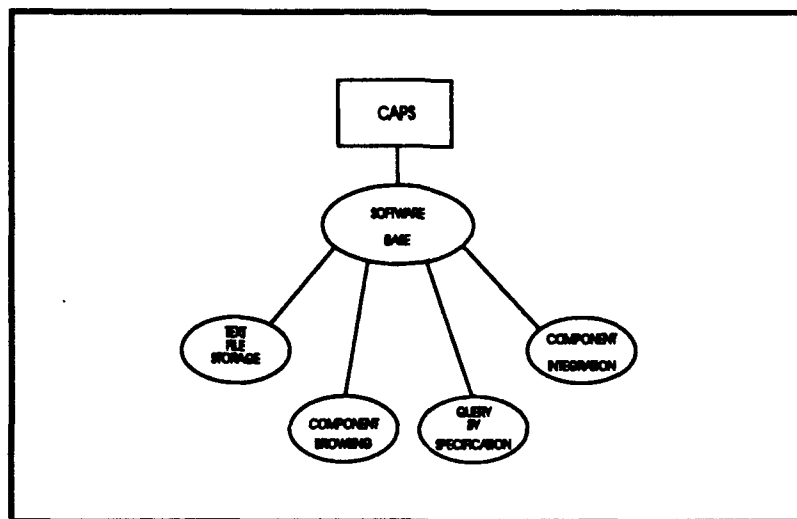


Figure 3 - Requirements for CAPS Software Base

Due to the complexity of storing variable length source code and querying the software base using PSDL specifications, a powerful DBMS system is necessary [Ref 21]. CAPS is designed to exist in a multi-user networked environment, therefore the DBMS system also needs to support multi-user, networked access to its data.

Section B of this chapter is a description of the DBMS system that was used to implement the CAPS software base. Section C describes the segregation of reusable components into language domain areas. Section D is a description of the method used to store text files in the database. Section E is a description of the implementation of the software base browsing facilities. Sections F through H describe the implementation of the query by specification function of the software base. Section I discusses the requirements for integration of components into CAPS prototypes. Section J describes a prototype graphical user interface for the CAPS software base.

B. ONTOS DATABASE MANAGEMENT SYSTEM

The Ontos database management system [Ref 22] is one of a growing number of Object-Oriented Database Management Systems (OODBMS). It was selected for use in the software base project because it has sufficient capabilities to handle the requirements for the implementation of an advanced reusable software component library.

The Ontos OODBMS is not constrained by a particular data model such as relational or hierarchical systems, but rather allows the database developer the ability to make any data object persist past the execution of the program that created it.

Persistence of objects is accomplished by assigning each object a system generated unique identifier (UID), and providing methods to store and retrieve each type of object.

The Ontos system uses C++ [Ref 23] as its implementation and application language. The database developer defines the database schema using C++ class definitions and the Ontos Classify utility. These classes are then implemented using standard C++.

In Ontos all that is needed to make an instance of a particular class persistent is to have that class inherit from the Ontos-defined class Object. The Object class constructor assigns each object a UID. The methods necessary for reading and writing instances of a persistent class are defined by the Object class, and thus inherited by all instances of persistent classes. The reading and writing of persistent objects is transparent to the application.

Ontos includes a set of persistent aggregate classes in order to efficiently handle collections of persistent objects. These aggregate classes include List, Set, Array, and Dictionary.

The List class provides functionality analogous to a linked list data structure. The Set class implements the standard concept of a set and its associated operations. The Array class implements the programming language concept of a dynamically sizable array structure.

The Dictionary class is the most robust of all Ontos aggregate classes. It is a keyed data structure that can be ordered or unordered. For every entry in a Dictionary there are

two attributes stored, the Tag and the Element. The Tag is used for indexed look up and the Element holds the desired data. Dictionaries can be defined with or without duplicate Tags being allowed. The implementation of these structures is very efficient. Dictionaries that are unordered and do not allow duplicates are implemented via hash tables. All other Dictionaries are implemented as B-tree's. Figure 4 shows the inheritance relationship between the pre-defined Ontos aggregate classes.

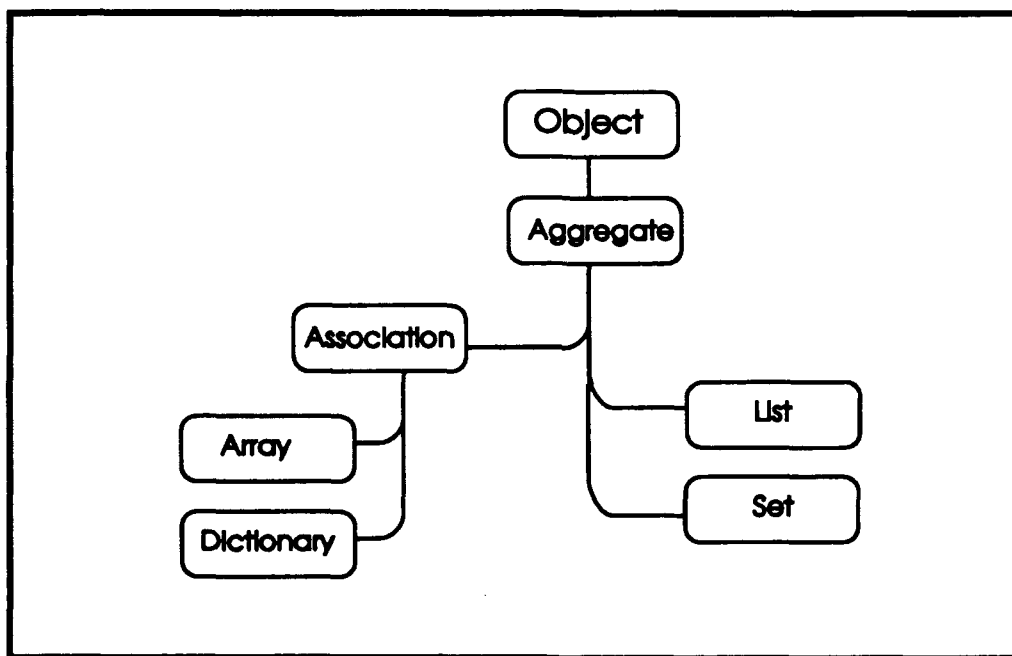


Figure 4 - Ontos Aggregate Inheritance Structure

Using aggregate classes the designer can define a database architecture that best suits the needs of the application rather than modifying the application's structure to fit a particular database model. Using the transparent referencing of the database and

aggregate data structures, the developer designs the application as though all data is immediately available when referenced in a program.

While this type of DBMS may be difficult to use where general ad-hoc query capabilities are desired, it is ideal for the development of software tools where the nature of the queries to be issued are known well in advance and the database schema must be designed to support them efficiently.

C. SEGREGATION OF REUSABLE COMPONENT DOMAINS

The CAPS software base is designed as a general purpose tool capable of storing components implemented in many programming languages. Because of differences in the capabilities of each programming language there are differences in the way the pre-defined abstract data types used in PSDL to specify components are interpreted by the software base. An example of this is that the *char* type in C++ is a subset of the type *int* while in Ada *character* is a system defined enumerated type. These differences in the interpretation of PSDL specifications require that all components of a particular implementation language be considered in a unique domain.

It is also possible to create multiple component domains for a given implementation language. This allows segregation of components into major problem areas such as information systems and control systems.

Each domain in the software base is referred to as a library and is an instance of the class SB_LIBRARY. The class SB_LIBRARY inherits from the Ontos class Object and

thus its members are persistent objects. Each instance of SB_LIBRARY is composed of five parts: a component dictionary, a keyword library, an operator library, an abstract data type library, and a recognized type matrix.

The component dictionary is used to ensure that duplicate component names are not used within a particular library. The keyword library provides the ability to formulate and process keyword queries on the domain library. The operator and abstract data type libraries are used for the query by PSDL specification and are discussed in more detail in sections F.2 and F.3. The recognized type matrix contains the type name matching information for this library domain and is discussed in section G.

The specification and implementation for the class SB_LIBRARY are in Appendix B. Figure 5 is an attribute diagram for the class SB_LIBRARY. The symbols used in this attribute diagram are the same as those used in Entity / Relationship Diagrams. Single ovals represent attributes of an object. Concentric ovals indicate a multi-valued attribute (Ontos Dictionary). The attributes shown for multi-valued objects are the contents of a single instance contained in that multi-valued object. Underlined attributes are the key or tag field of a multi-valued attribute (Ontos Dictionary).

D. STORAGE OF UNCONSTRAINED TEXT OBJECTS

In a typical development environment, program source files are stored in the operating system's directory structure as text files. This is an effective method for storing source code for a small number (< 100) of software components. As the number of

components increases however, this method becomes unacceptable. This is because the burden for maintaining the integrity of the files is placed on the users with little or no automated assistance.

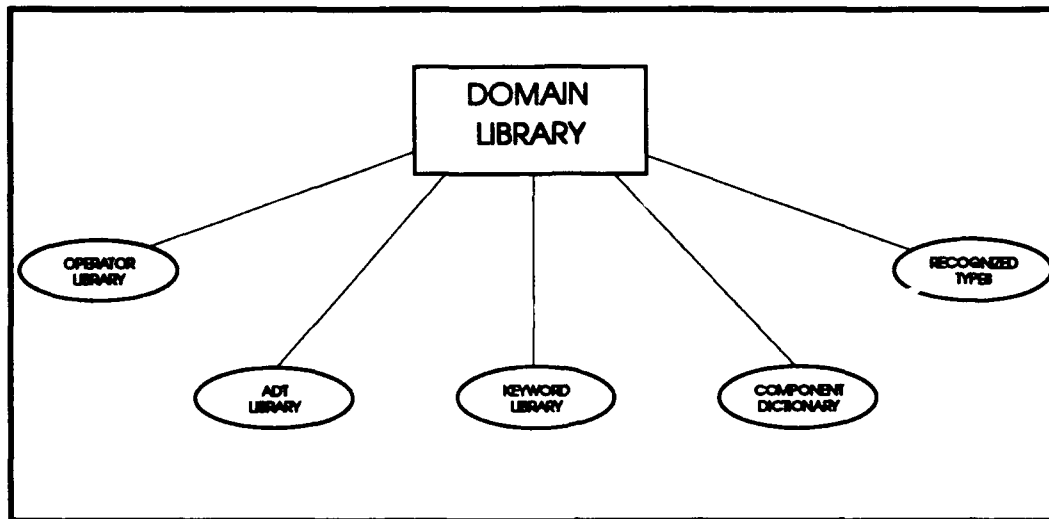


FIGURE 5 - LIBRARY ATTRIBUTE DIAGRAM

Because of the anticipated size of the CAPS software base the decision was made to encapsulate all of the component text inside of the software base itself rather than using the operating system's file structure.

For each component in the CAPS software base there are six text files that must be stored. These files are the PSDL specification source code, the implementation language specification, and the implementation body, the informal description, the axiomatic specification, and a normalized version of the axiomatic specification.

In order to store these text attributes it was necessary to design a persistent class for Ontos that would allow storage and retrieval of variable length text strings in an efficient

manner. The software base class SB_TEXT_OBJECT was developed to perform this function.

The SB_TEXT_OBJECT class supports the creation of persistent text objects and appends to these objects C++ character strings (*char **) or a C++ input stream (*ifstream&*). For output, an instance of the class SB_TEXT_OBJECT can output its text via a C++ character string (*char **) or to an output stream (*ofstream&*).

The class SB_TEXT_OBJECT is a child of the Ontos class Object and thus has all of the Ontos persistent methods for storage to and retrieval from the software base. Instances of the SB_TEXT_OBJECT class can be used as attributes of each component in the software base to store the PSDL and implementation source code. The full definition and implementation of the class SB_TEXT_OBJECT is given in Appendix B.

E. BROWSING THE SOFTWARE BASE

Although browsing by component name and keyword browsing are not the preferred methods for finding reusable components in a large software base, they are a necessary feature of any software collection. These types of features are required to allow users to familiarize themselves with the components in the software base as well as to allow the software base administrators to maintain them. Due to this need the software base was designed and implemented to support both keyword queries and named look up.

1. Named Look Up Of Components

PSDL has only two types of software components: abstract data types and operators. Each software base domain library has been divided into these disjoint categories of components. For browsing purposes the software base provides a complete list of either all abstract data types, all operators, or all components in a particular library. These lists are in alphabetical order and are used to support for named look up of individual components.

2. Keyword Querying

Each software base library includes a keyword library for handling keyword access to its components. A keyword library is an instance of the class `SB_KEYWORD_LIBRARY`. The class `SB_KEYWORD_LIBRARY` has been designed to allow the keyword attribute of PSDL to form a keyword structured method of browsing the software base.

An instance of `SB_KEYWORD_LIBRARY` provides a method for listing all keywords used in the library. From this list a keyword query can be formulated.

The result of a keyword query is a list of those components that possess one or more of the query keywords. The list is ordered with those components that satisfy the most query keywords coming first. Figure 6 graphically represents the keyword query process for a query defined by keywords A,B, and C.

The result of the query shown in Figure 6 will follow the following format:

1. All components in area 1 will be listed first (since these components contain all keywords in the query).

2. The next components in the list will be those in areas 2, 3, and 4.
3. The last components listed will be those in areas 5, 6, and 7.
4. Those components in area 8 will not be included in the list since they do not contain any of the keywords in the query.

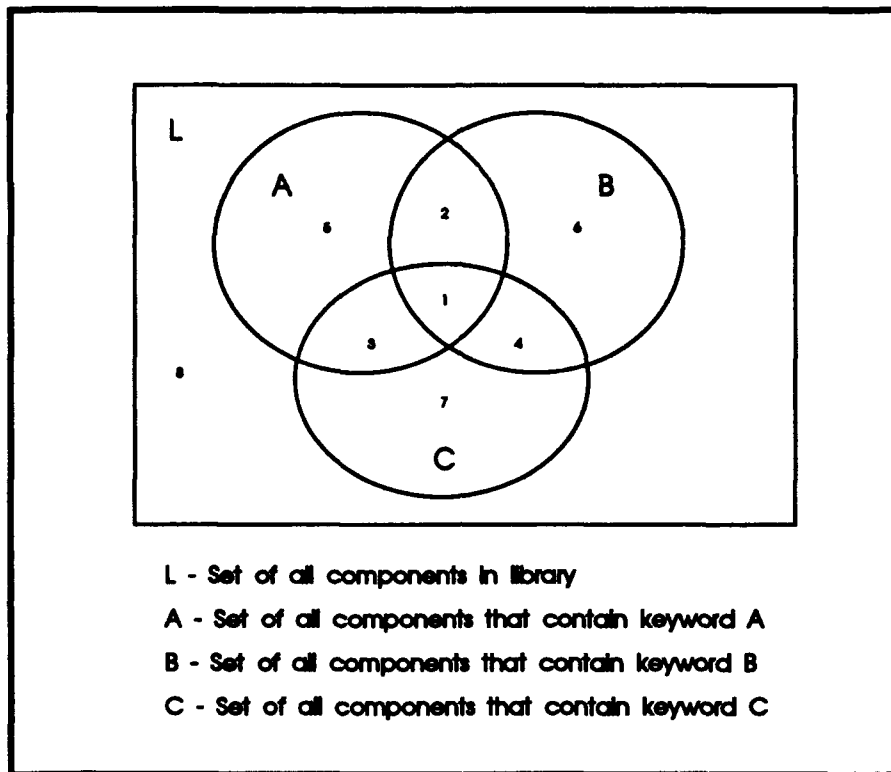


Figure 6 - Venn Diagram of Keyword Query

The class `SB_KEYWORD_LIBRARY` is a Dictionary with individual keywords as the Dictionary tags. Each tag is associated with a separate Dictionary that contains a list of components that contain that particular keyword. Figure 7 is an attribute diagram for the class `SB_KEYWORD_LIBRARY`.

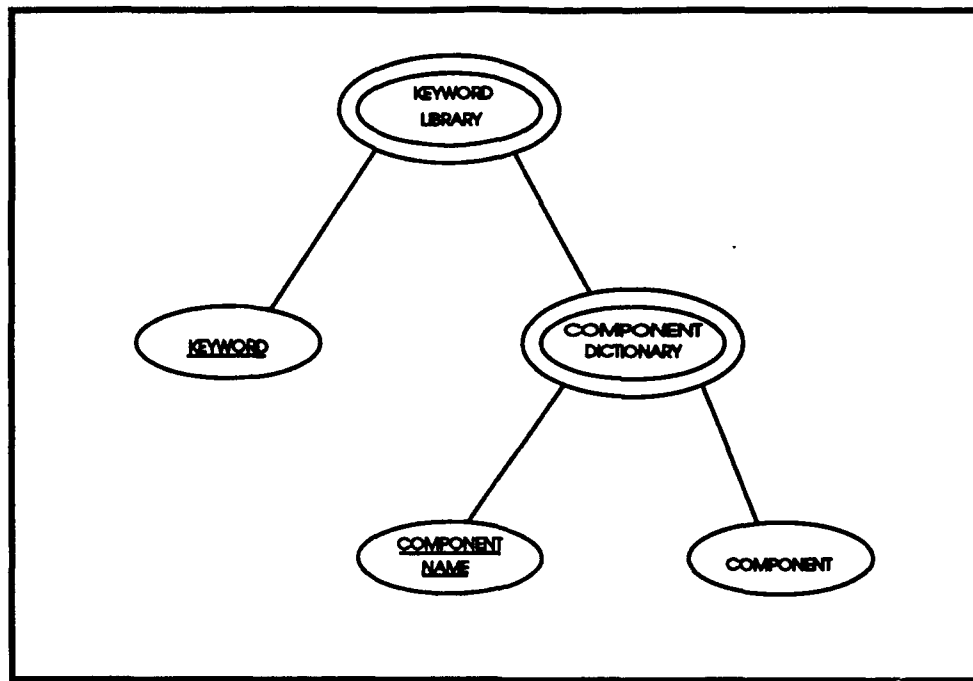


Figure 7 - Keyword Library E/R Diagram

F. QUERY BY SPECIFICATION

As stated previously, the implementation method that was chosen for the CAPS software base is to store components in a database and use PSDL specifications as the basis for high recall queries. Each stored component consists of a PSDL specification, an implementation specification, the implementation code, and a normalized version of the PSDL specification. The syntax and semantics of the PSDL specification will be used to direct the search for a component.

Figures 8 and 9 summarize the steps necessary to store components in the software base and to retrieve them using a given query specification. Components to be stored

must first pass through syntactic and semantic normalization (see Figure 8). The normalization processes transform the component's PSDL specification to facilitate later matching [Ref 24]. Syntactic normalization involves primarily format changes and statistical calculations while semantic normalization requires specification expansion and transformations.

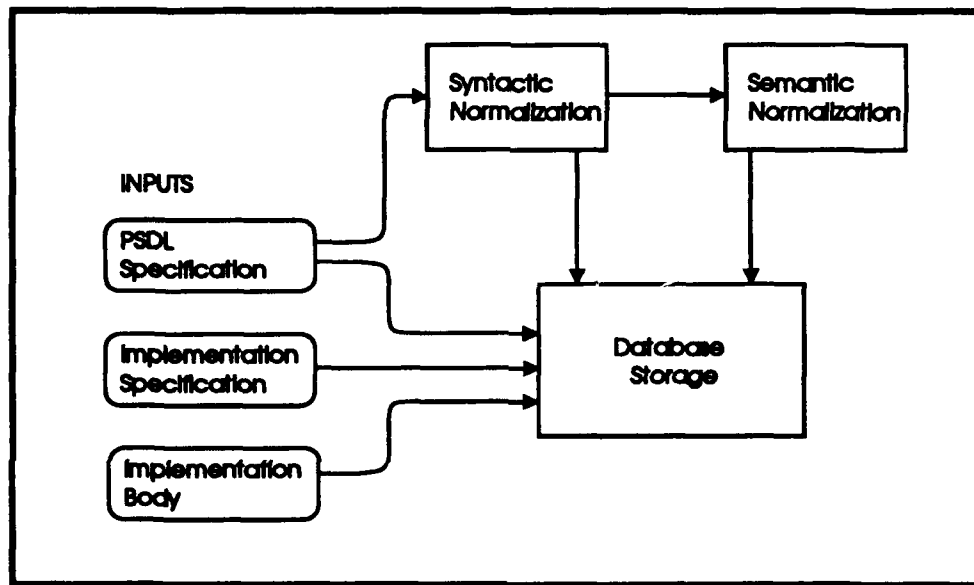


Figure 8 - Component Storage Mechanism

Figure 9 shows the general process for component retrieval. A query for a library component is formed by constructing the PSDL specification for the desired component. The query specification is syntactically and semantically normalized and then matched against the stored specifications.

Syntactic matching of the query component takes place before semantic matching. The reason for this is that syntactic matching is faster than semantic matching and will be used to partition the software base quickly in order to narrow the list of possible

candidates that the semantic matching algorithm must consider. Semantic matching is time consuming and must be applied to as small a candidate list as possible.

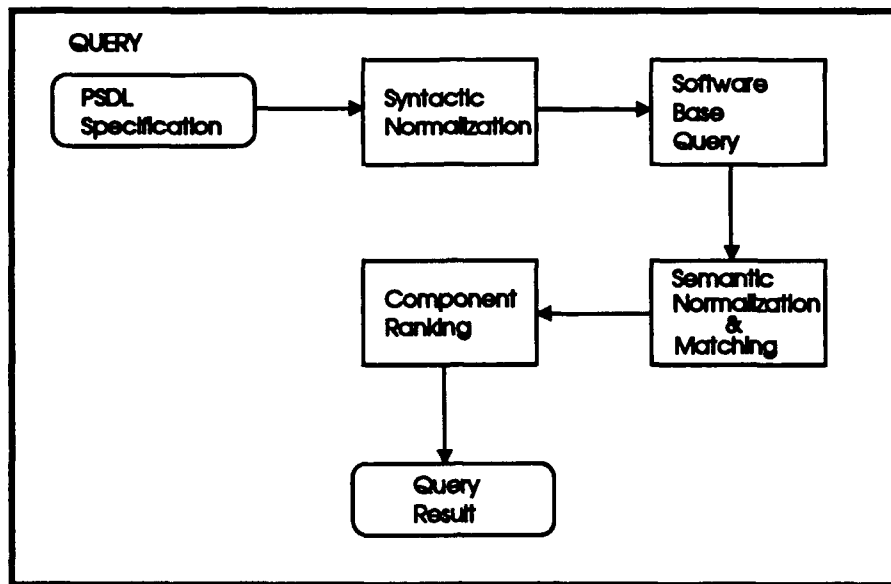


Figure 9 - Query By PSDL Specification

Both syntactic and semantic normalization and matching are required to achieve the best performance from the system. The main benefit of syntactic matching is speed whereas the advantage of semantic matching is accuracy. Accuracy is required in order to reduce the number of reusable components that a designer will have to evaluate before making a selection.

Consider the example of trying to find an abstract data type for a set. The Booch component library [Ref 25] contains 34 different variations for implementing a set. The specifications for these set packages are quite similar but the implementations are clearly different.

If we consider generic packages to perform sorting, the Booch library contains 15. Nine of the 15 Ada specifications are identical with the exception of the name given to the package. Clearly we cannot rely on syntax alone to provide us a sufficiently fine grained search. Semantics are also required.

A semantic process alone would be unacceptable because semantic matching would have to be applied to every software base component causing the search process to be impractically time consuming. For a more detailed discussion of the semantic matching mechanisms used by the software base refer to [Ref 26].

The details of the syntactic matching mechanisms employed in the CAPS software base are addressed in the following sections of this thesis.

1. Syntactic Matching

The purpose of syntactic matching is to rapidly eliminate from consideration those modules in the software base that cannot match the query specification's interface. This matching process uses the query module's PSDL interface specification to formulate a query. Once those modules with unsuitable interfaces have been removed, only a small subset of the software base needs to be semantically analyzed. The syntactic matching process reduces the number of candidate modules sufficiently to make semantic matching practical.

Prior to discussing the design of the software base architecture needed to support syntactic matching it is necessary to rigorously define what constitutes a syntactic match. PSDL allows the definition of both *type* and *operator* modules. Since a

type module is a superset of an *operator* module, the definition of an *operator* module match will be given in detail and then extended for use with *type* modules.

The attributes of a PSDL specification p for a software component c that are important to the syntactic matching process are the following:

1. $S(p) = (\{ \text{In}(t,n) : \text{there are } n > 0 \text{ occurrences of type } t \text{ as input parameters to } c \},$
 $\{ \text{Out}(t,m) : \text{there are } m > 0 \text{ occurrences of type } t \text{ as output parameters from } c \},$
 $\{ E : E \text{ is an exception defined in } c \}, \{$
 $\text{St} : \text{St is a state variable in } c \})$

$S(p)$ is the interface subset of the PSDL specification for module c and is the only part of the specification that pertains to the syntactic matching process.

Given a software base module m , and a query module q , along with their respective PSDL interface specifications $S(m)$ and $S(q)$ then m is a syntactic match for q if and only if all of the following constraints are met:

1. $\exists f_i : S(q) \rightarrow S(m) \ni [(f_i(\text{In}(t,n)_q) = \text{In}(t',m)_i) \text{ (} m=n \text{ and } (t=t' \text{ or } t' \text{ is a generic match of } t)) \text{ and } f_i \text{ is bijective}]$
2. $\exists f_o : S(q) \rightarrow S(m) \ni [(f_o(\text{Out}(t,n)_q) = \text{Out}(t',m)_o) \text{ (} m=n \text{ and } (t=t' \text{ or } t' \text{ is a generic match of } t)) \text{ and } f_o \text{ is injective}]$
3. $\text{if } (|\{STq\}| > 0 \text{ then } |\{STm\}| > 0) \text{ else } (|\{STq\}| = |\{STm\}| = 0)$

This definition of a syntactic match could be used directly to determine if a software base component could match a query specification's interface but would require the system to check every component in the software base. This type of implementation would be very inefficient. A better strategy involves using the matching rules to derive a

set of module attributes that can be used to rapidly identify and reject modules with unsuitable interfaces. Some examples of these derived attributes include:

1. If the number of input parameters in $S(q)$ is not equal to the number input parameters in $S(m)$, then there can be no function f_i to satisfy rule 1. Therefore $S(m)$ can be eliminated from the search.
2. If the number of output parameters in $S(q)$ is greater than the number of output parameters in $S(m)$, then there can be no function f_o to satisfy rule 2. Therefore $S(m)$ can be eliminated from the search.
3. If $S(q)$ has state variables defined (i.e. q defines a state machine) but $S(m)$ has no state variables, then $S(m)$ can be eliminated from the search.

If a component passes these tests, it does not mean that it is a syntactic match, a failure however, does eliminate the module from further consideration because it cannot be a syntactic match. These attributes are derivable from the PSDL specification and can be used to form multi-attribute keys. These keys allow a rapid reduction in the size of the viable subset of the software base via multi-attribute queries without the need to attempt to identify the individual mapping functions for each module. For those modules that are selected by the multi-attribute query additional checks can be made to identify components that cannot meet rules 1 and 2. These checks form a filtering mechanism that removes any unsuitable components from the query result.

The rules for the syntactic matching of *type* modules are similar to those for *operator* modules with the addition of a mapping function to map the operators of $S(q)$ to the operators of $S(m)$ and an additional check to ensure the generic parameter substitutions used for this mapping function are consistent for all operators in $S(m)$. Multi-attribute keys can be formulated that incorporate these additional requirements.

These keys can then be used for the initial *type* module database query and additional checks only applied to those modules that are selected by the multi-attribute query.

Through the use of a complex aggregate hierarchy, the software base can be separated into disjoint areas, each queriable via multi-attribute keys.

2. Operator Component Library

The class SB_OPERATOR_LIBRARY is structured to allow a multi-attribute query to be performed efficiently on the following attributes:

1. State_Flag
2. Number_of_Inputs
3. Number_of_Outputs
4. Number_of_Generic_Types / Number_of_Unrecognized_Types

The Number_of_Generic_Types attribute is for software base components and Number_of_Unrecognized_Types is the corresponding attribute for query components.

In order for a software base operator component *m* to be returned from the multi-attribute query for component *q* it must satisfy the following conditions:

1. State_Flag(*m*) = State_Flag(*q*)
2. Number_of_Inputs(*m*) = Number_of_Inputs(*q*)
3. Number_of_Outputs(*m*) >= Number_of_Outputs(*q*)
4. Number_of_Generic_Types >= Number_of_Unrecognized_Types(*q*)

The fourth requirement is due to the fact that if the software base library does not recognize a particular type in the query specification the only way that type could be matched is via a generic type.

The result of this query is a set of software base components that are potential syntactic matches of the query specification. At this point additional tests (filters) can be applied to each remaining component to determine if it should be passed to the semantic matching step. Applying these filters is an iterative process that must be carried out on one software base component at a time.

The schema for the class SB_OPERATOR_LIBRARY is shown in figure 10.

3. Abstract Data Type Library

The abstract data type library is similar to the operator component library. It is an instance of the class SB_ADT_LIBRARY and uses the following attributes for multi attribute queries:

1. Number_of_ADTs
2. Total_Number_of_Inputs
3. Total_Number_of_Outputs
4. Total_Number_of_Generic_Types / Total_Number_of_Unrecognized_Types
5. Number_of_Operators

In order for a software base operator component m to meet an attribute query for component q all of the following must be true:

1. Number_of_ADTs(m) $\geq$ Number_of_ADTs(q)
2. Total_Number_of_Inputs(m) $\geq$ Total_Total_Number_of_Inputs(q)
3. Total_Number_of_Outputs(m) $\geq$ Total_Number_of_Outputs(q)
4. Total_Number_of_Generic_Types(m) $\geq$
Total_Number_of_Unrecognized_Types(q)
5. Number_of_Operators(m) $\geq$ Number_of_Operators(q)

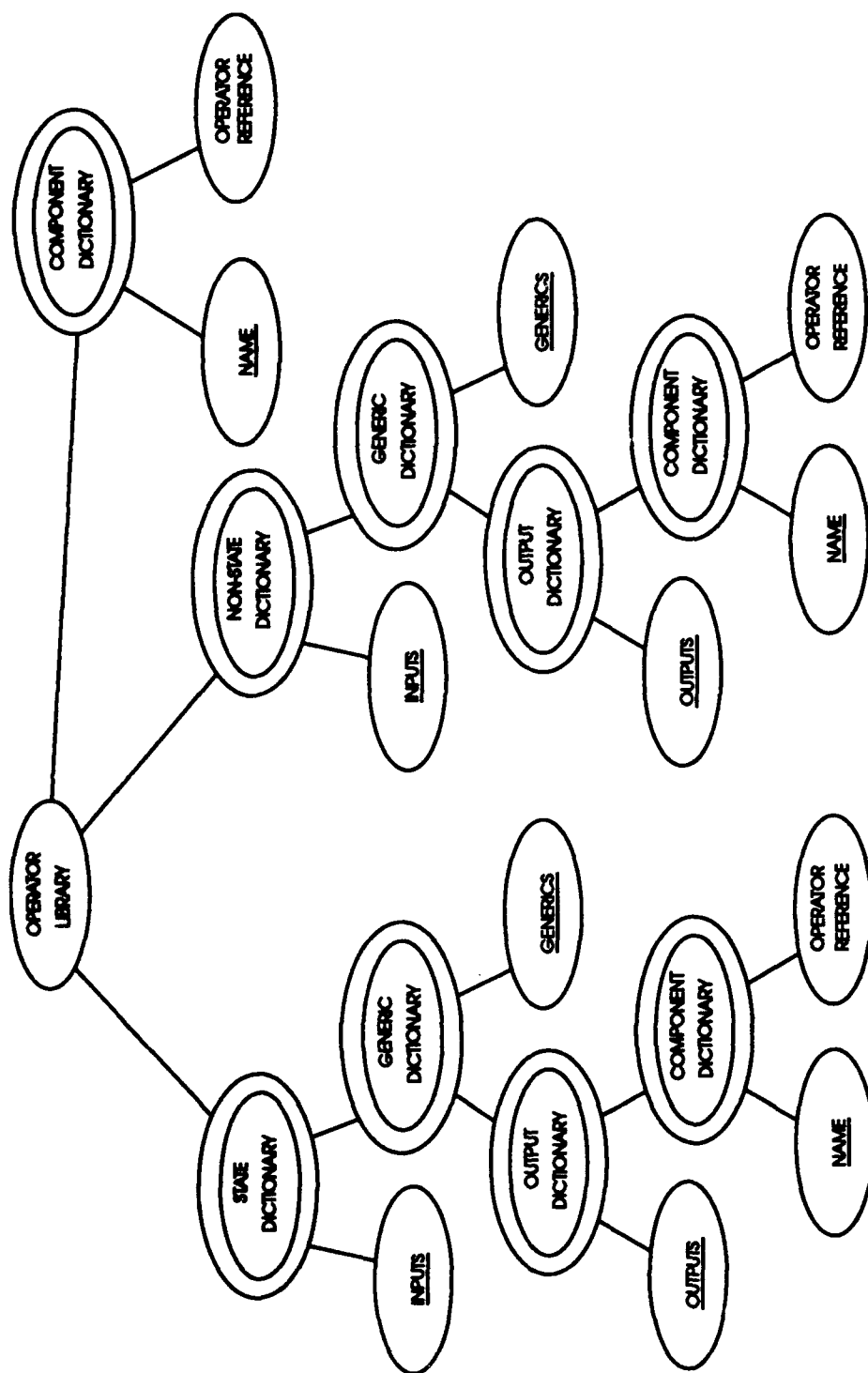


Figure 10 - Operator Library Attribute Diagram

The rationale behind requirement 5 is the same as for the operator query. Again the results of the multi-attribute query is a set of type components that are potential maps for the query component. The schema for the class SB_ADT_LIBRARY is shown in Figure 11.

G. DATA STREAM TYPE MATCHING

One of the critical concepts in the syntactic matching methodology is the determination if a library component stream data type is a match of a query component stream data type. The criteria for making this decision differs for each implementation language because they each have their own set of predefined data types and inheritance techniques.

In order to identify if one stream type can map into another stream type, each library contains an instance of the class SB_RECOGNIZED_TYPES. This class contains the names of all of the type identifiers the library recognizes along with a matrix for determining whether a given type can map into another type that the system recognizes. This matrix represents all of the subtype relationships among the recognized types.

The direction of the mapping is important as illustrated by the following example. In Ada the subtype *Natural* is defined as the range from 0..Integer'Max and *Positive* is defined as 1..Integer'Max.

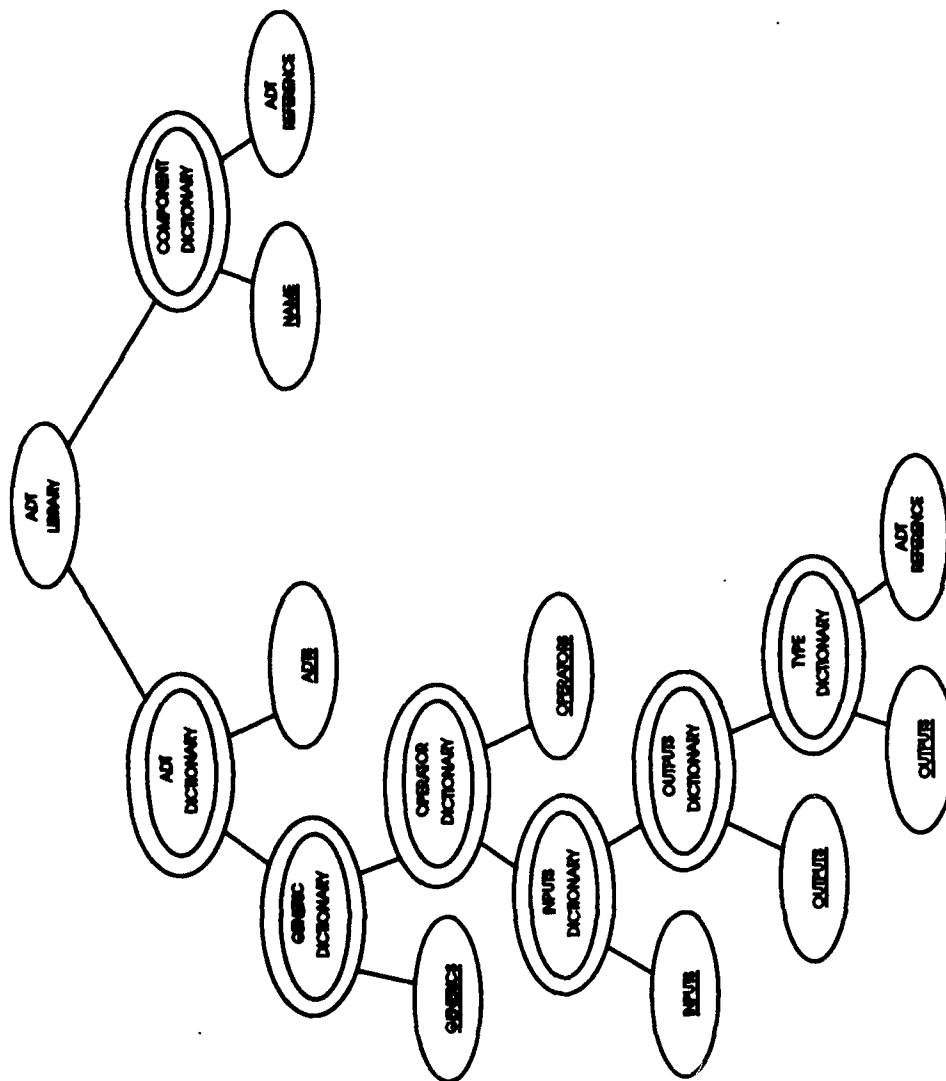


Figure 11 - ADT Library Attribute Diagram

A *Positive* data type in an input stream of a query can map into a *Natural* in an input stream of a stored component since all of the *Positive*'s allowed values are also valid *Natural* values. A *Natural* however cannot map into a *Positive* because 0 is not a valid *Positive* number. The situation is reversed for output streams.

The `SB_RECOGNIZED_TYPES` class also contains information about how some standard programming language concepts will be identified. These include:

1. Whether or not the language is case sensitive.
2. How type inheritance will be identified.
3. The base type name for generic types, values, and procedures.
4. The base type name for abstract data types.
5. How array types will be specified (including the index type and element type).

An example of the type matrix for Ada and its use is presented in Appendix A.

H. ABSTRACT REPRESENTATION OF SOFTWARE BASE COMPONENTS

The class `SB_COMPONENT` is an abstract base class for storing the attributes of software base components. It includes attributes that are common to all software components. The classes `SB_ADT_COMPONENT` and `SB_OPERATOR` inherit from `SB_COMPONENT` and include additional attributes that are specific to each.

Two classes inherit from `SB_OPERATOR`. These are `SB_OPERATOR_COMPONENT` and `SB_ADT_OPERATOR`. These two classes differ only in the methods for handling generic and recognized types.

Figure 12 shows the inheritance hierarchy used in defining the persistent classes for software base components. Figure 13 is the schema for the class

SB_ADT_COMPONENT and Figure 14 is the schema for the class SB_OPERATOR respectively.

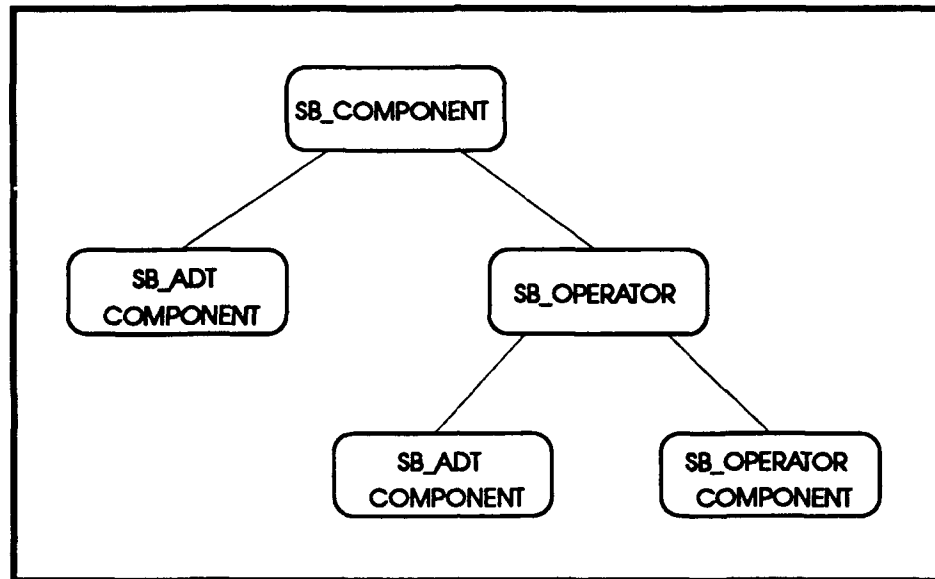


Figure 12 - SOFTWARE BASE COMPONENT INHERITANCE

Each of these schemas contain some derived attributes. These derived attributes are stored in the software base to prevent them from being recomputed each time they are needed.

A parser for the specification subset of PSDL was developed using lex [Ref 27] and yacc [Ref 28] in order to construct instances of the SB_ADT_COMPONENT and SB_OPERATOR_COMPONENT classes. For this parser to take the appropriate semantic actions, language preserving transformations of the original PSDL grammar were necessary. These transformations consist of the addition of non-terminals and productions to allow appropriate semantic actions to be carried out.

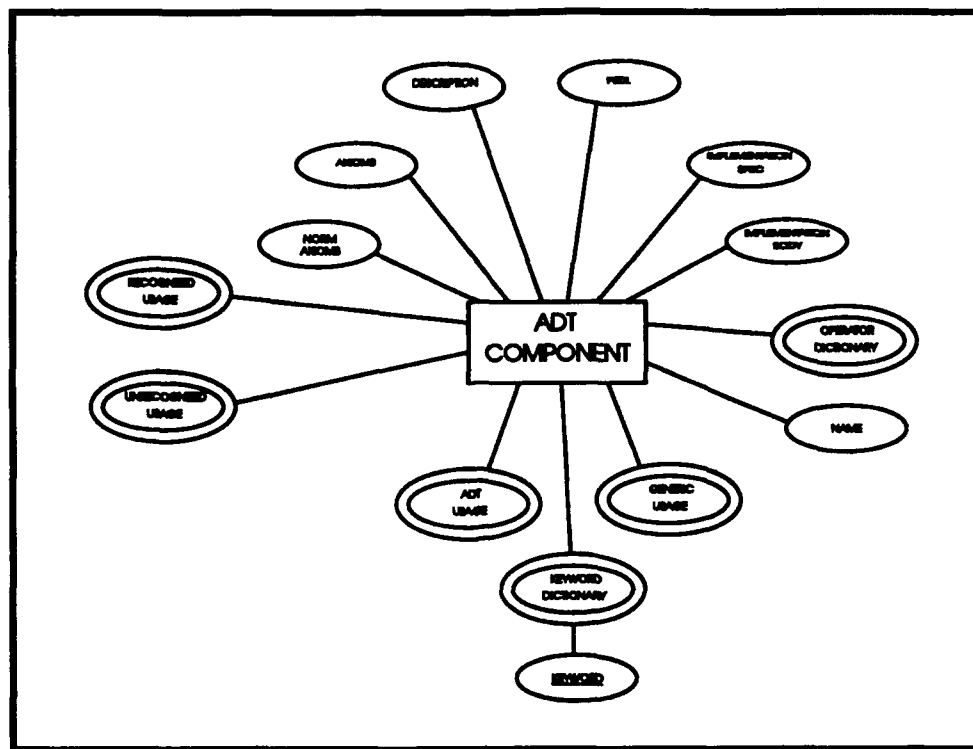


Figure 13 - ADT Component Attribute Diagram

The source code input to lex and yacc that was used to generate the parser is included in Appendix C.

I. INTEGRATING RETRIEVED COMPONENTS INTO CAPS

The goal of the software base is to provide to CAPS a component implementation that is an exact match for a query specification and meets the needs of the CAPS execution support system. To accomplish this, once a reusable software component has been located it must be transformed into a form that matches all of these requirements.

This transformation involves changing parameter, type, and operator names of the library component to match those of the query specification as well as instantiating any generics.

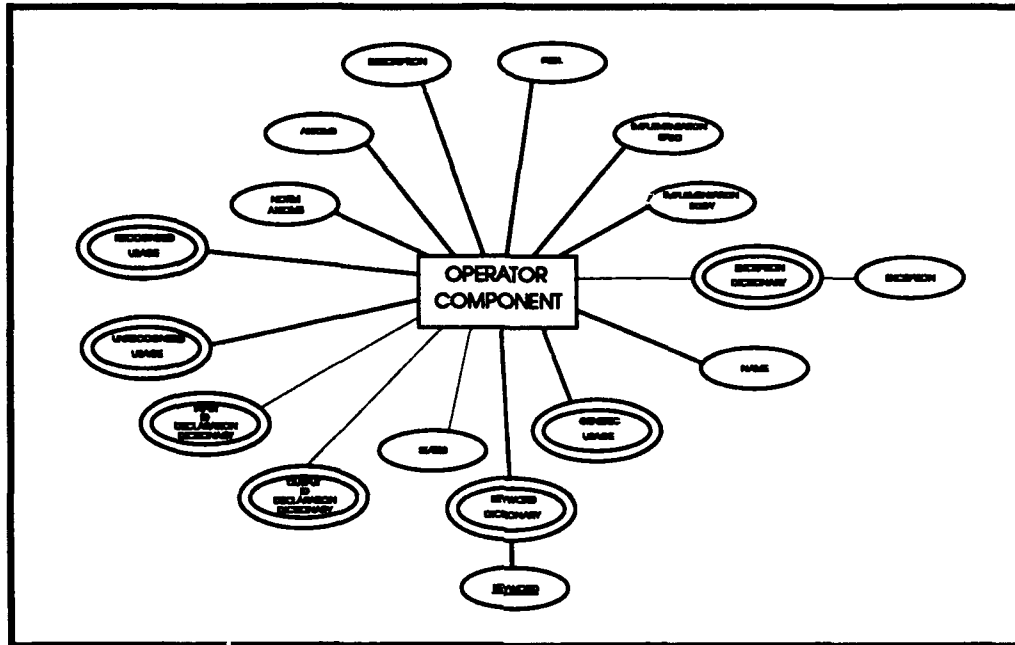


Figure 14 - Operator Component Attribute Diagram

Rather than modifying the library component itself, the library unit can be used as a basis for the creation of a separate component that meets the needs of the query component. This is accomplished via inheritance or using the *with* statement in Ada.

The software base cannot directly generate implementation code because it is not language specific. It can generate an abstract representation of how the library component satisfies the syntax and semantics of a query component. This representation can then be used by a translation tool specific to a particular implementation language to generate the implementation code. Figure 15 shows the details of the integration process.

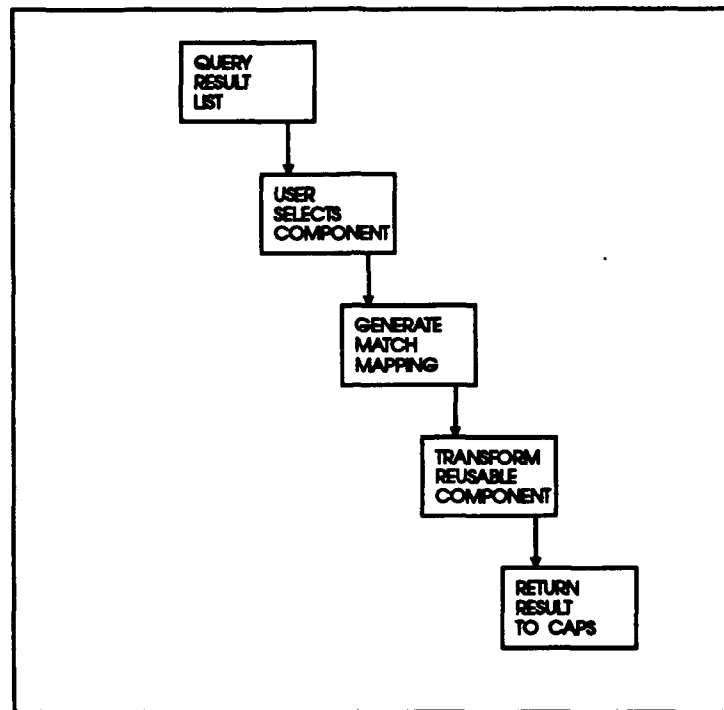


Figure 15 - Integration of Components Into CAPS

This method of component integration is preferable since additional implementation languages can be added to the software base as long as a translation tool to generate the final implementation is provided. Appendix D provides a specification for a proposed mapping grammar that can be generated by the semantic matching system and used for generation of component implementation. Appendix D also gives an example of this process to generate an implementation for an abstract integer set using a generic set package.

J. SOFTWARE BASE INTERFACE

CAPS itself is a set of individual tools. These tools are linked together with a *tool interface*. One of the CAPS tools is a graphical user interface. The user graphical interface gains access to all of the other tools in the system via the tools interface. The reason for having all graphical user interface functions in a single tool is to simplify future enhancements to the interface.

Based on this structure, each tool in the CAPS system provides a command line interface that is used by the tool interface to invoke the tool. The software base has been designed with an interface that meets this requirement.

1. Command Line Interface

The software base implementation provides a command line interface. This type of interface supports easy integration of the software base functions into the CAPS system. The functions provided by the software base command line interface are the following:

1. Make a new domain library.
2. Add a new software base component.
3. Update a software base component.
4. Delete a software base component
5. Generate a list of components in a library
6. Generate a list of operators in a library
7. Generate a list of types in a library
8. Generate list of keywords in a library

9. Keyword query
10. Component query
11. View a component's source files
12. Output diagnostic information (for testing and maintenance only)
13. Generate a component's mapping for a given query

A function to generate a mapping for a query has not been implemented in the current version of the software base. For details on the exact syntax of these commands refer to Appendix E.

2. Graphical User Interface

In order to demonstrate the capabilities of the software base system and the command line interface, a graphical user interface was prototyped for the software base system using the Interviews 3.0b [Ref 29] interface builder application and the Interviews 3.0b object library. This interface is not intended to be full functioning but rather an example of the functionality of the software base system.

Appendix F is the user's manual for the prototype software base graphical interface. Appendix G is the source code for this graphical user interface.

V. CONCLUSIONS AND FUTURE RESEARCH

The software base system described in this thesis has been implemented. It has not yet been integrated into CAPS. Due to the complexity of the software base system, there are many areas that can be improved by future research. This chapter identifies those areas that need improvements and provides recommendations where possible.

A. ADDING COMPONENTS TO THE SOFTWARE BASE

Reusable components are currently being selected and tested for possible inclusion into the software base. This is a labor intensive activity for several reasons.

1. Component Testing

Any component that is added to the software base must be adequately tested to ensure that it fully meets its specification. Testing software components continues to be a difficult area in software engineering research and further advances in testing are necessary to make reusable component libraries more successful. Some relevant research in this direction is provided in [Ref 30].

2. Component Implementation Restrictions

The CAPS system restricts the nature of the components used in prototypes. The first restriction is that implementations of *OPERATORS* must be procedures rather than functions. The second restriction is that "in out" parameters are not allowed. These

restrictions necessitate the modification of most existing components that are candidates for reuse. This includes sources such as the Booch library, the Ada software repository, the RAPID project, the CAMPS project etc. One method of overcoming this difficulty is automate the modification process using a translation tool.

3. Writing Formal Specifications Of Existing Components

CAPS and the software base system require that reusable components be specified in two additional specification languages. These are PSDL and the OBJ3 used for semantic matching. Writing these specifications is a time consuming process that could be partially automated by using the implementation language's specification to generate skeletons of PSDL and OBJ3 interface specifications.

B. DELETING AND UPDATING COMPONENTS

Updating or deleting components from the software base could cause system inconsistencies. These inconsistencies take two forms. The first involves other software base components that may depend on the deleted or updated component. The second is that previously generated prototypes may depend on the component that has been deleted or modified. The current implementation of the software base relies on the software base administrator to ensure that these conditions do not arise. This process should be automated to ensure that the inconsistencies do not occur.

To correct the first problem it is necessary to add dependency relationships between software base components and the software base. This would allow the software base to ensure that all updates and deletions do not create inconsistencies.

To correct the second problem it is necessary to save the source code for all deleted or updated components external to the software base. This could be accomplished through the use of a version control system such as SCCS.

C. EFFICIENCY

The most time consuming portion of a software base query is semantic matching. The easiest way to improve overall query performance is to reduce the number of components that must be analyzed by the semantic matching system.

As more components are added to the software base, experience will be gained on the performance of the syntactic matching system. This experience will make it possible to identify additional attributes for the multi-attribute queries and to add more detail to the post-query filtering routines. These additions will reduce the number of candidates passed to the semantic matching system and thus increase overall query performance.

D. SOFTWARE BASE SYSTEM IMPLEMENTATION LANGUAGE

The long term goal for CAPS is that it be entirely implemented in Ada. A major portion of the software base system is currently implemented in C++. C++ was used because there does not exist a tool with the capabilities of Ontos that interfaces directly to

Ada. As more robust database tools become available for Ada, it will be possible to re-implement the software base tool fully in Ada. A DBMS of this type is currently being developed which could be used for future versions of the software base [Ref 31].

LIST OF REFERENCES

1. Luqi, M. Ketabchi, *A Computer Aided Prototyping System*, IEEE Software, March 1988, pp. 66-72.
2. Elbert, T. F., *Embedded Programming in Ada*, Van Nostrand Reinhold Publishing Co., Inc., New York, NY, 1986.
3. Yourdon, E., *Modern Structured Analysis*, Yourdon Press, Englewood Cliffs, NJ, 1989.
4. Military Standard Defense System Software Development, DOD-STD 2167a, February, 1988.
5. Berzins, V. and Luqi, *Software Engineering with Abstractions*, Addison-Wesley Publishing Company, Reading, MA, 1990.
6. Luqi, *Computer Aided Software Prototyping*, IEEE Computer, September, 1991, p. 111-112.
7. Luqi, *Software Evolution via Rapid Prototyping*, IEEE Computer, May 1989, pp. 13-25.
8. Luqi, *Knowledge-Based Support for Rapid Prototyping*, IEEE Expert, Fall 1988, pp. 9-18.
9. Luqi, V. Berzins, R. Yeh, *A Prototyping Language for Real-Time Software*, IEEE Transactions on Software Engineering, October, 1988, Vol. 14, No. 10, pp. 1409-1423.
10. Salton, G. and McGill, M., *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983.
11. Goguen, J. A., SRI International Report SRI-CSL-88-4R2, *OBJ as a Theorem Prover with Applications to Hardware Verification*, August, 1988.
12. Prieto-Diaz, Ruben, and Freeman, Peter, *Classifying Software for Reusability*, IEEE Software, v. 4, pp. 6-16, January, 1987.
13. Vogelsong, T. and Rothrock, J., *Reusable Ada Products for Information Systems Development (RAPID) Lessons Learned During Pilot Operations*, U.S. Army Information Systems Software Development Center - Washington, 1990.
14. Burton, B., Wienk, R., and others, *The Reusable Software Library*, IEEE Software, v. 4, pp. 25-33, July, 1987.

15. Air Force Armament Laboratory, Contract F08635-88-C-0002, CDRL No. A009, *CAMP Parts Engineering System Catalog User's Guide*, McDonnell Douglas Missile Systems Company, 1989.
16. Griss, G. L., *Software Reuse at Hewlett-Packard*, OOPSLA'91 Proceedings, Phoenix, AZ, October 6, 1991.
17. White, L. J., *The Development of a Rapid Prototyping Environment*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, December 1989.
18. Berzins, V., and Luqi, *Rapidly Prototyping Real-Time Systems*, IEEE Software, September 1988.
19. Cummings, M. A., *The Development of User Interface Tools for the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, June 1990.
20. D. Galik, Luqi, *A Conceptual Design of a Software Base Management System for the Computer Aided Prototyping System*, Technical Report NPS 52-89-002, Computer Science Department, Naval Postgraduate School, 1989.
21. J. Huskins, *Issues in Expanding the Software Base Management System Supporting the CAPS*, M.S. Thesis, CS, Naval Postgraduate School, June, 1990.
22. Ontologic Inc., *Ontos Object Database Documentation Release 1.5*, Bulington, MA, 1991.
23. Stroustrup, B., *The C++ Programming Language*, Addison Wesley, 1986.
24. Luqi, *Normalized Specifications for Identifying Reusable Software*, Presented at ACM-IEEE Computer Society 1987 Fall Joint Computer Conference, Refereed Paper, Conference Proceedings, pp. 46-49, Dallas, TX, October, 1987.
25. Booch, G., *Library of Reusable Ada Components*, Wizard Software, Lakewood, CO, 1990.
26. B. Steigerwald, Luqi, J. McDowell, *A CASE Tool for Reusable Software Component Storage and Retrieval in Rapid Prototyping*, Vol 38, #11, Nov. 1991, Information and Software Technology, England.
27. Lesk, M. E. and Schmidt, E., *Lex - A Lexical Analyzer Generator*, AT&T Bell Laboratories, Murray Hill, NJ.
28. Johnson, S. C., *Yacc: Yet Another Compiler-Compiler*, AT&T Bell Laboratories, Murray Hill, NJ.
29. Computer System Laboratory Department of Electrical Engineering and Computer Science Stanford University, *Interviews Reference Manual, Version 3.0-beta*, Stanford, CA, 1991.

30. Depasquale, J., *Design and Implementation of Module Driver and Output Analyzer Communication*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, June 1990.
31. Persistant Data Systems Inc., *IDB User's Manual, IDB Version 1.0*, January, 1991.

APPENDIX A - USING PSDL TO SPECIFY REUSABLE COMPONENTS

The ability to accurately specify reusable components with PSDL is critical to the success of the software base. Due to the general nature of the PSDL interface specification its use for specifying specific programming languages must be refined. The software base is designed to recognize the enumeration of PSDL for any language in the following areas.

A. STRUCTURE OF EXTENSIONS TO PSDL INTERFACE SEMANTICS

1. Generic Parameters

In languages that support the concept of generic units, such as Ada, or in macro expansion facilities, there are three categories of generic parameters. These are generic types, generic values, or generic program units.

The generic specification structure in PSDL must be extended to identify a particular generic parameter as either a type, value, or program unit. This is accomplished in the software base by defining of three identifiers that have special significance in the generic structure.

2. Abstract Data Types

There are cases where in the definition of one abstract data type it is necessary to define others as well. Programming languages such as Ada allow an individual package

to define an unlimited number of abstract data types. PSDL TYPE's can specify multiple abstract data type structures through the definition of an identifier that has a special meaning in the type declaration structure of a PSDL TYPE.

3. Type Inheritance

Most modern programming languages support user defined types. In many cases user defined types actually inherit from a predefined language type and the new type retains compatibility with its parent. An example of this is the subtype construct in Ada.

The software base needs to be able to identify when a user defined type is compatible with a predefined type or another user defined type. One way of achieving this is to allow the ability to specify from what base type a user defined type inherits. This inheritance identification is achieved in the software base through the definition of an identifier that has special meaning in the type name construct of PSDL.

4. The Array Abstract Data Type

The concept of an array of data is present in almost all programming languages. Because of this, the decision was made to add the definition of special identifiers in PSDL to allow the software base to decide when two array types are compatible. The identifiers are used to identify the type of the index of the array as well as the type of the element of the array.

B. EXAMPLE DEFINITIONS FOR ADA

For a particular language library the definitions of the special identifiers are contained in the rule file used when the library is created. The rule file defined for Ada follows this description of a rule file's contents. The first field in the rule file indicates if the language being defined is case sensitive (1) or not (0).

The next six rules define the special identifiers for the following concepts.

1. Type that must be matched generically
2. Inheritance
3. Generic type
4. Generic program unit
5. Generic value
6. Abstract data type
7. Array
8. Array index
9. Array element

These identifiers must all be defined and in this order.

Following these identifiers is a list of type names that the designer wants the software base library to recognize in this library. The list is terminated with a "~".

Following the ~ is a matrix of boolean (0 or 1) values concerning type compatibility. This matrix is constructed by listing all identifiers above the ~ to identify the rows and columns of the matrix. A value of 1 at (row x, column y) indicates that type x can map into the type y.

C. ADA RULE FILE

```
0
UNRECOGNIZED
BASE_TYPE
GENERIC_TYPE
GENERIC_PROCEDURE
GENERIC_VALUE
ADT
ARRAY
ARRAY_INDEX
ARRAY_ELEMENT
DISCRETE
RANGE
DIGIT
DELTA
PRIVATE
ENUMERATION
INTEGER
NATURAL
POSITIVE
BOOLEAN
FLOAT
FIXED
CHARACTER
STRING
ACCESS
~
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0
```

```

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

```

D. EXAMPLE PSDL SPECIFICATIONS USING ADA RULES.

1. Bubble Sort Operator

E. PSDL

```

operator bubble_sort
specification
  generic
    array_type : GENERIC_TYPE[
      BASE_TYPE : ARRAY[
        ELEMENT : PRIVATE, INDEX : DISCRETE]],
    less_than : GENERIC_PROCEDURE
  input
    the_array : array_type
  output
    the_array : array_type
  keywords array,sort,bubble
  description { Booch library bubble sort }
end

```

F. ADA SPECIFICATION

```

--
-- (C) Copyright 1986, 1987, 1988, 1989, 1990 Grady Booch
-- All Rights Reserved
--
-- Serial Number 0100219
--
-- "Restricted Rights Legend"
-- Use, duplication, or disclosure is subject to
-- restrictions as set forth in subdivision (b) (3) (ii)
-- of the rights in Technical Data and Computer
-- Software Clause of FAR 52.227-7013. Manufacturer:
-- Wizard software, 2171 S. Parfet Court, Lakewood,
-- Colorado 80227 (1-303-987-1874)
--

```

```

generic
  type Item is private;
  type Index is (<>);
  type Items is array(Index range <>) of Item;
  with function "<" (Left : in Item;
                    Right : in Item) return Boolean;
package Bubble_Sort is

  procedure Sort (The_Items : in out Items);

end Bubble_Sort;

```

1. Set Abstract Data Type

G. PSDL SPECIFICATION FOR SB_SET_PKG

```
type sb_set_pkg
specification
  generic
    t : GENERIC_TYPE,
    block_size : GENERIC_VALUE,
    eq : GENERIC_PROCEDURE

  set : ADT

  operator empty
  specification
    output
      s : set
  end

  operator add
  specification
    input
      x : t,
      si : set
    output
      so : set
  end

  operator remove
  specification
    input
      x : t,
      si : set
    output
      so : set
  end

  operator member
  specification
    input
      x : t,
      s : set
    output
      v : boolean
  end

  operator union
```

```

specification
    input
        s1,s2 : set
    output
        s3 : set
end

operator difference
specification
    input
        s1,s2 : set
    output
        s3 : set
end

operator intersection
specification
    input
        s1,s2 : set
    output
        s3 : set
end

operator size
specification
    input
        s : set
    output
        v : natural
end

operator equal
specification
    input
        s1,s2 : set
    output
        v : boolean
end

operator subset
specification
    input
        s1,s2 : set
    output
        v : boolean
end

keywords SET

description { SET ADT WITH OPERATIONS FOR EMPTY, ADD, SUBSET, EQUAL }
end

```

H. ADA SPECIFICATION FOR SB_SET_PKG

```
with text_io; use text_io;

generic
  type t is private;
  block_size: in natural:=128;
  with procedure eq(x,y: in t, v : BOOLEAN);
package sb_set_pkg is

  type set is private;
  type index_array is array(natural range <>) of natural;

  procedure empty(s: out set);
  procedure add(x: in t; si: in set, so : out set);
  procedure remove(x: in t; s: in out set);
  procedure member(x: in t; s: in set, v : boolean);
  procedure union(s1, s2: in set; s3: out set);
  procedure difference(s1, s2: in set; s3: out set);
  procedure intersection(s1, s2: in set; s3: out set);
  procedure size(s: in set, v : out natural);
  procedure equal(s1, s2: in set, v : out boolean);
  procedure subset(s1, s2: in set, v : out boolean);

  private

  type link is access set;
  type elements_type is array(1..block_size) of t;

  type set is
    record
      size: natural:=0; --The size of the set
      elements: elements_type; --The actual elements of the set
      next: link:=null; --The next node in the list
    end record;
  --Elements(1..min(size,block_size)) contains data

end sb_set_pkg;
```

APPENDIX B - C++ SOURCE CODE FOR SOFTWARE BASE

The source code for the software base included in this Appendix was formatted using the c++2latex code formatting system written by Norbert Kiesel. His program was modified to have it generate output that conforms to the requirements of the Naval Postgraduate School thesis format.

```

//-----
// CAPS REUSABLE COMPONENT RETRIEVAL SYSTEM CLASS DEFINITIONS
//
// J. K. MCDOWELL 23 AUG 91
//
//-----
//

#include <stream.hxx>
#include <strstream.hxx>
extern "C--"
{
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
}

//-----
//
//-----

# include <Dictionary.h>
# include <Array.h>
# include <List.h>
# include <TRef.h>
# include <Type.h>
# include <Database.h>
# include <Directory.h>
# include <GlobalEntities.h>

//-----
// PRE-DECLARE ALL CLASSES TO AVOID PROBLEMS WITH DECLARATION ORDER
//-----

class SB_LIBRARY; //sbl.cxx
class SB_ADT_COMPONENT_LIBRARY; //sbac.cxx
class SB_OPERATOR_COMPONENT_LIBRARY; //sbocl.cxx
class SB_KEYWORD_LIBRARY; //sbkwl.cxx
class SB_COMPONENT; //sbc.cxx
class SB_COMPONENT_DICTIONARY; //sbcd.cxx
class SB_KEYWORD_DICTIONARY; //sbkwd.cxx
class SB_TEXT_OBJECT; //sbto.cxx
class SB_ADT_COMPONENT; //sbac.cxx
class SB_OPERATOR; //sbo.cxx
class SB_OPERATOR_COMPONENT; //sboc.cxx
class SB_ADT_OPERATOR; //sbao.cxx
class SB_ID_DECL_DICTIONARY; //sbidd.cxx
class SB_ID_DECL; //sbid.cxx
class SB_TYPE_USAGE; //sbtu.cxx
class SB_TYPE_USAGE_DICTIONARY; //sbtud.cxx

```

```

class SB_TYPE_NAME; //sbtn.cxx
class SB_ADT_OPERATOR_DICTIONARY; //sbaod.cxx
class SB_EXCEPTION_DICTIONARY; //sbed.cxx
class SB_RECOGNIZED_TYPES; //sbrt.cxx

//-----
// SOFTBASE NAMEING CONVENTIONS
//
// A. all softbase class names start with "SB_" this eliminates any
// potential name space conflicts with any other software.
// B. all TRef instances start with "the_" and there dereferencing
// functions use the rest of the name. (ie. TRef*the_name,
// SB_* name() )
//-----

//-----
// to eliminate confusion between PSDL types and stream types
// PSDL types are refered to as abstract data types (ADT) and
// stream types are refered to as simply types
//-----

```

```

class SB_LIBRARY : public Object
{
private:

    TRef *the_adt_component_library;
    DictionaryIterator adt_iterator();
    SB_ADT_COMPONENT_LIBRARY *adt_component_library();

    TRef *the_operator_component_library;
    DictionaryIterator operator_iterator();
    SB_OPERATOR_COMPONENT_LIBRARY *operator_component_library();

    TRef *the_component_dictionary;
    DictionaryIterator component_iterator();
    SB_COMPONENT_DICTIONARY *component_dictionary();

    TRef *the_keyword_library;
    SB_KEYWORD_LIBRARY *keyword_library();

    TRef *the_recognized_types;

    SB_COMPONENT_DICTIONARY *query(SB_COMPONENT *);

public:

    // methods for outos

    SB_LIBRARY (APL *);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    Type *getDirectType();

    //-----

    SB_LIBRARY(char *name,char *table);

    void component_list(ofstream& outstream);

    void keyword_list(ofstream& outstream);

    void type_list(ofstream& outstream);

    void operator_list(ofstream& outstream);

    void query(SB_COMPONENT *query_component,ofstream& outstream);

```

```
void keyword_query(ifstream& instream, ofstream& outstream);  
Boolean add(SB_COMPONENT *);  
SB_COMPONENT *query(char *component_name);  
void update_recognized_types(char *file);  
SB_RECOGNIZED_TYPES *recognized_types();  
void delete_component(SB_COMPONENT *the_component);  
};
```

```

class SB_ADT_COMPONENT_LIBRARY : public Object
{
private:

    TRef *the_adt_component_dictionary;

    SB_COMPONENT_DICTIONARY *adt_component_dictionary();

    TRef *the_main_library;

    Dictionary *main_library();

public:

    // methods for outos

    SB_ADT_COMPONENT_LIBRARY(APL *):

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    Type *getDirectType();

    //-----

    SB_ADT_COMPONENT_LIBRARY();

    Boolean add(SB_ADT_COMPONENT* );

    SB_COMPONENT_DICTIONARY *query(SB_ADT_COMPONENT *query_component);

    void list(ofstream& outstream);

    DictionaryIterator iterator();

    void delete_component(SB_ADT_COMPONENT *the_component);

};

```

```

class SB_COMPONENT_DICTIONARY : public Dictionary
{
public:
    //ontos methods

    SB_COMPONENT_DICTIONARY(APL *);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    Type *getDirectType();

    //-----

    SB_COMPONENT_DICTIONARY();

    Boolean add(SB_COMPONENT *):

    SB_COMPONENT *query(char *name):

    void printOn(ofstream& ostream);

    DictionaryIterator iterator();

};

```

```

class SB_OPERATOR_COMPONENT_LIBRARY : public Object
{
private:

    TRef *the_operator_component_dictionary;
    SB_COMPONENT_DICTIONARY *operator_component_dictionary();

    TRef *the_state_dictionary;
    Dictionary *state_dictionary();

    TRef *the_non_state_dictionary;
    Dictionary *non_state_dictionary();

public:

    // ontos methods

    SB_OPERATOR_COMPONENT_LIBRARY(APL *);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    Type *getDirectType();

    // -----

    SB_OPERATOR_COMPONENT_LIBRARY();

    Boolean add(SB_OPERATOR_COMPONENT *new_component);

    SB_COMPONENT_DICTIONARY *query(
        SB_OPERATOR_COMPONENT *query_component);

    void list(ofstream& outstream);

    DictionaryIterator iterator();

    void delete_component(SB_OPERATOR_COMPONENT *the_component);

};

```

```

class SB_KEYWORD_LIBRARY : public Dictionary
{
public:

    // ontos methods

    SB_KEYWORD_LIBRARY(APL *);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    // -----

    SB_KEYWORD_LIBRARY();

    void query(ifstream& instream, ofstream& outstream);

    Boolean add_component(SB_COMPONENT *new_component);

    void delete_component(SB_COMPONENT *the_component);

    DictionaryIterator iterator();

    void list(ofstream& outstream);

};

```

```

class SB_COMPONENT : public Object
{
private:

    char *the_component_name;

    TRef *the_keyword_dictionary;
    TRef *the_psdl_text;
    TRef *the_imp_spec_text;
    TRef *the_imp_body_text;
    TRef *the_informal_description;
    TRef *the_formal_description;
    TRef *the_norm_formal_description;
    TRef *the_recognized_type_usage;
    TRef *the_unrecognized_type_usage;

    TRef *the_generic_usage;

protected:

    SB_COMPONENT(APL *theAPL);

public:

    virtual Type *getDirectType()=0;

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    //-----

    SB_COMPONENT(char *id);

    virtual void printOn(ofstream& outstream)=0;

    Boolean add_keyword(char *keyword);

    char *component_name();

    SB_TEXT_OBJECT *psdl_text();

    SB_TEXT_OBJECT *imp_spec_text();

    SB_TEXT_OBJECT *imp_body_text();

    SB_TEXT_OBJECT *informal_description();

```

```

    SB_TEXT_OBJECT *formal_description();

    SB_TEXT_OBJECT *norm_formal_description();

    SB_TYPE_USAGE_DICTIONARY *generic_usage();

    void add_text(ifstream& psdl, ifstream& spec, ifstream& body);

    void insert_generics(SB_TYPE_USAGE_DICTIONARY * new_generic_usage);

    SB_TYPE_USAGE_DICTIONARY *recognized_type_usage();

    SB_TYPE_USAGE_DICTIONARY *unrecognized_type_usage();

    SB_KEYWORD_DICTIONARY *keyword_dictionary();

    int num_unrecognized_types();

    virtual int num_generic_types()=0;

    int total_types();

};

```

```

class SB_ADT_COMPONENT : public SB_COMPONENT
{
private:

    TRef *the_adt_usage;

    TRef *the_operator_specs;
    SB_ADT_OPERATOR_DICTIONARY *operator_specs();

public:

    // ontos methods

    SB_ADT_COMPONENT (APL *);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    Type *getDirectType();

    // -----

    SB_ADT_COMPONENT (char *id);

    void insert_adt_usage(SB_TYPE_USAGE_DICTIONARY * new_adt_usage);

    void insert_operators(SB_ADT_OPERATOR_DICTIONARY *new_operators);

    int num_adts();

    int num_adt_operators();

    int total_inputs();

    int total_outputs();

    virtual int num_generic_types();

    Boolean filter(SB_ADT_COMPONENT *library_component);

    DictionaryIterator adt_iterator();

    DictionaryIterator adt_operator_iterator();

    void printOn(ofstream& outstream);

    virtual Boolean process_type_info();

    SB_TYPE_USAGE_DICTIONARY *adt_usage();

```

};

```

class SB_OPERATOR : public SB_COMPONENT
{
private:
    TRef *the_input_attributes;

    TRef *the_output_attributes;

    TRef *the_exceptions;

    Boolean states_flag;

protected:
    // ontos methods

    SB_OPERATOR(APL * theAPL);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    virtual Type *getDirectType()=0;

    //-----

    SB_ID_DECL_DICTIONARY *input_attributes();

    SB_ID_DECL_DICTIONARY *output_attributes();

    SB_EXCEPTION_DICTIONARY *exceptions();

    SB_OPERATOR(char *id);

public:
    Boolean add_inputs(SB_ID_DECL_DICTIONARY *);

    Boolean add_outputs(SB_ID_DECL_DICTIONARY *);

    Boolean add_exceptions(SB_EXCEPTION_DICTIONARY *),

    int num_inputs();

    int num_outputs();

    virtual int num_generic_types();

    DictionaryIterator input_iterator();

```

```
DictionaryIterator output_iterator();  
DictionaryIterator exception_iterator();  
Boolean states();  
void set_states();  
virtual void printOn(ofstream& );  
};
```

```

class SB_ADT_OPERATOR : public SB_OPERATOR
{
public:

    // ontos methods

    SB_ADT_OPERATOR(APL * theAPL);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    virtual Type *getDirectType();

    //-----

    Boolean process_type_info(SB_ADT_COMPONENT *adt);

    SB_ADT_OPERATOR(char *);

};

```

```

class SB_OPERATOR_COMPONENT : public SB_OPERATOR
{
public:
    // ontos methods

    SB_OPERATOR_COMPONENT(APL * theAPL);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    virtual Type *getDirectType();

    //-----

    SB_OPERATOR_COMPONENT(char *);

    Boolean processTypeInfo();

    Boolean filter(SB_OPERATOR_COMPONENT *library_unit);
};

```

```

class SB_TEXT_OBJECT : public Object
{
private:
    char *the_text;

public:
    //ontos methods
    SB_TEXT_OBJECT(APL *);
    Type *getDirectType();
    virtual void Destroy(Boolean aborted=FALSE);
    virtual void deleteObject(Boolean deallocate=FALSE);
    virtual void putObject(Boolean deallocate=FALSE);
    //-----
    SB_TEXT_OBJECT();
    void append(ifstream& );
    void append(char *);
    void text(ofstream& );
    char *text();
};

```

```

class SB_KEYWORD_DICTIONARY : public Dictionary
{
public:
    //ontos methods

    SB_KEYWORD_DICTIONARY(APL *);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    Type *getDirectType();

    //-----

    SB_KEYWORD_DICTIONARY();

    Boolean add(char *);

    DictionaryIterator iterator();

    void printOn(ofstream& );

};

```

```

class SB_ADT_OPERATOR_DICTIONARY : public Dictionary
{
public:
    // onto's methods

    SB_ADT_OPERATOR_DICTIONARY(APL* );

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    Type* getDirectType();

    // -----

    SB_ADT_OPERATOR_DICTIONARY();

    void add(SB_ADT_OPERATOR *);

    void append(SB_ADT_OPERATOR_DICTIONARY *);

    int num();

    int total_inputs();

    int total_outputs();

    DictionaryIterator iterator();

    void printOn(ofstream& );
};

```

```

class SB_EXCEPTION_DICTIONARY : public Dictionary
{
public:
    //ontos methods

    SB_EXCEPTION_DICTIONARY(APL * );

    virtual void Destroy(Boolean aborted=FALSE);
    virtual void deleteObject(Boolean deallocate=FALSE);
    virtual void putObject(Boolean deallocate=FALSE);
    Type *getDirectType();
    //-----
    SB_EXCEPTION_DICTIONARY();

    Boolean add(char *):
    Boolean append(SB_EXCEPTION_DICTIONARY * );
    DictionaryIterator iterator();

    void printOn(ofstream& );
};

```

```

class SB_ID_DECL_DICTIONARY : public Object
{
private:

    TRef *the_dictionary_by_type;
    TRef *the_dictionary_by_id;
    TRef *the_id_declaration_list;

    Dictionary *dictionary_by_type();
    Dictionary *dictionary_by_id();
    List *id_declaration_list();

public:

    // ontos methods

    SB_ID_DECL_DICTIONARY(APL *);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    Type *getDirectType();

    //-----

    SB_ID_DECL_DICTIONARY();

    Boolean add_decl(char *id, SB_TYPE_NAME *type_name);

    Boolean add_decl(SB_ID_DECL *decl);

    void remove_decl(SB_ID_DECL *decl);

    Boolean append(SB_ID_DECL_DICTIONARY *);

    int num();

    SB_ID_DECL *query_id(char *query_name);

    DictionaryIterator id_iterator();

    DictionaryIterator type_iterator();

    ListIterator order_iterator();

    void printOn(ofstream& );

};

```



```

class SB_ID_DECL : public Object
{
private:
    char *the_id;

    TRef *the_type_name;

public:
    //ontos methods

    SB_ID_DECL(APL *);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    Type* getDirectType();

    //-----

    SB_ID_DECL(char *, SB_TYPE_NAME * );

    SB_TYPE_NAME *type_name();

    char *id();

    void printOn(ofstream& );

};

```

```

class SB_TYPE_NAME : public Object
{
private:

    char *the_id;
    char *the_base_type_id;

    int the_type_code;
    int the_base_type_code;

    TRef *the_id_decl_dictionary;

    SB_ID_DECL_DICTIONARY *id_decl_dictionary();

public:

    // ontos methods

    SB_TYPE_NAME(APL * );

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    Type* getDirectType();

    //-----

    SB_TYPE_NAME(char *.SB_ID_DECL_DICTIONARY *);

    virtual Boolean operator == (Entity& );

    virtual Boolean operator ≥ (Entity& );

    ListIterator declIterator();

    char *id();

    char *base_type_id();

    int type_code();

    int base_type_code();

    Boolean recognized();

    int num_decl();

    void printOn(ofstream& outstream);

```

};

•

•

•

•

```

class SB_TYPE_USAGE : public Object
{
private:
    char *the_type_id;
    TRef *the_type_name;
    int the_times_used;

public:
    //ontos methods

    SB_TYPE_USAGE(APL *theAPL);

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    //-----

    SB_TYPE_USAGE(char *new_type_id,SB_TYPE_NAME *new_type_name);

    char *type_id();

    void used();

    int times_used();

    Type *getDirectType();

    void printOn(ofstream& outstream);

    SB_TYPE_NAME *type_name();

    char *base_type_id();
};

```

```

class SB_TYPE_USAGE_DICTIONARY : public Object
{
private:

    TRef *the_dictionary_by_type_id;
    TRef *the_dictionary_by_times_used;
    TRef *the_dictionary_by_base_type;

    Dictionary *dictionary_by_times_used();
    Dictionary *dictionary_by_type_id();
    Dictionary *dictionary_by_base_type();

public:

    // ontos methods

    SB_TYPE_USAGE_DICTIONARY(APL *):

    virtual void Destroy(Boolean aborted=FALSE);

    virtual void deleteObject(Boolean deallocate=FALSE);

    virtual void putObject(Boolean deallocate=FALSE);

    Type *getDirectType();

    //-----

    SB_TYPE_USAGE_DICTIONARY();

    Boolean add_type(char *type_id,SB_TYPE_NAME *type_name);

    Boolean add_type(SB_TYPE_USAGE *type_usage);

    Boolean update(SB_TYPE_NAME *type_name);

    Boolean append(SB_TYPE_USAGE_DICTIONARY *);

    void remove_usage(SB_TYPE_USAGE *the_usage);
    int num();

    DictionaryIterator type_id_iterator();

    DictionaryIterator times_used_iterator();

    DictionaryIterator base_type_iterator();

    void printOn(ofstream& );

};

```

```

class SB_RECOGNIZED_TYPES : public Object
{
    // this object uses a data file which follows the following format
    // for its input

    /*-----

    case_sensitivity (0 for no, 1 for yes)

    unrecognized_id (ie UNRECOGNIZED for Ada. This id is
                      automatically assigned to all unrecognized id's)
    inheritance_id (ie base_type used for Ada)
    generic_type_id (ie generic_type used for Ada)
    generic_subprogram_id (ie generic_procedure used for Ada)
    abstract_data_type_id (ie adt used for Ada)
    array_id (ie array used for Ada)
    array_index_id (ie index used for Ada)
    array_element_id (ie element used for Ada)
    {recognized_type_id}* (all other type_id's known to this language
                          ie INTEGER, POSITIVE etc. used in Ada)
    ~ (used to separate the IDs from the rule map)

    rule matrix where 0 indicates no mapping 1 indicates yes
    each id entered above the ~ makes up the rows and columns of
    the matrix

    -----*/

private:
    TRef *the_name_dictionary;
    Dictionary *name_dictionary();

    TRef *the_row_array;
    Array *row_array();

    int array_size;

    Boolean case_sensitive;

    char *convert_to_upper(char *type_id);

public:
    // onto methods

    SB_RECOGNIZED_TYPES(APL *theAPL);

    Type *getDirectType();

    virtual void Destroy(Boolean aborted=FALSE);

```

```
virtual void deleteObject(Boolean deallocate=FALSE);  
virtual void putObject(Boolean deallocate=FALSE);  
  
//-----  
SB_RECOGNIZED_TYPES(char *file);  
int type_number(char *type_id);  
Boolean map(int map_in,int map_out);  
};
```

```
//  
// this file contains all of the external references to the  
// ontos type schemas  
//
```

```
extern Type *SB.LIBRARY_OType;  
extern Type *SB.ADT_COMPONENT_LIBRARY_OType;  
extern Type *SB.OPERATOR_COMPONENT_LIBRARY_OType;  
extern Type *SB.COMPONENT_OType;  
extern Type *SB.COMPONENT_DICTIONARY_OType;  
extern Type *SB.KEYWORD_DICTIONARY_OType;  
extern Type *SB.TEXT_OBJECT_OType;  
extern Type *SB.ADT_COMPONENT_OType;  
extern Type *SB.OPERATOR_COMPONENT_OType;  
extern Type *SB.ADT_OPERATOR_OType;  
extern Type *SB.ID_DECL_DICTIONARY_OType;  
extern Type *SB.ID_DECL_OType;  
extern Type *SB.TYPE_NAME_OType;  
extern Type *SB.ADT_OPERATOR_DICTIONARY_OType;  
extern Type *SB.EXCEPTION_DICTIONARY_OType;  
extern Type *SB.TYPE_USAGE_OType;  
extern Type *SB.TYPE_USAGE_DICTIONARY_OType;  
extern Type *SB.RECOGNIZED_TYPES_OType;
```

```
extern SB.LIBRARY *SB.MAIN_LIBRARY;
```

```
# define DEFAULT_NAME_SIZE 21
```

```
#define SB.UNRECOGNIZED_TYPE 1
```

```
#define SB.BASE_TYPE 2
```

```
#define SB.GENERIC_TYPE 3
```

```
#define SB.GENERIC_SUBPROGRAM 4
```

```
#define SB.GENERIC_VALUE 5
```

```
#define SB.ABSTRACT_DATA_TYPE 6
```

```
#define SB.ARRAY 7
```

```
#define SB.ARRAY_INDEX 8
```

```
#define SB.ARRAY_ELEMENT 9
```

```

#include "sball.hxx"
// -----
// these defines are to declare the command line
// arguments
// -----
extern "C--"
{
    int system(char *);
}

#define KWL "kwl" // caps_softbase kwl language out_file
#define KWL_N 1 // keyword list

#define KWQ "kwq" // caps_softbase kwq language in_file out_file
#define KWQ_N 2 // keyword query (2)

#define OL "ol" // caps_softbase ol language out_file
#define OL_N 3 // operator list (3)

#define TL "tl" // caps_softbase tl language out_file
#define TL_N 4 // type list (4)

#define CQ "cq" // caps_softbase cq language psdl_file out_file
#define CQ_N 5 // component query (5)

#define CA "ca" // caps_softbase ca language
// psdl_file spec_in body_in
#define CA_N 6 // component add (6)

#define CU "cu" // caps_softbase cu language
// psdl_file spec_in body_in
#define CU_N 7 // component update (7)

#define CD "cd" // caps_softbase cd language component_name
#define CD_N 8 // component delete (8)

#define CL "cl" // caps_softbase cl language out_fil
#define CL_N 9 // component list (9)

#define CGM "cgm" // caps_softbase cg language
// psdl component map_out
#define CGM_N 10 // component generate map (10)

#define ML "ml" // caps_softbase ml language generator table
#define ML_N 11 // make new library

#define CV "cv" // caps_softbase cv language
// component_name psdl
// ada_spec ada_body
#define CV_N 12 // component view

#define DL "dl" // caps_softbase dl language

```

```

#define DLN 13 // delete language library

#define CDIAG "cdiag" // caps_softbase cdiag language
// component_name outfile
#define CDIAG_N 14 // print component diagnostics

#define LOGICAL_DB_NAME "caps_softbase_LogDB"

#define LIBRARY_PREFIX "SB_"

#define LIBRARY_SUFFIX "_LIBRARY"

#define TEMP_ENVIRONMENT "TEMP"
#define DEFAULT_TEMP "./"
#define NORMALIZE "caps_softbase_normalize "

#ifdef __TURBOC__
int line_number;
FILE *yyin;
#else
extern "C--"
{
    extern int line_number; // used to report the line number of an error
    extern FILE *yyin; // lex input stream
}
#endif

Type *SB_LIBRARY_OType;
Type *SB_ADT_COMPONENT_LIBRARY_OType;
Type *SB_OPERATOR_COMPONENT_LIBRARY_OType;
Type *SB_COMPONENT_OType;
Type *SB_COMPONENT_DICTIONARY_OType;

Type *SB_KEYWORD_DICTIONARY_OType;
Type *SB_TEXT_OBJECT_OType;

Type *SB_ADT_COMPONENT_OType;

Type *SB_OPERATOR_COMPONENT_OType;
Type *SB_ADT_OPERATOR_OType;

Type *SB_ID_DECL_DICTIONARY_OType;
Type *SB_ID_DECL_OType;
Type *SB_TYPE_NAME_OType;
Type *SB_ADT_OPERATOR_DICTIONARY_OType;
Type *SB_EXCEPTION_DICTIONARY_OType;
Type *SB_TYPE_USAGE_OType;
Type *SB_TYPE_USAGE_DICTIONARY_OType;
Type *SB_RECOGNIZED_TYPES_OType;

```

```

SB_LIBRARY *SB_MAIN_LIBRARY;

SB_COMPONENT *YYPARSE_component; // used by yyparse to pass the components

extern yyparse();

main(int argc, char *argv[])
{
    int exit_flag;

    void update_db_types();

    int parse_command(int argc, char *argv[]);

    Boolean get_language_library(int argc, char *argv[]);

    if(OC_open(LOGICAL_DB_NAME)≠TRUE)
    {
        cout << "THE LOGICAL SOFTBASE " << LOGICAL_DB_NAME << "OPEN FAILED\n";
        exit(1);
    };

    int the_operation=parse_command(argc,argv);
    if(the_operation≠ML_N && the_operation≠0)
    {
        if(!get_language_library(argc,argv))
        {
            exit(1);
        };
    };

    update_db_types();

    switch(the_operation)
    {
        case 0:
        {
            cout << "AN INVALID OPERATION WAS GIVEN TO SB\n";
            break;
        };
        case KWL_N:
        {
            if(argc==4)
            {
                ofstream outfile(argv[3].ios::noreplace);
                if(outfile)
                {
                    // outfile was opened successfully
                    SB_MAIN_LIBRARY→keyword_list(outfile);
                }
            }
        }
    }
}

```

```

        exit_flag=0;
    }
    else
    {
        cout << "UNABLE TO OPEN OUTPUT FILE\n";
        exit_flag=1;
    };
}
else
{
    cout << "INCORRECT NUMBER OF ARGUMENTS\n";
    exit_flag=1;
};

break;
};
case KWQ_N:
{
    if(argc==5)
    {
        ifstream infile(argv[3].ios::nocreate);
        if(infile)
        {
            // query file open successful
            ofstream outfile(argv[4].ios::noreplace);
            if(outfile)
            {
                // output file opened successful
                SB_MAIN_LIBRARY—keyword_query(infile,outfile);
                exit_flag=0;
            }
        }
        else
        {
            cout << "UNABLE TO OPEN OUTPUT FILE\n";
            exit_flag=1;
        }
    }
    else
    {
        cout << "UNABLE TO OPEN INPUT FILE\n";
        exit_flag=1;
    };
}
else
{
    cout << "INCORRECT NUMBER OF ARGUMENTS\n";
    exit_flag=1;
};

break;
};

```

```

case OL_N:
{
    if(argc==4)
    {
        ofstream outfile(argv[3],ios::noreplace);
        if(outfile)
        {
            // output file opened successfully
            SB_MAIN_LIBRARY→operator_list(outfile);
            exit_flag=0;
        }
        else
        {
            cout << "UNABLE TO OPEN INPUT FILE\n";
            exit_flag=1;
        }
    }
    else
    {
        cout << "INCORRECT NUMBER OF ARGUMENTS\n";
        exit_flag=1;
    }
    break;
};

case CL_N:
{
    if(argc==4)
    {
        ofstream outfile(argv[3],ios::noreplace);
        if(outfile)
        {
            // output file opened successfully
            SB_MAIN_LIBRARY→component_list(outfile);
            exit_flag=0;
        }
        else
        {
            cout << "UNABLE TO OPEN INPUT FILE\n";
            exit_flag=1;
        }
    }
    else
    {
        cout << "INCORRECT NUMBER OF ARGUMENTS\n";
        exit_flag=1;
    }
    break;
};

case TL_N:
{

```

```

if(argc==4)
{
    ofstream outfile(argv[3],ios::noreplace);
    if(outfile)
    {
        // output file opened successfully
        SB_MAIN_LIBRARY→type_list(outfile);
        exit_flag=0;
    }
    else
    {
        cout << "UNABLE TO OPEN INPUT FILE\n";
        exit_flag=1;
    };
}
else
{
    cout << "INCORRECT NUMBER OF ARGUMENTS\n";
    exit_flag=1;
};
break;
};
case CQ_N:
{
    if(argc==5)
    {
        yyin=fopen(argv[3],"r");
        if(yyin≠NULL)
        {
            // psdl file opened succesfully
            ofstream outfile(argv[4],ios::noreplace);
            if(outfile)
            {
                // outfile open so do parse psdl_file
                // nest the transaction so syntax error transaction
                // can be aborted

                get_language_library(argc,argv);

                if(yparse()==0)
                {
                    // file parsed successfully result is
                    // in YYPARSE_component
                    if(YYPARSE_component→getDirectType()==SB_OPERATOR_COMPONENT_OTy
                    {
                        ((SB_OPERATOR_COMPONENT *)YYPARSE_component)→
                        process_type_info();
                    }
                    else
                    {
                        ((SB_ADT_COMPONENT *)YYPARSE_component)→

```

```

        process_type_info();
    };

    SB_MAIN_LIBRARY→query(YYPARSE_component,outfile);
    //YYPARSE_component->Destroy(FALSE);
    exit_flag=0;
}
else
{
    cout << "THERE WAS AN ERROR DURING PARSING ";
    cout << argv[3] << "\n";
    exit_flag=1;
};

}
else
{
    cout << "UNABLE TO OPEN OUTPUT FILE\n";
    exit_flag=1;
};

}
else
{
    cout << "UNABLE TO OPEN INPUT FILE\n";
    exit_flag=1;
};

}
else
{
    cout << "INCORRECT NUMBER OF ARGUMENTS\n";
    exit_flag=1;
};
break;
};

case CA_N:
{
    if(argc==6)
    {
        yyin=fopen(argv[3],"r");
        if(yyin≠NULL)
        {
            // psdl file opened succesfully
            ifstream spec_in(argv[4],ios::nocreate);
            if(spec_in)
            {
                // spec_in file open so do parse psdl_file
                ifstream body_in(argv[5],ios::noreplace);
                if(body_in)
                {
                    // all files successfully open so start transaction

```

```

// nest the transactions so syntax errors can
// be aborted

get_language_library(argc,argv);
update_db_types();

if(yyparse()==0)
{
    // file parsed successfully result is
    // in YYPARSE_component so add the psdl to it
    // and the spec and body
    ifstream psdl_in(argv[3],ios::nocreate);
    YYPARSE_component→add_text(psdl_in,spec_in,body_in);

    // now normalize the formal description if there is
    // one
    if(strlen(YYPARSE_component→
        formal_description()→text())>0)
    {
        char *temp_dir = getenv(TEMP_ENVIRONMENT);
        if (temp_dir == NULL)
        {
            temp_dir = new char[strlen(DEFAULT_TEMP) + 1];
            strcpy(temp_dir, DEFAULT_TEMP);
        };
        char *temp_file = tempnam(temp_dir, "obj");
        char *error_file = tempnam(temp_dir, "rm");

        ostringstream command_buffer;
        ostringstream remove_buffer;
        ostringstream obj_file_buffer;
        ostringstream norm_file_buffer;

        obj_file_buffer << temp_file << ".obj" << ends;
        norm_file_buffer << temp_file << ".obj.norm" << ends;

        char *obj_file=obj_file_buffer.str();
        char *norm_file=norm_file_buffer.str();

        command_buffer << NORMALIZE;
        command_buffer << obj_file << " ";
        command_buffer << error_file << ends;

        remove_buffer << "rm " << temp_file << ".*";
        remove_buffer << ends;
        ofstream formal_desc(obj_file);
        YY-
        PARSE_component→formal_description()→text(formal_desc);
        formal_desc.close();
        char *command=command_buffer.str();
        int status_flag=system(command);
        if(status_flag==0)

```

```

{
    // successful so get the norm file
    ifstream norm_in(norm_file);
    if(norm_in)
    {
        // file was there so append it
        YYPARSE_component→
            norm_formal_description()→
                append(norm_in);

        if(YYPARSE_component→getDirectType()==
            SB_OPERATOR_COMPONENT_OType)
        {
            ((SB_OPERATOR_COMPONENT
*)YYPARSE_component)→
                process_type_info();
        }
        else
        {
            ((SB_ADT_COMPONENT
*)YYPARSE_component)→
                process_type_info();
        }
    };

    if(SB_MAIN_LIBRARY→
        query(YYPARSE_component→component_name())==NULL)
    {
        // component not already in library so store it
        OC_transactionStart();
        YYPARSE_component→putObject();

        Boolean add_status=FALSE;
        add_status=SB_MAIN_LIBRARY→add(YYPARSE_component
        if(add_status==TRUE)
        {
            OC_transactionCommit();
            exit_flag=0;
        }
        else
        {
            exit_flag=1;
            cout << "UNABLE TO ADD COMPONENT ";
            cout <<

            YYPARSE_component→component_name();

            cout << "TO MAIN LIBRARY\n";
            OC_transactionAbort();
        }
    }
    else
    {
        cout << "COMPONENT " <<

```

```

YYPARSE_component→component_name();
cout << " IS ALREADY IN MAIN LIBRARY " <<

endl;

        cout << "UNABLE TO ADD IT AGAIN \n";
        exit_flag=1;
    };
}
else
{
    cout << "ERROR NORMALIZING AXIOMS\n";
    exit_flag=1;
};
}
else
{
    cout << "ERROR NORMALIZING AXIOMS\n";
    ifstream error_stream(error_file);
    if(error_stream)
    {
        char *the_line=new char[256];
        while(!error_stream.eof())
        {
            error_stream.getline(the_line,255);
            cout << the_line << endl;
            error_stream >> ws;
        };
    }
    else
    {
        cout << "COULD NOT OPEN NORMALIZE";
        cout << " ERROR FILE\n";
    };
    exit_flag=1;
};
}
else
{
    if(YYPARSE_component→getDirectType()==
        SB_OPERATOR_COMPONENT_OType)
    {
        ((SB_OPERATOR_COMPONENT
*)YYPARSE_component)→
        process_type_info();
    }
    else
    {
        ((SB_ADT_COMPONENT *)YYPARSE_component)→
        process_type_info();
    };
}

if(SB_MAIN_LIBRARY→

```

```

        query(YYPARSE_component→component_name())==NULL)
    {
        // component not already in library so store it
        OC_transactionStart();
        YYSYMBOL_component→putObject();

        Boolean add_status=FALSE;
        add_status=SB_MAIN_LIBRARY→add(YYPARSE_component);
        if(add_status==TRUE)
        {
            OC_transactionCommit();
            exit_flag=0;
        }
        else
        {
            exit_flag=1;
            cout << "UNABLE TO ADD COMPONENT ";
            cout <<
YYPARSE_component→component_name();
            cout << "TO MAIN LIBRARY\n";
            OC_transactionAbort();
        }
    }
    else
    {
        cout << "COMPONENT " <<
            YYSYMBOL_component→component_name();
        cout << " IS ALREADY IN MAIN LIBRARY "<<endl;

        cout << "UNABLE TO ADD IT AGAIN \n";
        exit_flag=1;
    };
};

}
else
{
    cout << "THERE WAS AN ERROR DURING PARSING ";
    cout << argv[3] << "\n";
    exit_flag=1;
};

}
else
{
    cout << "UNABLE TO OPEN THE IMPLEMENTATION BODY FILE\n";
    exit_flag=1;
};

}
else
{
    cout << "UNABLE TO THE IMPLEMENTATION SPEC FILE\n";
    exit_flag=1;
}
}

```

```

        };
    }
    else
    {
        cout << "UNABLE TO OPEN PSDL FILE\n";
        exit_flag=1;
    };
}
else
{
    cout << "INCORRECT NUMBER OF ARGUMENTS\n";
    exit_flag=1;
};
break;
};

case ML_N:
{
    if(argc==4)
    {
        char *language_name=argv[2];
        char *library_name=new char[strlen(language_name)+
                                     strlen(LIBRARY_PREFIX)+
                                     strlen(LIBRARY_SUFFIX)+1];

        strcpy(library_name,LIBRARY_PREFIX);
        strcat(library_name,language_name);
        strcat(library_name,LIBRARY_SUFFIX);
        OC_transactionStart();
        SB_MAIN_LIBRARY=new SB_LIBRARY(library_name,argv[3]);
        SB_MAIN_LIBRARY->putObject();
        OC_transactionCommit();
        exit_flag=0;
    }
    else
    {
        cout << "INCORRECT NUMBER OF ARGUMENTS\n";
        exit_flag=1;
    };
    break;
};

case DL_N:
{
    if(argc==3)
    {
        OC_transactionStart();
        SB_MAIN_LIBRARY->deleteObject(TRUE);
        OC_transactionCommit();
    }
};

```

```

        exit_flag=0;
    }
    else
    {
        cout << "INCORRECT NUMBER OF ARGUMENTS\n";
        exit_flag=1;
    };
    break;

};

case CV_N:
{
    if(argc==7)
    {

        ofstream psdl_out(argv[4],ios::noreplace);
        if(psdl_out)
        {
            ofstream spec_out(argv[5],ios::noreplace);
            if(spec_out)
            {
                ofstream body_out(argv[6],ios::noreplace);
                if(body_out)
                {
                    get_language_library(argc,argv);
                    update_db_types();
                    SB_COMPONENT
*the_component=SB_MAIN_LIBRARY—query(argv[3]);
                    if(the_component≠NULL)
                    {
                        // component found so dump all streams
                        (the_component→psdl_text())→text(psdl_out);
                        (the_component→imp_spec_text())→text(spec_out);
                        (the_component→imp_body_text())→text(body_out);
                        psdl_out.close();
                        spec_out.close();
                        body_out.close();
                        exit_flag=0;
                    }
                    else
                    {
                        cout << "COMPONENT " << argv[3] << " NOT FOUND\n";
                        exit_flag=1;
                    };
                }
            }
        }
        else
        {
            cout << "UNABLE TO OPEN THE IMPLEMENTATION BODY FILE\n";
            exit_flag=1;
        };
    }
};

```

```

        }
    else
    {
        cout << "UNABLE TO OPEN THE IMPLEMENTATION SPEC FILE\n";
        exit_flag=1;
    };
}
else
{
    cout << "UNABLE TO OPEN PSDL FILE\n";
    exit_flag=1;
};
}
else
{
    cout << "INCORRECT NUMBER OF ARGUMENTS\n";
    exit_flag=1;
};
break;
};

case CD_N:
{
    if(argc==1)
    {

        SB_COMPONENT *the_component=SB_MAIN_LIBRARY->query(argv[3]);
        if(the_component!=NULL)
        {
            // component found so output its sources to backup
            // this is where the SCCS code goes for the_component
            //(the_component->psdl_text())->text(psdl_out);
            //(the_component->imp_spec_text())->text(spec_out);
            //(the_component->imp_body_text())->text(body_out);
            //psdl_out.close();
            //spec_out.close();
            //body_out.close();
            OC_transactionStart();
            // now delete the component from the library
            SB_MAIN_LIBRARY->delete_component(the_component);
            exit_flag=0;
            OC_transactionCommit();
        }
    else
    {
        cout << "COMPONENT " << argv[3] << " NOT FOUND\n";
        exit_flag=1;
    };
}
else
{

```

```

        cout << "INCORRECT NUMBER OF ARGUMENTS\n";
        exit_flag=1;
    };
    break;
};

case CDIAG_N:
{
    if(argc==5)
    {

        ofstream outfile(argv[4],ios::noreplace);
        if(outfile)
        {
            // outfile was opened successfully
            SB_COMPONENT *the_component=SB_MAIN_LIBRARY->query(argv[3]);
            if(the_component!=NULL)
            {
                // component found so outputs its diagnostics

                the_component->printOn(outfile);
            }
            else
            {
                cout << "COMPONENT " << argv[3] << " NOT FOUND\n";
                exit_flag=1;
            };
        }
        else
        {
            cout << "UNABLE TO OPEN OUTPUT FILE\n";
            exit_flag=1;
        };
    }

    else
    {
        cout << "INCORRECT NUMBER OF ARGUMENTS\n";
        exit_flag=1;
    };
    break;
};

};

OC_close();

exit(exit_flag);
};

int yyerror(char *s)

```

```

{
    cout << "\n" << s << " on line number " << line_number;
    cout << "\n";
    return(0);
};

void update_db_types()
{
    SB_LIBRARY_OType=(Type *) OC_lookup("SB_LIBRARY");
    SB_COMPONENT_OType=(Type *) OC_lookup("SB_COMPONENT");
    SB_ADT_COMPONENT_OType=(Type *) OC_lookup("SB_ADT_COMPONENT");
    SB_ID_DECL_DICTIONARY_OType=(Type *) OC_lookup("SB_ID_DECL_DICTIONARY");
    SB_ID_DECL_OType=(Type *) OC_lookup("SB_ID_DECL");
    SB_TYPE_NAME_OType=(Type *) OC_lookup("SB_TYPE_NAME");
    SB_ADT_OPERATOR_DICTIONARY_OType=(Type *)
OC_lookup("SB_ADT_OPERATOR_DICTIONARY");
    SB_ADT_OPERATOR_OType=(Type *) OC_lookup("SB_ADT_OPERATOR");
    SB_OPERATOR_COMPONENT_OType=(Type *) OC_lookup("SB_OPERATOR_COMPONENT");
    SB_EXCEPTION_DICTIONARY_OType=(Type *) OC_lookup("SB_EXCEPTION_DICTIONARY");
    SB_TEXT_OBJECT_OType=(Type *) OC_lookup("SB_TEXT_OBJECT");
    SB_KEYWORD_DICTIONARY_OType=(Type *) OC_lookup("SB_KEYWORD_DICTIONARY");
    SB_COMPONENT_DICTIONARY_OType=(Type *)
OC_lookup("SB_COMPONENT_DICTIONARY");
    SB_ADT_COMPONENT_LIBRARY_OType=(Type *)
OC_lookup("SB_ADT_COMPONENT_LIBRARY");
    SB_OPERATOR_COMPONENT_LIBRARY_OType=(Type *)
OC_lookup("SB_OPERATOR_COMPONENT_LIBRARY");
    SB_TYPE_USAGE_OType=(Type *)OC_lookup("SB_TYPE_USAGE");

    SB_TYPE_USAGE_DICTIONARY_OType=(Type
*)OC_lookup("SB_TYPE_USAGE_DICTIONARY");
    SB_RECOGNIZED_TYPES_OType=(Type *)OC_lookup("SB_RECOGNIZED_TYPES");

};

int parse_command(int argc, char *argv[])
{
    int return_value=0;

    if(argc ≥ 2)
    {
        if(strcmp(argv[1].KWL)==0)
        {
            return_value=KWL_N;
        }
        else if(strcmp(argv[1].KWQ)==0)
        {
            return_value=KWQ_N;
        }
    }
}

```

```

else if(strcmp(argv[1],OL)==0)
{
    return_value=OL_N;
}
else if(strcmp(argv[1],TL)==0)
{
    return_value=TL_N;
}
else if(strcmp(argv[1],CQ)==0)
{
    return_value=CQ_N;
}
else if(strcmp(argv[1],CA)==0)
{
    return_value=CA_N;
}
else if(strcmp(argv[1],CU)==0)
{
    return_value=CU_N;
}
else if(strcmp(argv[1],CD)==0)
{
    return_value=CD_N;
}
else if(strcmp(argv[1],CL)==0)
{
    return_value=CL_N;
}
else if(strcmp(argv[1],CGM)==0)
{
    return_value=CGM_N;
}
else if(strcmp(argv[1],ML)==0)
{
    return_value=ML_N;
}
else if(strcmp(argv[1],CV)==0)
{
    return_value=CV_N;
}
else if(strcmp(argv[1],DL)==0)
{
    return_value=DL_N;
}
else if(strcmp(argv[1],CDIAG)==0)
{
    return_value=CDIAG_N;
}
}
return return_value;
};

```

```

Boolean get_language_library(int argc, char *argv[])
{
    Boolean return_flag=FALSE;

    if(argc > 0)
    {
        char *language_name=argv[2];

        char *library_name=new char[strlen(language_name)+
                                     strlen(LIBRARY_PREFIX)+
                                     strlen(LIBRARY_SUFFIX)+1];

        strcpy(library_name, LIBRARY_PREFIX);
        strcat(library_name, language_name);
        strcat(library_name, LIBRARY_SUFFIX);

        // ASSIGN THE GLOBAL VARIABLE SB_MAIN_LIBRARY THE VALUE OF THE
        LIBRARY

        SB_MAIN_LIBRARY=(SB_LIBRARY *) OC_lookup(library_name);

        if(SB_MAIN_LIBRARY==NULL)
        {
            cout << "LIBRARY FOR LANGUAGE ";
            cout << language_name << " NOT FOUND\n";
            return_flag=FALSE;
        }
        else
        {
            return_flag=TRUE;
        };
    }
    else
    {
        cout << "INCORRECT NUMBER OF ARGUMENTS\n";
    };

    return return_flag;
};

```

```

# include "sball.hxx"

# include "sbextern.h"

SB_ADT_COMPONENT::SB_ADT_COMPONENT(APL *theAPL):
SB_COMPONENT(theAPL)
{
};

SB_ADT_COMPONENT::SB_ADT_COMPONENT (char *id) : SB_COMPONENT(id)
{
    SB_TYPE_USAGE_DICTIONARY *new_adt_usage=new
SB_TYPE_USAGE_DICTIONARY();
    the_adt_usage=new_adt_usage→findTRef();

    SB_ADT_OPERATOR_DICTIONARY *new_operator_specs=new
SB_ADT_OPERATOR_DICTIONARY();

    the_operator_specs=new_operator_specs→findTRef();
};

void SB_ADT_COMPONENT::insert_adt_usage(SB_TYPE_USAGE_DICTIONARY*
new_adt_usage)
{
    adt_usage()→append(new_adt_usage);
};

void SB_ADT_COMPONENT::insert_operators(SB_ADT_OPERATOR_DICTIONARY*
new_operators)
{
    operator_specs()→append(new_operators);
};

void SB_ADT_COMPONENT::Destroy(Boolean aborted)
{
    adt_usage()→Destroy(aborted);

    operator_specs()→Destroy(aborted);

    delete the_adt_usage;
    delete the_operator_specs;

    SB_COMPONENT::Destroy(aborted);
};

void SB_ADT_COMPONENT::deleteObject(Boolean deallocate)
{

```

```

    adt_usage()→deleteObject(FALSE);

    operator_specs()→deleteObject(FALSE);

    SB_COMPONENT::deleteObject(deallocate);

};

void SB_ADT_COMPONENT::putObject(Boolean deallocate)
{
    adt_usage()→putObject(deallocate);

    operator_specs()→putObject(deallocate);

    SB_COMPONENT::putObject(deallocate);

};

SB_TYPE_USAGE_DICTIONARY *SB_ADT_COMPONENT::adt_usage()
{
    return (SB_TYPE_USAGE_DICTIONARY *) (the_adt_usage→Binding());
};

SB_ADT_OPERATOR_DICTIONARY *SB_ADT_COMPONENT::operator_specs()
{
    return (SB_ADT_OPERATOR_DICTIONARY *) (the_operator_specs→Binding());
};

Type *SB_ADT_COMPONENT::getDirectType()
{
    return SB_ADT_COMPONENT_OType;
};

void SB_ADT_COMPONENT::printOn(ofstream& outstream)
{
    outstream << "\nOUTPUTING THE CONTENTS OF DATA TYPE ";
    outstream << this→component_name() << ":\n\n";
    outstream << "num unrecognized types " << num_unrecognized_types() << "\n";
    outstream << "GENERIC SPECS\n";
    SB_COMPONENT::generic_usage()→printOn(outstream);
    outstream << "\nTYPE SPECS\n";
    adt_usage()→printOn(outstream);

    outstream << "\nOPERATOR SPECS\n";
    operator_specs()→printOn(outstream);

    outstream << "\n\n";
};

```

```

int SB_ADT_COMPONENT::num_adt_operators()
{
    return operator_specs()→num();
};

int SB_ADT_COMPONENT::total_inputs()
{
    return operator_specs()→total_inputs();
};

int SB_ADT_COMPONENT::num_adts()
{
    return adt_usage()→num();
};

int SB_ADT_COMPONENT::total_outputs()
{
    return operator_specs()→total_outputs();
};

DictionaryIterator SB_ADT_COMPONENT::adt_operator_iterator()
{
    return operator_specs()→iterator();
};

DictionaryIterator SB_ADT_COMPONENT::adt_iterator()
{
    return adt_usage()→type_id_iterator();
};

Boolean SB_ADT_COMPONENT::process_type_info()
{
    // tell each operator to update its type usage lists
    // they inturn update the adt lists

    DictionaryIterator next_operator=adt_operator_iterator();
    while(next_operator.moreData())
    {
        ((SB_ADT_OPERATOR *)(Entity *)next_operator())→
            process_type_info(this);
    };

    return TRUE;
};

int SB_ADT_COMPONENT::num_generic_types()
{
    int the_num=0;

```

```

DictionaryIterator next_operator=adt_operator_iterator();
while(next_operator.moreData())
{
    the_num=the_num+
        ((SB_ADT_OPERATOR *)(Entity *)next_operator())→
        num_generic_types();
};

// now get an iterator for the adt generics list

DictionaryIterator next_id=SB_COMPONENT::generic_usage()→
    type_id_iterator();

while(next_id.moreData())
{
    SB_TYPE_USAGE *the_usage=(SB_TYPE_USAGE*)(Entity *)next_id();
    SB_TYPE_NAME *the_type_name=the_usage→type_name();
    if(the_type_name→type_code()==SB_GENERIC_TYPE)
    {
        the_num=the_num++;
    };
};

return the_num;
};

Boolean SB_ADT_COMPONENT::filter(SB_ADT_COMPONENT *library_unit)
{
    // apply additional filter operations to the library unit
    // to see if the component can be rejected. True means
    // that it may still be a match. False indicates no match

    return TRUE;
};

```

```

# include "sball.hxx"

# include "sbextern.h"

SB_ADT_COMPONENT_LIBRARY::SB_ADT_COMPONENT_LIBRARY(APL *theAPL) :
Object(theAPL)
{
};

SB_ADT_COMPONENT_LIBRARY::SB_ADT_COMPONENT_LIBRARY() : Object()
{

    SB_COMPONENT_DICTIONARY *new_adt_component_dictionary=
        new SB_COMPONENT_DICTIONARY();

    the_adt_component_dictionary=
        new_adt_component_dictionary—findTRef();

    Dictionary *new_main_library=new Dictionary(OC_integer,
                                                OC_dictionary,
                                                TRUE,
                                                FALSE);

    the_main_library=new_main_library—findTRef();

};

void SB_ADT_COMPONENT_LIBRARY::Destroy(Boolean aborted)
{
    adt_component_dictionary()—Destroy(aborted);

    // now must iterate through the multi-attribute tree of
    // dictionaries to destroy each one of them

    Dictionary *by_num_adts;
    Dictionary *by_num_operators;
    Dictionary *by_num_total_inputs;
    Dictionary *by_num_generics;
    Dictionary *by_num_total_outputs;
    Dictionary *leaf_dictionary;

    by_num_adts=main_library();

    DictionaryIterator next_by_num_adt(by_num_adts);

    while(next_by_num_adt.moreData())
    {
        by_num_operators=(Dictionary *) (Entity *) next_by_num_adt();

        // components must have at least as many operators as the query
    }
}

```

```

DictionaryIterator next_by_num_operators(by_num_operators);

while(next_by_num_operators.moreData())
{
    by_num_generics=(Dictionary *)(Entity *)
        next_by_num_operators();

    DictionaryIterator next_by_num_generics(by_num_generics);

    while(next_by_num_generics.moreData())
    {

        by_num_totalOutputs=(Dictionary *)(Entity *)
            next_by_num_generics();

        DictionaryIterator
            next_by_num_total_outputs(by_num_totalOutputs);

        while(next_by_num_total_outputs.moreData())
        {

            by_num_totalInputs=(Dictionary *)(Entity *)
                next_by_num_total_outputs();

            DictionaryIterator
                next_by_num_total_inputs(by_num_totalInputs);

            while(next_by_num_total_inputs.moreData())
            {
                leaf_dictionary=(Dictionary *)(Entity *)
                    next_by_num_total_inputs();

                leaf_dictionary—Destroy(aborted);

            };
            by_num_totalInputs—Destroy(aborted);
        };
        by_num_totalOutputs—Destroy(aborted);
    };
    by_num_generics—Destroy(aborted);
};
by_num_adts—Destroy(aborted);
};

delete the_adt_component_dictionary;
delete the_main_library;

Object::Destroy(aborted);

};

```

```

void SB_ADT_COMPONENT_LIBRARY::deleteObject(Boolean deallocate)
{
    adt_component_dictionary()—deleteObject(deallocate);

    // now must iterate through the multi-attribute tree of
    // dictionaries to destroy each one of them

    Dictionary *by_num_adts;
    Dictionary *by_num_operators;
    Dictionary *by_num_total_inputs;
    Dictionary *by_num_generics;
    Dictionary *by_num_total_outputs;
    Dictionary *leaf_dictionary;

    by_num_adts=main_library();

    DictionaryIterator next_by_num_adt(by_num_adts);

    while(next_by_num_adt.moreData())
    {
        by_num_operators=(Dictionary *) (Entity *) next_by_num_adt();

        // components must have at least as many operators as the query

        DictionaryIterator next_by_num_operators(by_num_operators);

        while(next_by_num_operators.moreData())
        {
            by_num_generics=(Dictionary *) (Entity *)
                next_by_num_operators();

            DictionaryIterator next_by_num_generics(by_num_generics);

            while(next_by_num_generics.moreData())
            {
                by_num_total_outputs=(Dictionary *) (Entity *)
                    next_by_num_generics();

                DictionaryIterator
                    next_by_num_total_outputs(by_num_total_outputs);

                while(next_by_num_total_outputs.moreData())
                {
                    by_num_total_inputs=(Dictionary *) (Entity *)
                        next_by_num_total_outputs();

                    DictionaryIterator
                        next_by_num_total_inputs(by_num_total_inputs);

                    while(next_by_num_total_inputs.moreData())

```

```

        {
            leaf_dictionary=(Dictionary *)(Entity *)
                next_by_num_total_inputs();

            leaf_dictionary→deleteObject(FALSE);

        };
        by_num_total_inputs→deleteObject(FALSE);
    };
    by_num_total_outputs→deleteObject(FALSE);

};
    by_num_generics→deleteObject(FALSE);
};
    by_num_adts→deleteObject(FALSE);
};

Object::deleteObject(deallocate);

};

void SB_ADT_COMPONENT_LIBRARY::putObject(Boolean deallocate)
{
    adt_component_dictionary()→Dictionary::putObject(deallocate);
    main_library()→putObject(deallocate);

    Object::putObject(deallocate);

};

Type *SB_ADT_COMPONENT_LIBRARY::getDirectType()
{
    return SB_ADT_COMPONENT_LIBRARY_OType;
};

SB_COMPONENT_DICTIONARY
*SB_ADT_COMPONENT_LIBRARY::adt_component_dictionary()
{
    return (SB_COMPONENT_DICTIONARY *)(Entity *)
        the_adt_component_dictionary→Binding();
};

Dictionary *SB_ADT_COMPONENT_LIBRARY::main_library()
{
    return (Dictionary *)(Entity *)the_main_library→Binding();
};

Boolean SB_ADT_COMPONENT_LIBRARY::add(SB_ADT_COMPONENT *new_component)
{
    Boolean return_flag=TRUE;

```

```

Dictionary *by_num_adts;
Dictionary *by_num_operators;
Dictionary *by_num_total_inputs;
Dictionary *by_num_generics;
Dictionary *by_num_total_outputs;
Dictionary *leaf_dictionary;

adt_component_dictionary()→add(new_component);
adt_component_dictionary()→Dictionary::putObject();

// insert into the component dictionary was successfull
// so insert it into the library

// get the dictionary for the number of adt's

by_num_adts=main_library();

// now find the dictionary for adt_operators

if(by_num_adts→isIndex(new_component→num_adts()))
{
    by_num_operators=(Dictionary*)(Entity*)(*by_num_adts)
        [new_component→num_adts()];
}
else
{
    by_num_operators=new Dictionary(OC_integer,
                                    OC_dictionary,
                                    TRUE,
                                    FALSE);

    by_num_adts→Insert(new_component→
        num_adts(),
        by_num_operators);
}

// have correct by_num_operator dictionary so get the
// generic types dict.

if(by_num_operators→isIndex(new_component→
    num_adt_operators()))
{
    by_num_generics=(Dictionary*)(Entity*)(
        (*by_num_operators)
        [ new_component→
            num_adt_operators()]);
}
else

```

```

{
    by_num_generics=new Dictionary(OC_integer,
                                   OC_dictionary,
                                   TRUE,
                                   FALSE);

    by_num_operators—Insert(new_component—
                           num_adt_operators(),
                           by_num_generics);

};

// got the generics dictionary so get the total base
// types dictionary
if(by_num_generics—isIndex(new_component—
                           num_generic_types())==TRUE)
{
    by_num_total_outputs=(Dictionary*)(Entity*)
        (*by_num_generics)
        [new_component—num_generic_types()];
}
else
{
    by_num_total_outputs=new Dictionary(OC_integer,
                                       OC_dictionary,
                                       TRUE,
                                       FALSE);

    by_num_generics—Insert(new_component—
                           num_generic_types(),
                           by_num_total_outputs);

};

if(by_num_total_outputs—
   isIndex(new_component—total_outputs())==TRUE)
{
    by_num_total_inputs=(Dictionary*)(Entity*)
        (*by_num_total_outputs)
        [new_component—total_outputs()];
}
else
{
    by_num_total_inputs=new Dictionary(OC_integer,
                                       OC_dictionary,
                                       TRUE,

```

```

        TRUE);

    by_num_total_outputs—Insert(new_component—
                                total_outputs(),
                                by_num_total_inputs);

};

if(by_num_total_inputs—
   isIndex(new_component—total_inputs())==TRUE)
{
    leaf_dictionary=(Dictionary*)(Entity*)
        (*by_num_total_inputs)
        [new_component—total_inputs()];
}
else
{
    leaf_dictionary=new Dictionary(OC_string,
                                   SB_ADT_COMPONENT_OType,
                                   FALSE,
                                   FALSE);

    by_num_total_inputs—Insert(new_component—
                                total_inputs(),
                                leaf_dictionary);

};

// have to leaf dictionary so now insert the component into it

leaf_dictionary—Insert(new_component—component_name(),
                       new_component);

by_num_adts—putObject();
by_num_operators—putObject();
by_num_total_inputs—putObject();
by_num_generics—putObject();
by_num_total_outputs—putObject();
leaf_dictionary—putObject();

return return_flag;
};

void SB_ADT_COMPONENT_LIBRARY::delete_component(SB_ADT_COMPONENT
*the_component)
{

    Dictionary *by_num_adts;
    Dictionary *by_num_operators;

```

```

Dictionary *by_num_total_inputs;
Dictionary *by_num_generics;
Dictionary *by_num_total_outputs;
Dictionary *leaf_dictionary;

adt_component_dictionary()→Remove(the_component→component_name());

adt_component_dictionary()→Dictionary::putObject();

by_num_adts=main_library();

// now find the dictionary for adt_operators

if(by_num_adts→isIndex(the_component→num_adts()))
{
    by_num_operators=(Dictionary *)(Entity *)(*by_num_adts)
        [the_component→num_adts()];
    // have correct by_num_operator dictionary so get the
    // generic types dict.

    if(by_num_operators→isIndex(the_component→
                                num_adt_operators()))
    {
        by_num_generics=(Dictionary *)(Entity *)
            (*by_num_operators)
            [the_component→
                num_adt_operators()];

        // got the generics dictionary so get the total base
        // types dictionary

        if(by_num_generics→isIndex(the_component→
                                    num_generic_types())==TRUE)
        {
            by_num_total_outputs=(Dictionary *)(Entity *)
                (*by_num_generics)
                [the_component→num_generic_types()];

            if(by_num_total_outputs→
                isIndex(the_component→total_outputs())==TRUE)
            {
                by_num_total_inputs=(Dictionary *)(Entity *)
                    (*by_num_total_outputs)
                    [the_component→total_outputs()];

                if(by_num_total_inputs→
                    isIndex(the_component→total_inputs())==TRUE)
                {
                    leaf_dictionary=(Dictionary *)(Entity *)

```

```

(*by_num_total_inputs)
  [the_component→total_inputs()];

// have to leaf dictionary

leaf_dictionary→Remove(the_component→component_name());
leaf_dictionary→putObject();
if(leaf_dictionary→Cardinality()==0)
{
  by_num_total_inputs→
    Remove(the_component→total_inputs());
  by_num_total_inputs→putObject();
  leaf_dictionary→deleteObject(TRUE);

  if(by_num_total_inputs→Cardinality()==0)
  {
    by_num_total_outputs→
      Remove(the_component→total_outputs());
    by_num_total_outputs→putObject();
    by_num_total_inputs→deleteObject(TRUE);

    if(by_num_total_outputs→Cardinality()==0)
    {
      by_num_generics→
        Remove(the_component→num_generic_types());
      by_num_generics→putObject();
      by_num_total_outputs→deleteObject(TRUE);

      if(by_num_generics→Cardinality()==0)
      {
        by_num_operators→
          Remove(the_component→num_adt_operators());
        by_num_operators→putObject();
        by_num_generics→deleteObject();

        if(by_num_operators→Cardinality()==0)
        {
          by_num_adts→
            Remove(the_component→num_adts());
          by_num_adts→putObject();
          by_num_operators→deleteObject();

          };

        };

      };

    };
  };
};

```



```

DictionaryIterator next_by_num_generics(by_num_generics,
                                         FALSE,
                                         query_component→num_unrecognized_types());

while(next_by_num_generics.moreData())
{
    by_num_total_outputs=(Dictionary *)(Entity *)
        next_by_num_generics();

    DictionaryIterator next_by_num_total_outputs(by_num_total_outputs,
                                                  FALSE,
                                                  query_component→total_outputs());

    while(next_by_num_total_outputs.moreData())
    {
        by_num_total_inputs=(Dictionary *)(Entity *)
            next_by_num_total_outputs();

        DictionaryIterator next_by_num_total_inputs(by_num_total_inputs,
                                                    FALSE,
                                                    query_component→total_inputs());

        while(next_by_num_total_inputs.moreData())
        {
            leaf_dictionary=(Dictionary *)(Entity *)
                next_by_num_total_inputs();

            DictionaryIterator next_component(leaf_dictionary);

            while(next_component.moreData())
            {
                SB_ADT_COMPONENT *the_component=
                    (SB_ADT_COMPONENT *)(Entity *)next_component();
                if(query_component→filter(the_component)==TRUE)
                {
                    query_result→add(the_component);
                }
            }
        }
    }
}

// add code here for semantic matching interface

return query_result;

```

```
};  
  
void SB_ADT_COMPONENT_LIBRARY::list(ofstream& outstream)  
{  
    adt_component_dictionary().—printOn(outstream);  
};
```

```

# include "sball.hxx"

# include "sbextern.h"

SB_ADT_OPERATOR::SB_ADT_OPERATOR(APL *theAPL) :
SB_OPERATOR(theAPL)
{
};

SB_ADT_OPERATOR::SB_ADT_OPERATOR(char *id) :
SB_OPERATOR(id)
{
};

void SB_ADT_OPERATOR::Destroy(Boolean aborted)
{
    SB_OPERATOR::Destroy(aborted);
};

void SB_ADT_OPERATOR::deleteObject(Boolean deallocate)
{
    SB_OPERATOR::deleteObject(deallocate);
};

void SB_ADT_OPERATOR::putObject(Boolean deallocate)
{
    SB_OPERATOR::putObject(deallocate);
};

Type *SB_ADT_OPERATOR::getDirectType()
{
    return SB_ADT_OPERATOR_OType;
};

Boolean SB_ADT_OPERATOR::process_type_info(SB_ADT_COMPONENT *adt)
{
    // process types by checking local generic then adt.adt usage then
    // adt.generic usage before making it unrecognized. This will update
    // the adt usage dictionaries as well

    // update all usage dictionaries for inputs and outputs
    //
    // first go through all of the inputs
    DictionaryIterator next_input=input_attributes()→id_iterator();
    while(next_input.moreData())
    {
        SB_ID_DECL *this_decl=(SB_ID_DECL *) (Entity *) next_input();
        SB_TYPE_NAME *this_type_name=this_decl→type_name();
        // first see if this id_decl type is a generic
    }
}

```

```

if(generic_usage()→update(this_type_name)==FALSE)
{
    // was not a generic type check the ADT generic list

    if(adt→generic_usage()→update(this_type_name)==FALSE)
    {
        // was not an adt generic so check the adt list
        if(adt→adt_usage()→update(this_type_name)==FALSE)
        {
            // was not an adt adt so put it in its local list

            // based on whether or not it is recognized
            if(this_type_name→recognized()==FALSE)
            {
                // was unrecognized so try to update
                // the unrecognized list or add it to
                // the list
                if(unrecognized_type_usage()→
                    update(this_type_name)==FALSE)
                {
                    // not yet in list so add it
                    unrecognized_type_usage()→
                        add_type(this_type_name→id(),
                            this_type_name);
                    // now update it for being used once
                    unrecognized_type_usage()→
                        update(this_type_name);
                }
            }
        }
    }
    else
    {
        // this type name is recognized so update
        // or add it
        if(recognized_type_usage()→
            update(this_type_name)==FALSE)
        {
            // not yet in list so add it
            recognized_type_usage()→
                add_type(this_type_name→id(),
                    this_type_name);
            // now update it for being used once
            recognized_type_usage()→
                update(this_type_name);
        }
    }
}

};

};

};

DictionaryIterator next_output=output_attributes()→id_iterator();
while(next_output.moreData())

```

```

{
  SBJD_DECL *this_decl=(SBJD_DECL *) (Entity *)next_output();
  SB_TYPE_NAME *this_type_name=this_decl->type_name();

  // first see if this id_decl type is a generic
  if(generic_usage()->update(this_type_name)==FALSE)
  {

    // was not a generic type check the ADT generic list

    if(ad->generic_usage()->update(this_type_name)==FALSE)
    {
      // was not an adt generic so check the adt list
      if(ad->adt_usage()->update(this_type_name)==FALSE)
      {

        // was not an adt adt
        // so put it in its local list
        // based on whether or not it is recognized
        if(this_type_name->recognized()==FALSE)
        {
          // was unrecognized so try to update
          // the unrecognized list or add it to
          // the list
          if(unrecognized_type_usage()->
            update(this_type_name)==FALSE)
          {
            // not yet in list so add it
            unrecognized_type_usage()->
              add_type(this_type_name->id(),
                this_type_name);
            // now update it for being used once
            unrecognized_type_usage()->
              update(this_type_name);
          };
        }

        // now update the adt list as well

        if(ad->unrecognized_type_usage()->
          update(this_type_name)==FALSE)
        {
          // not yet in list so add it
          ad->unrecognized_type_usage()->
            add_type(this_type_name->id(),
              this_type_name);
          // now update it for being used once
          ad->unrecognized_type_usage()->
            update(this_type_name);
        };
      }
    }
  }
}

```

```

else
{
    // this type name is recognized so update
    // or add it
    if(recognized_type_usage()→
        update(this_type_name)==FALSE)
    {
        // not yet in list so add it
        recognized_type_usage()→
            add_type(this_type_name→id(),
                this_type_name);
        // now update for being used onece
        recognized_type_usage()→
            update(this_type_name);
    };
    // now update the adt usage list
    if(adt→recognized_type_usage()→
        update(this_type_name)==FALSE)
    {
        // not yet in list so add it
        adt→recognized_type_usage()→
            add_type(this_type_name→id(),
                this_type_name);
        // now update for being used onece
        adt→recognized_type_usage()→
            update(this_type_name);
    };
};

};

};

};
return TRUE;
};

```

```
#include "sball.hxx"
```

```
# include "sbextern.h"
```

```
SB_ADT_OPERATOR_DICTIONARY::SB_ADT_OPERATOR_DICTIONARY(APL *theAPL) :  
Dictionary(theAPL)  
{  
};
```

```
SB_ADT_OPERATOR_DICTIONARY::SB_ADT_OPERATOR_DICTIONARY() : Dictionary  
(OC_integer, // key  
                                     SB_ADT_OPERATOR_OType,  
                                     TRUE,  
                                     TRUE)  
{  
};
```

```
void SB_ADT_OPERATOR_DICTIONARY::Destroy(Boolean aborted)  
{  
    DictionaryIterator next_operator(this);  
    while(next_operator.moreData())  
    {  
        ((SB_ADT_OPERATOR *) (Entity *) next_operator()) → Destroy(aborted);  
    };  
  
    Dictionary::Destroy(aborted);  
};
```

```
void SB_ADT_OPERATOR_DICTIONARY::deleteObject(Boolean deallocate)  
{  
  
    DictionaryIterator next_operator(this);  
    while(next_operator.moreData())  
    {  
        ((SB_ADT_OPERATOR *) (Entity *) next_operator()) → deleteObject(FALSE);  
    };  
  
    Dictionary::deleteObject(deallocate);  
};
```

```
void SB_ADT_OPERATOR_DICTIONARY::putObject(Boolean deallocate)  
{  
  
    DictionaryIterator next_operator(this);  
    while(next_operator.moreData())  
    {  
        ((SB_ADT_OPERATOR *) (Entity *) next_operator()) → putObject(deallocate);  
    };  
};
```

```

    Dictionary::putObject(deallocate);

};

Type *SB_ADT_OPERATOR_DICTIONARY::getDirectType()
{
    return SB_ADT_OPERATOR_DICTIONARY_OType;
};

void SB_ADT_OPERATOR_DICTIONARY::add(SB_ADT_OPERATOR *op)
{
    Insert(op->num_inputs().op);
};

void SB_ADT_OPERATOR_DICTIONARY::append(SB_ADT_OPERATOR_DICTIONARY
*new_dict)
{
    // get an iterator for the dictionary then insert each operator into
    // the dictionary. Finally delete the input dictionary

    DictionaryIterator next_operator=new_dict->iterator();
    while(next_operator.moreData())
    {
        SB_ADT_OPERATOR *the_operator=
            (SB_ADT_OPERATOR *) (Entity *) next_operator();
        this->add(the_operator);
    };
    next_operator.Reset();
    // destroy the dictionary new_dict but not its members
    new_dict->Dictionary::Destroy(FALSE);
};

void SB_ADT_OPERATOR_DICTIONARY::printOn(ofstream& outstream)
{
    DictionaryIterator next_operator=iterator();

    while(next_operator.moreData())
    {
        ((SB_ADT_OPERATOR *) (Entity *) next_operator())->printOn(outstream);
    };
};

int SB_ADT_OPERATOR_DICTIONARY::num()
{
    return (int) Cardinality(); // cast to int (no need for long)
};

```

```

int SB_ADT_OPERATOR_DICTIONARY::total_inputs()
{
    int total=0;

    DictionaryIterator next_operator=iterator();

    while(next_operator.moreData())
    {
        total=total +
            ((SB_ADT_OPERATOR *)(Entity *)next_operator())→num_inputs();
    };

    return total;
};

```

```

int SB_ADT_OPERATOR_DICTIONARY::total_outputs()
{
    int total=0;

    DictionaryIterator next_operator=iterator();

    while(next_operator.moreData())
    {
        total=total +
            ((SB_OPERATOR *)(Entity *)next_operator())→num_outputs();
    };

    return total;
};

```

```

DictionaryIterator SB_ADT_OPERATOR_DICTIONARY::iterator()
{
    return DictionaryIterator((Dictionary *)this);
};

```

```

# include "sball.hxx"

# include "sbextern.h"

SB_COMPONENT::SB_COMPONENT(APL *theAPL) : Object(theAPL)
{
};

SB_COMPONENT::SB_COMPONENT(char *id) : Object()
{
    the_component_name=new char[strlen(id)+1];
    strcpy(the_component_name,id);

    SB_KEYWORD_DICTIONARY *new_keyword_dictionary=new
    SB_KEYWORD_DICTIONARY();

    the_keyword_dictionary=new_keyword_dictionary→findTRef();

    SB_TEXT_OBJECT *new_psdl_text=new SB_TEXT_OBJECT();
    the_psdl_text=new_psdl_text→findTRef();

    SB_TEXT_OBJECT *new_imp_spec_text=new SB_TEXT_OBJECT();
    the_imp_spec_text=new_imp_spec_text→findTRef();

    SB_TEXT_OBJECT *new_imp_body_text=new SB_TEXT_OBJECT();
    the_imp_body_text=new_imp_body_text→findTRef();

    SB_TEXT_OBJECT *new_informal_description=new SB_TEXT_OBJECT();
    the_informal_description=new_informal_description→findTRef();

    SB_TEXT_OBJECT *new_formal_description=new SB_TEXT_OBJECT();
    the_formal_description=new_formal_description→findTRef();

    SB_TEXT_OBJECT *new_norm_formal_description=new SB_TEXT_OBJECT();
    the_norm_formal_description=new_norm_formal_description→findTRef();

    SB_TYPE_USAGE_DICTIONARY *new_recognized_type_usage=
        new SB_TYPE_USAGE_DICTIONARY();
    the_recognized_type_usage=new_recognized_type_usage→findTRef();

    SB_TYPE_USAGE_DICTIONARY *new_unrecognized_type_usage=
        new SB_TYPE_USAGE_DICTIONARY();
    the_unrecognized_type_usage=new_unrecognized_type_usage→findTRef();

    SB_TYPE_USAGE_DICTIONARY *new_generic_usage=new
    SB_TYPE_USAGE_DICTIONARY();

    the_generic_usage=new_generic_usage→findTRef();

```

```

};

void SB_COMPONENT::Destroy(Boolean aborted)
{
    psdl_text()→Destroy(aborted);
    imp_spec_text()→Destroy(aborted);
    imp_body_text()→Destroy(aborted);
    informal_description()→Destroy(aborted);
    formal_description()→Destroy(aborted);
    norm_formal_description()→Destroy(aborted);
    recognized_type_usage()→Destroy(aborted);
    unrecognized_type_usage()→Destroy(aborted);
    keyword_dictionary()→Destroy(aborted);
    generic_usage()→Destroy(aborted);

    delete the_component_name;
    delete the_keyword_dictionary;
    delete the_psdl_text;
    delete the_imp_spec_text;
    delete the_imp_body_text;
    delete the_informal_description;
    delete the_formal_description;
    delete the_norm_formal_description;
    delete the_recognized_type_usage;
    delete the_unrecognized_type_usage;
    delete the_generic_usage;

    Object::Destroy(aborted);
};

void SB_COMPONENT::deleteObject(Boolean deallocate)
{
    psdl_text()→deleteObject(FALSE);
    imp_spec_text()→deleteObject(FALSE);
    imp_body_text()→deleteObject(FALSE);
    informal_description()→deleteObject(FALSE);
    formal_description()→deleteObject(FALSE);
    norm_formal_description()→deleteObject(FALSE);
    recognized_type_usage()→deleteObject(FALSE);
    unrecognized_type_usage()→deleteObject(FALSE);
    keyword_dictionary()→deleteObject(FALSE);
    generic_usage()→deleteObject(FALSE);
    Object::deleteObject(deallocate);
};

void SB_COMPONENT::putObject(Boolean deallocate)
{
    psdl_text()→putObject(deallocate);
    imp_spec_text()→putObject(deallocate);
    imp_body_text()→putObject(deallocate);

```

```

informal_description()→putObject(deallocate);
formal_description()→putObject(deallocate);
norm_formal_description()→putObject(deallocate);
recognized_type_usage()→putObject(deallocate);
unrecognized_type_usage()→putObject(deallocate);
keyword_dictionary()→putObject(deallocate);
generic_usage()→putObject(deallocate);
Object::putObject(deallocate);
};

•
SB_KEYWORD_DICTIONARY *SB_COMPONENT::keyword_dictionary()
{
    return ((SB_KEYWORD_DICTIONARY *) (the_keyword_dictionary→Binding()));
};

•
SB_TEXT_OBJECT *SB_COMPONENT::psdl_text()
{
    return((SB_TEXT_OBJECT *) (the_psdl_text→Binding()));
};

SB_TEXT_OBJECT *SB_COMPONENT::imp_spec_text()
{
    return((SB_TEXT_OBJECT *) (the_imp_spec_text→Binding()));
};

SB_TEXT_OBJECT *SB_COMPONENT::imp_body_text()
{
    return((SB_TEXT_OBJECT *) (the_imp_body_text→Binding()));
};

SB_TEXT_OBJECT *SB_COMPONENT::informal_description()
{
    return((SB_TEXT_OBJECT *) (the_informal_description→Binding()));
};

SB_TEXT_OBJECT *SB_COMPONENT::formal_description()
{
    return((SB_TEXT_OBJECT *) (the_formal_description→Binding()));
};

SB_TEXT_OBJECT *SB_COMPONENT::norm_formal_description()
{
    return((SB_TEXT_OBJECT *) (the_norm_formal_description→
                                Binding()));
};

SB_TYPE_USAGE_DICTIONARY * SB_COMPONENT::recognized_type_usage()
{
    return (SB_TYPE_USAGE_DICTIONARY*)

```

```

    the_recognized_type_usage—Binding();
};

SB_TYPE_USAGE_DICTIONARY* SB_COMPONENT::unrecognized_type_usage()
{
    return (SB_TYPE_USAGE_DICTIONARY*)
        the_unrecognized_type_usage—Binding();
};

SB_TYPE_USAGE_DICTIONARY* SB_COMPONENT::generic_usage()
{
    return (SB_TYPE_USAGE_DICTIONARY*)the_generic_usage—Binding();
};

Boolean SB_COMPONENT::add_keyword(char *keyword)
{
    return keyword_dictionary()—add(keyword);
};

int SB_COMPONENT::num_unrecognized_types()
{
    return int(unrecognized_type_usage()—num());
};

int SB_COMPONENT::total_types()
{
    return num_unrecognized_types()+
        recognized_type_usage()—num();
};

void SB_COMPONENT::insert_generics( SB_TYPE_USAGE_DICTIONARY *new_generic_usage)
{
    if(new_generic_usage≠NULL)
    {
        SB_COMPONENT::generic_usage()—append(new_generic_usage);
    }
};

char *SB_COMPONENT::component_name()
{
    return the_component_name;
};

void SB_COMPONENT::add_text(ifstream& psdl, ifstream& spec, ifstream& body)
{
    psdl.text()—append(psdl);
};

```

```
imp_spec_text()—append(spec);  
imp_body_text()—append(body);  
};
```

```
# include "sball.hxx"
# include "sbextern.h"
```

```
SB_COMPONENT_DICTIONARY::SB_COMPONENT_DICTIONARY(APL *theAPL) :
Dictionary(theAPL)
{
};
```

```
SB_COMPONENT_DICTIONARY::SB_COMPONENT_DICTIONARY() : Dictionary
(OC_string, // KEY
                                     SB_COMPONENT_OType,
                                     TRUE,
                                     FALSE)
{
};
```

```
Type *SB_COMPONENT_DICTIONARY::getDirectType()
{
    return SB_COMPONENT_DICTIONARY_OType;
};
```

```
void SB_COMPONENT_DICTIONARY::Destroy(Boolean aborted)
{
    // first destroy all of the references in the dictionary

    DictionaryIterator next_component(this);

    while(next_component.moreData())
    {
        ((SB_COMPONENT *) (Entity *) next_component())->Destroy(aborted);
    };

    Dictionary::Destroy(aborted);
};
```

```
void SB_COMPONENT_DICTIONARY::deleteObject(Boolean deallocate)
{
    // first delete all of the references in the dictionary

    DictionaryIterator next_component(this);

    while(next_component.moreData())
    {
        ((SB_COMPONENT *) (Entity *) next_component())->deleteObject(FALSE);
    };

    Dictionary::deleteObject(deallocate);
};
```

```

};

void SB_COMPONENT_DICTIONARY::putObject(Boolean deallocate)
{
    // first put all of the references in the dictionary

    DictionaryIterator next_component(this);

    while(next_component.moreData())
    {
        ((SB_COMPONENT*)(Entity*)next_component())->putObject(deallocate);
    };

    Dictionary::putObject(deallocate);
};

Boolean SB_COMPONENT_DICTIONARY::add(SB_COMPONENT *new_component)
{
    Boolean return_flag;

    if(Dictionary::isIndex(new_component->component_name())==FALSE)
    {
        // keyword is not yet in the dictionary so insert it

        Dictionary::Insert(new_component->component_name(),new_component);
        return_flag=TRUE;
    }
    else
    {
        return_flag=FALSE;
    };

    return return_flag;
};

SB_COMPONENT *SB_COMPONENT_DICTIONARY::query(char *name)
{
    SB_COMPONENT *return_component=NULL;

    if(Dictionary::isIndex(name)==TRUE)
    {
        return_component=(SB_COMPONENT*)(Entity*)(*this)[name];
    };
    return return_component;
};

void SB_COMPONENT_DICTIONARY::printOn(ofstream& outstream)
{

```

```

DictionaryIterator next_component=
    DictionaryIterator(this);

while(next_component.moreData())
{
    int i;
    SB_COMPONENT *the_component=(SB_COMPONENT *) (Entity *)next_component();
    ostream << the_component->component_name();
    for(i=strlen(the_component->component_name());i < DEFAULT_NAME_SIZE; i++)
    {
        ostream << " ";
    };
    ostream << " ";
    char *informal_desc=(the_component->informal_description())->text();
    i=0;
    while(informal_desc[i]≠NULL && informal_desc[i]≠'\n')
    {
        ostream << informal_desc[i];
        i++;
    };
    ostream << "\n";
};
};

```

```

# include "sball.hxx"
# include "sbextern.h"

SB_EXCEPTION_DICTIONARY::SB_EXCEPTION_DICTIONARY(APL *theAPL) :
Dictionary(theAPL)
{
};

void SB_EXCEPTION_DICTIONARY::Destroy(Boolean aborted)
{
    DictionaryIterator next_exception(this);
    while(next_exception.moreData())
    {
        delete (char *)next_exception();
    };

    Dictionary::Destroy(aborted);
};

void SB_EXCEPTION_DICTIONARY::deleteObject(Boolean deallocate)
{
    Dictionary::deleteObject(deallocate);
};

void SB_EXCEPTION_DICTIONARY::putObject(Boolean deallocate)
{
    Dictionary::putObject(deallocate);
};

Type *SB_EXCEPTION_DICTIONARY::getDirectType()
{
    return SB_EXCEPTION_DICTIONARY_OType;
};

SB_EXCEPTION_DICTIONARY::SB_EXCEPTION_DICTIONARY() : Dictionary (OC_string,
// KEY
OC_string,
TRUE,
FALSE)
{
};

Boolean SB_EXCEPTION_DICTIONARY::add(char *exception_id)
{

```

```

    Boolean return_flag;

    if(Dictionary::isIndex(exception_id)==FALSE)
    {
        // exception_id is not yet in the dictionary so insert it

        Dictionary::Insert(exception_id,"");

        return_flag=TRUE;
    }
    else
    {
        return_flag=FALSE;
    };

    return return_flag;

};

Boolean SB_EXCEPTION_DICTIONARY::append(SB_EXCEPTION_DICTIONARY
*dictionary)
{
    Boolean return_flag=TRUE;

    DictionaryIterator next_id=dictionary->iterator();

    while(next_id.moreData() && return_flag==TRUE)
    {
        if(add((char *)next_id())==FALSE)
        {
            return_flag=FALSE;
        };
    };

    dictionary->Destroy(FALSE); // delete the object and deallocate

    return return_flag;
};

void SB_EXCEPTION_DICTIONARY::printOn(ofstream& outstream)
{
    DictionaryIterator next_exception=iterator();

    while(next_exception.moreData())
    {
        outstream << (char *)next_exception() << "\n";
    };
};

```

```
DictionaryIterator SB_EXCEPTION_DICTIONARY::iterator()
{
    // use tagiterate since tag is the data

    return DictionaryIterator(this, TRUE);
};
```

```

# include "sball.hxx"

# include "sbextern.h"

SB_ID_DECL::SB_ID_DECL(APL *theAPL) : Object(theAPL)
{
};

SB_ID_DECL::SB_ID_DECL(char *new_the_id,
                        SB_TYPE_NAME *new_type_name):
                        Object()
{

    the_id=new char[strlen(new_the_id)+1];
    strcpy(the_id,new_the_id);

    the_type_name=new_type_name—findTRef();

};

void SB_ID_DECL::Destroy(Boolean aborted)
{
    if(the_id≠NULL)
    {
        delete the_id;
    };

    type_name()—Destroy(aborted);

    Object::Destroy(aborted);
};

void SB_ID_DECL::deleteObject(Boolean deallocate)
{
    type_name()—deleteObject(FALSE);

    Object::deleteObject(deallocate);
};

void SB_ID_DECL::putObject(Boolean deallocate)
{
    type_name()—putObject(deallocate);

    Object::putObject(deallocate);
};

```

```

SB_TYPE_NAME *SB_ID_DECL::type_name()
{
    return ((SB_TYPE_NAME *) (the_type_name→Binding()));
};

```

```

Type *SB_ID_DECL::getDirectType()
{
    return SB_ID_DECL_OType;
};

```

```

void SB_ID_DECL::printOn(ofstream& outstream)
{
    outstream << the_id << " : ";
    type_name()→printOn(outstream);
};

```

```

char *SB_ID_DECL::id()
{
    return the_id;
};

```

```

# include "sball.hxx"

# include "sbextern.h"

SB_ID_DECL_DICTIONARY::SB_ID_DECL_DICTIONARY(APL *theAPL) : Object(theAPL)
{
};

SB_ID_DECL_DICTIONARY::SB_ID_DECL_DICTIONARY() : Object()
{

    Dictionary *new_dictionary_by_type= new Dictionary(SB_TYPE_NAME_OType,
                                                         SB_ID_DECL_OType,
                                                         TRUE,
                                                         TRUE);

    the_dictionary_by_type=new_dictionary_by_type→findTRef();

    Dictionary *new_dictionary_by_id= new Dictionary(OC_string,
                                                         SB_ID_DECL_OType,
                                                         TRUE,
                                                         FALSE);

    the_dictionary_by_id=new_dictionary_by_id→findTRef();

    List *new_id_declaration_list=new List(SB_ID_DECL_OType);

    the_id_declaration_list=new_id_declaration_list→findTRef();

};

void SB_ID_DECL_DICTIONARY::Destroy(Boolean aborted)
{

    ListIterator next_id_decl(id_declaration_list());
    while(next_id_decl.moreData())
    {
        ((SB_ID_DECL *)(Entity *)next_id_decl())→Destroy(aborted);
    };

    dictionary_by_type()→Destroy(aborted);
    dictionary_by_id()→Destroy(aborted);
    id_declaration_list()→Destroy(aborted);

    delete the_dictionary_by_type;

```

```

    delete the_dictionary_by_id;
    delete the_id_declaration_list;

    Object::Destroy(aborted);

};

void SB_ID_DECL_DICTIONARY::deleteObject(Boolean deallocate)
{
    ListIterator next_id_decl(id_declaration_list());
    while(next_id_decl.moreData())
    {
        ((SB_ID_DECL *)(Entity *)next_id_decl())→deleteObject(FALSE);
    };

    dictionary_by_type()→deleteObject(FALSE);
    dictionary_by_id()→deleteObject(FALSE);
    id_declaration_list()→deleteObject(FALSE);

    Object::deleteObject(deallocate);

};

void SB_ID_DECL_DICTIONARY::putObject(Boolean deallocate)
{
    ListIterator next_id_decl(id_declaration_list());
    while(next_id_decl.moreData())
    {
        ((SB_ID_DECL *)(Entity *)next_id_decl())→putObject(deallocate);
    };

    dictionary_by_type()→putObject(deallocate);
    dictionary_by_id()→putObject(deallocate);
    id_declaration_list()→putObject(deallocate);

    Object::putObject(deallocate);

};

Type *SB_ID_DECL_DICTIONARY::getDirectType()
{
    return SB_ID_DECL_DICTIONARY_OType;
};

Dictionary *SB_ID_DECL_DICTIONARY::dictionary_by_type()
{
    return (Dictionary *)(the_dictionary_by_type→Binding());
};

Dictionary *SB_ID_DECL_DICTIONARY::dictionary_by_id()

```

```

{
    return (Dictionary *)(the_dictionary_by_id→Binding());
};

List *SB_ID_DECL_DICTIONARY::id_declaration_list()
{
    return (List *)(the_id_declaration_list→Binding());
};

Boolean SB_ID_DECL_DICTIONARY::add_decl(SB_ID_DECL *decl)
{
    Boolean return_flag;

    if(dictionary_by_id()→isIndex(decl→id())==FALSE)
    {
        // ID NOT YET USED

        dictionary_by_type()→Insert(decl→type_name(),decl);

        dictionary_by_id()→Insert(decl→id(),decl);

        id_declaration_list()→Insert(decl);

        return_flag=TRUE;
    }
    else
    {
        // id already in use so can not insert
        return_flag=FALSE;
    };

    return return_flag;
};

void SB_ID_DECL_DICTIONARY::remove_decl(SB_ID_DECL *decl)
{
    dictionary_by_type()→Remove(decl→type_name(),decl);

    dictionary_by_id()→Remove(decl→id(),decl);

    id_declaration_list()→Remove(id_declaration_list()→Index(decl));
};

```

```
};
```

```
Boolean SB_ID_DECL_DICTIONARY::add_decl(char *id, SB_TYPE_NAME *type_name)
```

```
{
```

```
    Boolean return_flag;
```

```
    if(dictionary_by_id()→isIndex(id)==FALSE)
```

```
    {
```

```
        // ID NOT YET USED
```

```
        // create new SB_ID_DECL
```

```
        SB_ID_DECL *decl=new SB_ID_DECL(id,type_name);
```

```
        dictionary_by_type()→Insert(type_name,decl);
```

```
        dictionary_by_id()→Insert(id,decl);
```

```
        id_declaration_list()→Insert(decl);
```

```
        return_flag=TRUE;
```

```
    }
```

```
    else
```

```
    {
```

```
        // id already in use so can not insert
```

```
        return_flag=FALSE;
```

```
    };
```

```
    return return_flag;
```

```
};
```

```
Boolean SB_ID_DECL_DICTIONARY::append(SB_ID_DECL_DICTIONARY *dictionary)
```

```
{
```

```
    Boolean return_flag=TRUE;
```

```
    ListIterator next_id=dictionary→order_iterator();
```

```
    while(next_id.moreData() && return_flag==TRUE)
```

```
    {
```

```
        SB_ID_DECL *this_decl=(SB_ID_DECL *) (Entity *)next_id();
```

```
        return_flag=add_decl( this_decl );
```

```
    }
```

```

    /* next_id.Reset();
    while(next_id.moreData())
    {
        SB_ID_DECL *this_decl=(SB_ID_DECL *)(Entity *)next_id();
        dictionary->remove_decl(this_decl);
    };
    dictionary->Destroy(FALSE);*/

    return return_flag;
};

void SB_ID_DECL_DICTIONARY::printOn(ofstream& outstream)
{
    ListIterator next_decl=order_iterator();
    if(next_decl.moreData())
    {
        ((SB_ID_DECL *)(Entity *) next_decl())->printOn(outstream);

        while(next_decl.moreData())
        {
            outstream << ",\n";
            ((SB_ID_DECL *)(Entity *) next_decl())->printOn(outstream);
        };
    };
};

int SB_ID_DECL_DICTIONARY::num()
{
    return (int)(dictionary_by_type()->Cardinality());
};

DictionaryIterator SB_ID_DECL_DICTIONARY::id_iterator()
{
    return DictionaryIterator(dictionary_by_id());
};

DictionaryIterator SB_ID_DECL_DICTIONARY::type_iterator()
{
    return DictionaryIterator(dictionary_by_type());
};

ListIterator SB_ID_DECL_DICTIONARY::order_iterator()
{
    return ListIterator(id_declaration_list());
};

SB_ID_DECL *SB_ID_DECL_DICTIONARY::query_id(char *query_name)
{

```

```
SB_ID_DECL *return_value=NULL;
if(dictionary_by_id()→isIndex(query_name))
{
    return_value=(SB_ID_DECL *)(Entity *)dictionary_by_id()→
        getEntityElement(query_name);
};
return return_value;
};
```

```
# include "sball.hxx"
# include "sbextern.h"
```

```
SB_KEYWORD_DICTIONARY::SB_KEYWORD_DICTIONARY(APL *theAPL) :
Dictionary(theAPL)
{
};
```

```
SB_KEYWORD_DICTIONARY::SB_KEYWORD_DICTIONARY() : Dictionary (OC_string, //
KEY
OC_string,
TRUE,
FALSE)
```

```
{
};
```

```
Type *SB_KEYWORD_DICTIONARY::getDirectType()
{
return SB_KEYWORD_DICTIONARY_OType;
};
```

```
void SB_KEYWORD_DICTIONARY::Destroy(Boolean aborted)
{
Dictionary::Destroy(aborted);
};
```

```
void SB_KEYWORD_DICTIONARY::deleteObject(Boolean deallocate)
{
Dictionary::deleteObject(deallocate);
};
```

```
void SB_KEYWORD_DICTIONARY::putObject(Boolean deallocate)
{
Dictionary::putObject(deallocate);
};
```

```
Boolean SB_KEYWORD_DICTIONARY::add(char *keyword)
{
Boolean return_flag;

if(Dictionary::isIndex(keyword)==FALSE)
{
// keyword is not yet in the dictionary so insert it

Dictionary::Insert(keyword,"");
return_flag=TRUE;
}
else
{
return_flag=FALSE;
}
```

```

    };

    return return_flag;

};

void SB_KEYWORD_DICTIONARY::printOn(ofstream& outstream)
{
    DictionaryIterator next_keyword=iterator();

    while(next_keyword.moreData())
    {
        outstream << (char *)next_keyword() << "\n";
    };
};

DictionaryIterator SB_KEYWORD_DICTIONARY::iterator()
{
    // use tagiterate since tag is the data

    return DictionaryIterator(this.TRUE);
};

```

```

#include "sball.hxx"
#include "sbextern.h"

SB_KEYWORD_LIBRARY::SB_KEYWORD_LIBRARY(APL *theAPL) : Dictionary(theAPL)
{
};

SB_KEYWORD_LIBRARY::SB_KEYWORD_LIBRARY() :
Dictionary(OC_string,SB_COMPONENT_DICTIONARY_otype,TRUE,FALSE)
{

};

void SB_KEYWORD_LIBRARY::query(ifstream& instream, ofstream& outstream)
{
    char the_keyword[256];

    Dictionary *the_result=new
        Dictionary(OC_string,OC_integer,FALSE,FALSE);

    while(!instream.eof())
    {
        instream >> the_keyword;
        instream >> ws; // get the new_line character or eof
        if(this->isIndex(the_keyword)==TRUE)
        {
            SB_COMPONENT_DICTIONARY *the_component_dictionary=
                (SB_COMPONENT_DICTIONARY *)
                this->getEntityElement(the_keyword);
            DictionaryIterator next_component=
                DictionaryIterator(the_component_dictionary);
            // update the result dictionary

            while(next_component.moreData())
            {
                SB_COMPONENT *the_component=(SB_COMPONENT *)
                    (Entity *)next_component();
                char *the_component_name;
                the_component_name=the_component->component_name();

                if(the_result->
                    isIndex(the_component_name)==TRUE)
                {
                    int new_number= int(
                        the_result->getIntegerElement(the_component_name))-
1;

                    the_result->Remove(the_component_name);
                    the_result->
                        Insert(the_component_name,new_number);
                }
                else
                {

```

```

        the_result—Insert(the_component_name,-1);
    };
};
};

// create new dictionary ordered by number of times found
Dictionary *final_result=new
    Dictionary(OC_integer,OC_string,TRUE,TRUE);

DictionaryIterator next_result=DictionaryIterator(the_result);
DictionaryIterator next_result_tag=DictionaryIterator(the_result,TRUE);
while(next_result.moreData())
{
    char *component=(char *)next_result_tag();
    int times_used=int(next_result());
    final_result—Insert(times_used,component);
};

DictionaryIterator next_final_result=DictionaryIterator(final_result);
while(next_final_result.moreData())
{
    char* component_name=(char *)next_final_result();
    SB_COMPONENT *the_component=SB.MAIN_LIBRARY—query(component_name);
    // SB.MAIN_LIBRARY IS GLOBAL
    int i;
    ostream << the_component—component_name();
    for(i=strlen(the_component—component_name());i < DEFAULT_NAME_SIZE; i++)
    {
        ostream << " ";
    };
    ostream << " ";
    char *informal_desc=(the_component—informal_description())—text();
    i=0;
    while(informal_desc[i]≠NULL && informal_desc[i]≠'\n')
    {
        ostream << informal_desc[i];
        i++;
    };
    ostream << "\n";
};

the_result—Destroy();
final_result—Destroy();
};

Boolean SB_KEYWORD_LIBRARY::add_component(SB_COMPONENT *new_component)
{
    SB_KEYWORD_DICTIONARY* keyword_list=new_component—keyword_dictionary();
    DictionaryIterator next_keyword=keyword_list—iterator();
    while(next_keyword.moreData())
    {
        char *the_keyword=(char *)next_keyword();

```

```

        if(this→isIndex(the_keyword)==TRUE)
        {
            SB_COMPONENT_DICTIONARY
            *the_dictionary=(SB_COMPONENT_DICTIONARY *)
                this→getEntityElement(the_keyword);
            the_dictionary→add(new_component);
            the_dictionary→Dictionary::putObject();
        }
        else
        {
            // this is a new keyword so make a new sb_component_dict
            // and add it to the key_word library
            SB_COMPONENT_DICTIONARY* new_dictionary=new
                SB_COMPONENT_DICTIONARY();
            new_dictionary→add(new_component);
            new_dictionary→Dictionary::putObject();
            this→Insert(the_keyword,new_dictionary);
        }
    };

    return TRUE;
};

void SB_KEYWORD_LIBRARY::delete_component(SB_COMPONENT *the_component)
{
    SB_KEYWORD_DICTIONARY* keyword_list=the_component→keyword_dictionary();
    DictionaryIterator next_keyword=keyword_list→iterator();
    while(next_keyword.moreData())
    {
        char *the_keyword=(char *)next_keyword();
        if(this→isIndex(the_keyword)==TRUE)
        {
            SB_COMPONENT_DICTIONARY
            *the_dictionary=(SB_COMPONENT_DICTIONARY *)
                this→getEntityElement(the_keyword);
            the_dictionary→Remove(the_component→component_name());
            the_dictionary→putObject();
            if(the_dictionary→Cardinality()==0)
            {
                this→Remove(the_keyword);
                the_dictionary→deleteObject(TRUE);
            }
        }
    };
};

void SB_KEYWORD_LIBRARY::Destroy(Boolean aborted)
{

```

```

DictionaryIterator next_component_dictionary=iterator();
while(next_component_dictionary.moreData())
{
    ((SB_COMPONENT_DICTIONARY *)(Entity *)next_component_dictionary())→
        Destroy(aborted);
}
Dictionary::Destroy(aborted);
};

void SB_KEYWORD_LIBRARY::deleteObject(Boolean deallocate)
{
    DictionaryIterator next_component_dictionary=iterator();
    while(next_component_dictionary.moreData())
    {
        ((SB_COMPONENT_DICTIONARY *)(Entity *)next_component_dictionary())→
            deleteObject(FALSE);
    }
    Dictionary::deleteObject(deallocate);
};

void SB_KEYWORD_LIBRARY::putObject(Boolean deallocate)
{
    Dictionary::putObject(deallocate);
};

DictionaryIterator SB_KEYWORD_LIBRARY::iterator()
{
    return DictionaryIterator(this,TRUE); // return tag iterate
};

void SB_KEYWORD_LIBRARY::list(ofstream& outstream)
{
    DictionaryIterator next_keyword=iterator();
    while(next_keyword.moreData())
    {
        outstream << (char *)next_keyword() << "\n";
    }
};

```

```

# include "sball.hxx"

# include "sbextern.h"

SB_LIBRARY::SB_LIBRARY(APL *theAPL): Object(theAPL)
{
};

SB_LIBRARY::SB_LIBRARY(char *name, char *table) : Object(name)
{

    SB_COMPONENT_DICTIONARY *new_component_dictionary=
        new SB_COMPONENT_DICTIONARY();

    the_component_dictionary=
        new_component_dictionary→findTRef();

    the_recognized_types=NULL;

    update_recognized_types(table);

    SB_OPERATOR_COMPONENT_LIBRARY *new_operator_component_library=
        new SB_OPERATOR_COMPONENT_LIBRARY();

    the_operator_component_library=
        new_operator_component_library→findTRef();

    SB_ADT_COMPONENT_LIBRARY *new_adt_component_library=new
        SB_ADT_COMPONENT_LIBRARY();

    the_adt_component_library=new_adt_component_library→findTRef();

    SB_KEYWORD_LIBRARY *new_keyword_library=new
        SB_KEYWORD_LIBRARY();

    the_keyword_library=new_keyword_library→findTRef();

};

SB_RECOGNIZED_TYPES *SB_LIBRARY::recognized_types()
{
    return (SB_RECOGNIZED_TYPES *) (Entity *) the_recognized_types→Binding();
};

void SB_LIBRARY::update_recognized_types(char *file)
{
    if(the_recognized_types≠NULL)
    {
        recognized_types()→deleteObject();
    };
};

```

```

    SB_RECOGNIZED_TYPES *new_type_matrix=new
        SB_RECOGNIZED_TYPES(file);
    the_recognized_types=new_type_matrix→findTRef();
}

void SB_LIBRARY::Destroy( Boolean aborted)
{
    operator_component_library()→Destroy(aborted);
    adt_component_library()→Destroy(aborted);
    component_dictionary()→Destroy(aborted);
    recognized_types()→Destroy(aborted);
    keyword_library()→Destroy(aborted);

    delete the_adt_component_library;
    delete the_operator_component_library;
    delete the_component_dictionary;
    delete the_keyword_library;
    delete the_recognized_types;

    Object::Destroy(aborted);
};

void SB_LIBRARY::deleteObject( Boolean deallocate)
{
    operator_component_library()→deleteObject(FALSE);
    adt_component_library()→deleteObject(FALSE);
    component_dictionary()→deleteObject(FALSE);
    recognized_types()→deleteObject(FALSE);
    keyword_library()→deleteObject(FALSE);

    Object::deleteObject(deallocate);
};

void SB_LIBRARY::putObject( Boolean deallocate)
{
    operator_component_library()→putObject(deallocate);
    adt_component_library()→putObject(deallocate);
    component_dictionary()→Dictionary::putObject(deallocate);
    recognized_types()→putObject(deallocate);
    keyword_library()→putObject(deallocate);

    Object::putObject(deallocate);
};

Type *SB_LIBRARY::getDirectType()
{
    return SB_LIBRARY_OType;
};

```

```

SB_ADT_COMPONENT_LIBRARY *SB_LIBRARY::adt_component_library()
{
    return (SB_ADT_COMPONENT_LIBRARY *)
        (the_adt_component_library→Binding());
};

SB_OPERATOR_COMPONENT_LIBRARY *SB_LIBRARY::operator_component_library()
{
    return (SB_OPERATOR_COMPONENT_LIBRARY *)
        (the_operator_component_library→Binding());
};

SB_KEYWORD_LIBRARY *SB_LIBRARY::keyword_library()
{
    return (SB_KEYWORD_LIBRARY *)the_keyword_library→Binding();
};

Boolean SB_LIBRARY::add(SB_COMPONENT *new_component)
{
    Boolean return_flag=FALSE;

    // first ensure that this component name is not already in use

    if(component_dictionary()→add(new_component)==TRUE)
    {
        component_dictionary()→Dictionary::putObject();

        // name not in use so continue processing

        // add the component to the keyword libraries

        return_flag=TRUE;

        keyword_library()→add_component(new_component);
        keyword_library()→putObject();

        if(new_component→getDirectType()==SB_ADT_COMPONENT_OType)
        {
            return_flag=adt_component_library()→
                add((SB_ADT_COMPONENT *)new_component);
        }
        else
        {
            return_flag=operator_component_library()→
                add((SB_OPERATOR_COMPONENT *)new_component);
        }
    };

};

return return_flag;

```

```

};

void SB_LIBRARY::delete_component(SB_COMPONENT *the_component)
{
    keyword_library()→delete_component(the_component);
    keyword_library()→putObject();

    if(the_component→getDirectType()==SB_ADT_COMPONENT_OType)
    {
        adt_component_library()→
            delete_component((SB_ADT_COMPONENT *)the_component);
    }
    else
    {
        operator_component_library()→
            delete_component((SB_OPERATOR_COMPONENT *)the_component);
    };
    component_dictionary()→Remove(the_component→component_name());
    component_dictionary()→putObject();
    the_component→deleteObject(TRUE);
};

void SB_LIBRARY::component_list(ofstream& outstream)
{
    adt_component_library()→list(outstream);
    operator_component_library()→list(outstream);
};

SB_COMPONENT_DICTIONARY *SB_LIBRARY::query(SB_COMPONENT
*query_component)
{
    SB_COMPONENT_DICTIONARY *return_dictionary;

    if(query_component→getDirectType()==SB_ADT_COMPONENT_OType)
    {
        return_dictionary=adt_component_library()→
            query((SB_ADT_COMPONENT *)query_component);
    }
    else
    {
        return_dictionary=operator_component_library()→
            query((SB_OPERATOR_COMPONENT *)query_component);
    };

    return return_dictionary;
};

SB_COMPONENT *SB_LIBRARY::query(char *component_name)

```

```

{
    return component_dictionary()→query(component_name);
};
void SB_LIBRARY::keyword_query(ifstream& instream,ofstream& outstream)
{
    keyword_library()→query(instream,outstream);
};

SB_COMPONENT_DICTIONARY *SB_LIBRARY::component_dictionary()
{
    return (SB_COMPONENT_DICTIONARY *)(Entity *)
        the_component_dictionary→Binding();
};

void SB_LIBRARY::keyword_list(ofstream& outstream)
{
    keyword_library()→list(outstream);
};

void SB_LIBRARY::type_list(ofstream& outstream)
{
    adt_component_library()→list(outstream);
};

void SB_LIBRARY::operator_list(ofstream& outstream)
{
    operator_component_library()→list(outstream);
};

void SB_LIBRARY::query(SB_COMPONENT *query_component,ofstream& outstream)
{
    SB_COMPONENT_DICTIONARY* the_result=query(query_component);
    the_result→printOn(outstream);
};

```

```

# include "sball.hxx"

# include "sbextern.h"

SB_OPERATOR::SB_OPERATOR(APL *theAPL) : SB_COMPONENT(theAPL)
{
};

SB_OPERATOR::SB_OPERATOR(char *id) : SB_COMPONENT(id)
{

    SB_ID_DECL_DICTIONARY *new_input_attributes=new SB_ID_DECL_DICTIONARY();

    SB_ID_DECL_DICTIONARY *new_output_attributes=new SB_ID_DECL_DICTIONARY();

    SB_EXCEPTION_DICTIONARY *new_exceptions=new SB_EXCEPTION_DICTIONARY();

    the_input_attributes=new_input_attributes—findTRef();
    the_output_attributes=new_output_attributes—findTRef();
    the_exceptions=new_exceptions—findTRef();

    states_flag=FALSE;

}

SB_ID_DECL_DICTIONARY *SB_OPERATOR::input_attributes()
{
    return (SB_ID_DECL_DICTIONARY *) (the_input_attributes—Binding());
};

SB_ID_DECL_DICTIONARY *SB_OPERATOR::output_attributes()
{
    return (SB_ID_DECL_DICTIONARY *) (the_output_attributes—Binding());
};

Boolean SB_OPERATOR::states()
{
    return states_flag;
};

SB_EXCEPTION_DICTIONARY *SB_OPERATOR::exceptions()
{
    return (SB_EXCEPTION_DICTIONARY *) (the_exceptions—Binding());
};

```

```

Boolean SB_OPERATOR::add_inputs(SBJD_DECL_DICTIONARY *input_dictionary)
{
    Boolean return_flag;

    // add new declarations to the declaration list
    return_flag=input_attributes()—append(input_dictionary);

    return return_flag;
};

Boolean SB_OPERATOR::add_outputs(SBJD_DECL_DICTIONARY *output_dictionary)
{
    Boolean return_flag;

    return_flag=output_attributes()—append(output_dictionary);

    return return_flag;
};

void SB_OPERATOR::set_states()
{
    states_flag=TRUE;
};

Boolean SB_OPERATOR::add_exceptions(SB_EXCEPTION_DICTIONARY
*exception_dictionary)
{
    Boolean return_flag;

    return_flag=exceptions()—append(exception_dictionary);

    return return_flag;
};

void SB_OPERATOR::Destroy(Boolean aborted)
{
    input_attributes()—Destroy(aborted);

    output_attributes()—Destroy(aborted);

    exceptions()—Destroy(aborted);

    delete the_input_attributes;
    delete the_output_attributes;
    delete the_exceptions;

    SB_COMPONENT::Destroy(aborted);
}

```

```

};

void SB_OPERATOR::deleteObject( Boolean deallocate)
{
    input_attributes()—deleteObject(FALSE);

    output_attributes()—deleteObject(FALSE);

    exceptions()—deleteObject(FALSE);

    SB_COMPONENT::deleteObject(deallocate);
};

void SB_OPERATOR::putObject( Boolean deallocate)
{
    input_attributes()—putObject(deallocate);

    output_attributes()—putObject(deallocate);

    exceptions()—putObject(deallocate);
    SB_COMPONENT::putObject(deallocate);
};

void SB_OPERATOR::printOn(ofstream& outstream)
{
    outstream << "OUTPUTING INTERFACE FOR OPERATOR ";
    outstream << this—component_name() << "\n";

    outstream << "GENERIC ATTRIBUTES\n\n";
    SB_COMPONENT::generic_usage()—printOn(outstream);

    outstream << "\nINPUT ATTRIBUTES\n\n";
    input_attributes()—printOn(outstream);

    outstream << "\nOUTPUT ATTRIBUTES\n\n";
    output_attributes()—printOn(outstream);

    outstream << "\nUNRECOGNIZED TYPES\n\n";
    unrecognized_type_usage()—printOn(outstream);

    outstream << "\nRECOGNIZED TYPES\n\n";
    recognized_type_usage()—printOn(outstream);

    outstream << "\nSTATES\n\n";
    if(s'tates_flag==TRUE)
    {
        outstream << "YES\n";
    }
}

```

```

else
{
    ostream << "NO\n";
};

ostream << "\nEXCEPTIONS\n\n";
exceptions()→printOn(ostream);
ostream << "OUTPUTING THE PSDL TEXT\n\n";
ostream << psdl.text()→text() << "\n";
ostream << "OUTPUTING THE INFORMAL DESCRIPTION\n\n";
ostream << informal.description()→text() << "\n";
ostream << "OUTPUTING THE FORMAL DESCRIPTION\n\n";
ostream << formal.description()→text() << "\n";
ostream << "OUTPUTING THE NORMALIZED FORMAL DESCRIPTION\n\n";
ostream << norm.formal.description()→text() << "\n";
ostream << "OUTPUTING THE ADA SPEC\n\n";
ostream << imp.spec.text()→text() << "\n";
ostream << "OUTPUTING THE ADA BODY\n\n";
ostream << imp.body.text()→text() << "\n";

};

int SB_OPERATOR::num_inputs()
{
    return input_attributes()→num();
};

int SB_OPERATOR::num_outputs()
{
    return output_attributes()→num();
};

DictionaryIterator SB_OPERATOR::input_iterator()
{
    return input_attributes()→type_iterator();
};

DictionaryIterator SB_OPERATOR::output_iterator()
{
    return output_attributes()→type_iterator();
};

DictionaryIterator SB_OPERATOR::exception_iterator()
{
    return exceptions()→iterator();
};

```

```

int SB.OPERATOR::num_generic_types()
{
    // get an iterator for the type_decl list
    // and any spec that is not a procedure is a generic type

    int count=0;

    DictionaryIterator next_id=SB_COMPONENT::generic_usage()→
        type_id_iterator();

    while(next_id.moreData())
    {
        SB_TYPE_USAGE *the_usage=(SB_TYPE_USAGE*)(Entity *)next_id();
        SB_TYPE_NAME *the_type_name=the_usage→type_name();
        if(the_type_name→type_code()==SB_GENERIC_TYPE)
        {
            count=count++;
        };
    };
    return count;
};

```



```

        if(this_type_name→recognized()==FALSE)
        {
            // was unrecognized so try to update
            // the unrecognized list or add it to
            // the list
            if(unrecognized_type_usage()→
                update(this_type_name)==FALSE)
            {
                // not yet in list so add it
                unrecognized_type_usage()→
                    add_type(this_type_name→id(),
                        this_type_name);
                // now update it for being used once
                unrecognized_type_usage()→
                    update(this_type_name);
            };
        }
        else
        {
            // this type name is recognized so update
            // or add it
            if(recognized_type_usage()→
                update(this_type_name)==FALSE)
            {
                // not yet in list so add it
                recognized_type_usage()→
                    add_type(this_type_name→id(),
                        this_type_name);
                // now update it for being used once
                recognized_type_usage()→
                    update(this_type_name);
            };
        };
    };
};
DictionaryIterator next_output=output_attributes()→id_iterator();
while(next_output.moreData())
{
    SB_ID_DECL *this_decl=(SB_ID_DECL *)(Entity *)next_output();
    SB_TYPE_NAME *this_type_name=this_decl→type_name();

    // first see if this id_decl type is a generic
    if(generic_usage()→update(this_type_name)==FALSE)
    {
        // was not a generic type so put it in the usage list
        // based on whether or not it is recognized
        if(this_type_name→recognized()==FALSE)
        {
            // was unrecognized so try to update
            // the unrecognized list or add it to
            // the list

```

```

        if(unrecognized_type_usage()→
            update(this_type_name)==FALSE)
        {
            // not yet in list so add it
            unrecognized_type_usage()→
                add_type(this_type_name→id(),
                    this_type_name);
            // now update it for being used once
            unrecognized_type_usage()→
                update(this_type_name);
        };
    }
else
    {
        // this type name is recognized so update
        // or add it
        if(recognized_type_usage()→
            update(this_type_name)==FALSE)
        {
            // not yet in list so add it
            recognized_type_usage()→
                add_type(this_type_name→id(),
                    this_type_name);
            // now update for being used once
            recognized_type_usage()→
                update(this_type_name);
        };
    };
};

};
return TRUE;

};

Boolean SB_OPERATOR_COMPONENT::filter(SB_OPERATOR_COMPONENT *library_unit)
{
    // apply additional filter operations to the library unit
    // to see if the component can be rejected. True means
    // that it may still be a match. False indicates no match

    return TRUE;
};

```

```

# include "sball.hxx"

# include "sbextern.h"

SB_OPERATOR_COMPONENT_LIBRARY::SB_OPERATOR_COMPONENT_LIBRARY(APL
*theAPL) :
Object(theAPL)
{
};

SB_OPERATOR_COMPONENT_LIBRARY::SB_OPERATOR_COMPONENT_LIBRARY() :
Object()
{

    SB_COMPONENT_DICTIONARY *new_operator_component_dictionary=
        new SB_COMPONENT_DICTIONARY();

    the_operator_component_dictionary=
        new_operator_component_dictionary->findTRef();

    Dictionary *new_state_dictionary=new Dictionary(OC_integer,
                                                    OC_dictionary,
                                                    TRUE,
                                                    FALSE);

    the_state_dictionary=new_state_dictionary->findTRef();

    Dictionary *new_non_state_dictionary=new Dictionary(OC_integer,
                                                        OC_dictionary,
                                                        TRUE,
                                                        FALSE);

    the_non_state_dictionary=new_non_state_dictionary->findTRef();

};

void SB_OPERATOR_COMPONENT_LIBRARY::Destroy(Boolean aborted)
{
    operator_component_dictionary()->Destroy(aborted);

    // now must iterate through the multi-attribute query
    // dictionary tree

    Dictionary *leaf_dictionary;
    Dictionary *by_num_inputs_dictionary;
    Dictionary *by_num_unrecognized_dictionary;
    Dictionary *by_num_outputs_dictionary;

    by_num_inputs_dictionary=state_dictionary();

```

```

DictionaryIterator next_input_dictionary=
    DictionaryIterator(by_num_inputs_dictionary);

while(next_input_dictionary.moreData())
{
    by_num_unrecognized_dictionary=
        (Dictionary *)(Entity *)next_input_dictionary();

    DictionaryIterator next_outputs_dict(by_num_unrecognized_dictionary);

    while(next_outputs_dict.moreData())
    {
        by_num_outputs_dictionary=(Dictionary *)(Entity *)
            next_outputs_dict();

        DictionaryIterator next_leaf_dict(by_num_outputs_dictionary);

        while(next_leaf_dict.moreData())
        {
            leaf_dictionary=(Dictionary *)(Entity *)
                next_leaf_dict();

            DictionaryIterator next_component(leaf_dictionary);

            while(next_component.moreData())
            {
                ((SB_OPERATOR_COMPONENT *)(Entity *)next_component())→
                    Destroy(aborted);
            };
            leaf_dictionary→Destroy(aborted);
        };
        by_num_outputs_dictionary→Destroy(aborted);
    };
    by_num_unrecognized_dictionary→Destroy(aborted);
};
by_num_inputs_dictionary→Destroy(aborted);

by_num_inputs_dictionary=non_state_dictionary();

next_input_dictionary=
    DictionaryIterator(by_num_inputs_dictionary);

while(next_input_dictionary.moreData())
{
    by_num_unrecognized_dictionary=
        (Dictionary *)(Entity *)next_input_dictionary();

```

```

DictionaryIterator next_outputs_dict(by_num_unrecognized_dictionary);
while(next_outputs_dict.moreData())
{
    by_num_outputs_dictionary=(Dictionary *)(Entity *)
        next_outputs_dict();

    DictionaryIterator next_leaf_dict(by_num_outputs_dictionary);

    while(next_leaf_dict.moreData())
    {
        leaf_dictionary=(Dictionary *)(Entity *)
            next_leaf_dict();

        DictionaryIterator next_component(leaf_dictionary);

        while(next_component.moreData())
        {
            ((SB_OPERATOR_COMPONENT *)(Entity *)next_component())—
                Destroy(aborted);
        };
        leaf_dictionary—Destroy(aborted);
    };
    by_num_outputs_dictionary—Destroy(aborted);
};
by_num_unrecognized_dictionary—Destroy(aborted);
};
by_num_inputs_dictionary—Destroy(aborted);

delete the_operator_component_dictionary;
delete the_state_dictionary;
delete the_non_state_dictionary;

};

void SB_OPERATOR_COMPONENT_LIBRARY::deleteObject(Boolean deallocate)
{
    operator_component_dictionary()—deleteObject(deallocate);

    Dictionary *leaf_dictionary;
    Dictionary *by_num_inputs_dictionary;
    Dictionary *by_num_unrecognized_dictionary;
    Dictionary *by_num_outputs_dictionary;

    by_num_inputs_dictionary=state_dictionary();

    DictionaryIterator next_input_dictionary(by_num_inputs_dictionary);

```

```

while(next_input_dictionary.moreData())
{
    by_num_unrecognized_dictionary=
        (Dictionary *) (Entity *) next_input_dictionary();

    DictionaryIterator next_outputs_dict(by_num_unrecognized_dictionary);

    while(next_outputs_dict.moreData())
    {
        by_num_outputs_dictionary=(Dictionary *) (Entity *)
            next_outputs_dict();

        DictionaryIterator next_leaf_dict(by_num_outputs_dictionary);

        while(next_leaf_dict.moreData())
        {
            leaf_dictionary=(Dictionary *) (Entity *)
                next_leaf_dict();

            DictionaryIterator next_component(leaf_dictionary);

            while(next_component.moreData())
            {
                ((SB_OPERATOR_COMPONENT *) (Entity *) next_component())—
                    deleteObject(FALSE);
            };
            leaf_dictionary—deleteObject(FALSE);
        };
        by_num_outputs_dictionary—deleteObject(FALSE);
    };
    by_num_unrecognized_dictionary—deleteObject(FALSE);
};
by_num_inputs_dictionary—deleteObject(FALSE);

by_num_inputs_dictionary=non_state_dictionary();

next_input_dictionary=
    DictionaryIterator(by_num_inputs_dictionary);

while(next_input_dictionary.moreData())
{
    by_num_unrecognized_dictionary=
        (Dictionary *) (Entity *) next_input_dictionary();

    DictionaryIterator next_outputs_dict(by_num_unrecognized_dictionary);
    while(next_outputs_dict.moreData())

```

```

    {
        by_num_outputs_dictionary=(Dictionary *)(Entity *)
            next_outputs_dict();

        DictionaryIterator next_leaf_dict(by_num_outputs_dictionary);

        while(next_leaf_dict.moreData())
        {
            leaf_dictionary=(Dictionary *)(Entity *)
                next_leaf_dict();

            DictionaryIterator next_component(leaf_dictionary);

            while(next_component.moreData())
            {
                ((SB_OPERATOR_COMPONENT *)(Entity *)next_component())→
                    deleteObject(FALSE);
            };
            leaf_dictionary→deleteObject(FALSE);
        };
        by_num_outputs_dictionary→deleteObject(FALSE);
    };
    by_num_unrecognized_dictionary→deleteObject(FALSE);
};
by_num_inputs_dictionary→deleteObject(FALSE);

Object::deleteObject(deallocate);

};

void SB_OPERATOR_COMPONENT_LIBRARY::putObject(Boolean deallocate)
{
    operator_component_dictionary()→putObject(deallocate);

    state_dictionary()→putObject(deallocate);
    non_state_dictionary()→putObject(deallocate);
    Object::putObject(deallocate);

};

Type *SB_OPERATOR_COMPONENT_LIBRARY::getDirectType()
{
    return SB_OPERATOR_COMPONENT_LIBRARY_OType;
};

SB_COMPONENT_DICTIONARY
*SB_OPERATOR_COMPONENT_LIBRARY::operator_component_dictionary()
{

```

```

    return (SB.COMPONENT.DICTIONARY *)(Entity *)
        the_operator_component_dictionary→Binding();
};

Dictionary *SB.OPERATOR_COMPONENT_LIBRARY::state_dictionary()
{
    return (Dictionary *)(Entity *)the_state_dictionary→Binding();
};

Dictionary *SB.OPERATOR_COMPONENT_LIBRARY::non_state_dictionary()
{
    return (Dictionary *)(Entity *)the_non_state_dictionary→Binding();
};

Boolean SB.OPERATOR_COMPONENT_LIBRARY::add(SB.OPERATOR_COMPONENT
*new_component)
{
    Boolean return_flag=TRUE;
    Dictionary *leaf_dictionary;
    Dictionary *by_num_inputs_dictionary;
    Dictionary *by_num_unrecognized_dictionary;
    Dictionary *by_num_outputs_dictionary;

    operator_component_dictionary()→add(new_component);

    operator_component_dictionary()→Dictionary::putObject();

    // insert into the component dictionary was successfull
    // so insert it into the library

    if(new_component→states()==TRUE)
    {
        by_num_inputs_dictionary=state_dictionary();
    }
    else
    {
        by_num_inputs_dictionary=non_state_dictionary();
    };

    // have correct state dictionary so now find correct
    // input dictionary

    if(by_num_inputs_dictionary→isIndex(new_component→num_inputs())==TRUE)
    {
        by_num_unrecognized_dictionary=(Dictionary *)
            (Entity *)(*by_num_inputs_dictionary)
                [new_component→num_inputs()];
    }
    else
    {

```

```

by_num_unrecognized_dictionary=new Dictionary(OC_integer,
                                              OC_dictionary,
                                              TRUE,
                                              FALSE);

by_num_inputs_dictionary→Insert(new_component→
                                num_inputs(),
                                by_num_unrecognized_dictionary);

};

// have correct by num inputs dictionary so get the
// unrecognized types dict.

// got the unrecognized dictionary
// use num generics since for a library unit all unrecognized types
// must be generics

if(by_num_unrecognized_dictionary→
  isIndex(new_component→num_generic_types())==TRUE)
{
  by_num_outputs_dictionary=(Dictionary *)(Entity *)
    (*by_num_unrecognized_dictionary)
    [new_component→num_generic_types()];
}
else
{
  by_num_outputs_dictionary=new Dictionary(OC_integer,
                                          OC_dictionary,
                                          TRUE,
                                          FALSE);

  by_num_unrecognized_dictionary→Insert(new_component→
                                         num_generic_types(),
                                         by_num_outputs_dictionary);

};

if(by_num_outputs_dictionary→
  isIndex(new_component→num_outputs())≈=TRUE)
{
  leaf_dictionary=(Dictionary *)(Entity *)
    (*by_num_outputs_dictionary)
    [new_component→num_outputs()];
}

```

```

else
{
    leaf_dictionary=new Dictionary(OC_string,
                                   SB.COMPONENT.OType,
                                   FALSE,
                                   FALSE);

    by_num_outputs_dictionary→Insert(new_component→
                                     num_outputs(),
                                     leaf_dictionary);

};

// have to leaf dictionary so now insert the component into it
leaf_dictionary→Insert(new_component→component_name(),
                       new_component);

leaf_dictionary→putObject();
by_num_inputs_dictionary→putObject();
by_num_unrecognized_dictionary→putObject();
by_num_outputs_dictionary→putObject();

return return_flag;
};

void SB_OPERATOR_COMPONENT_LIBRARY::
delete_component(SB_OPERATOR_COMPONENT *the_component)
{
    Dictionary *leaf_dictionary;
    Dictionary *by_num_inputs_dictionary;
    Dictionary *by_num_unrecognized_dictionary;
    Dictionary *by_num_outputs_dictionary;

    operator_component_dictionary→
        Remove(the_component→component_name());
    operator_component_dictionary→putObject();

    if(the_component→states()==TRUE)
    {
        by_num_inputs_dictionary=state_dictionary();
    }
    else
    {
        by_num_inputs_dictionary=non_state_dictionary();
    }
};

```

```

// have correct state dictionary so now find correct
// input dictionary

if(by_num_inputs_dictionary→isIndex(the_component→num_inputs())==TRUE)
{
    by_num_unrecognized_dictionary=(Dictionary *)
        (Entity *)(*by_num_inputs_dictionary)
        [the_component→num_inputs()];

    // got the unrecognized dictionary
    // use num generics since for a library unit all unrecognized types
    // must be generics

    if(by_num_unrecognized_dictionary→
        isIndex(the_component→num_generic_types())==TRUE)
    {

        by_num_outputs_dictionary=(Dictionary *) (Entity *)
            (*by_num_unrecognized_dictionary)
            [the_component→num_generic_types()];

        if(by_num_outputs_dictionary→
            isIndex(the_component→num_outputs())==TRUE)
        {
            leaf_dictionary=(Dictionary *) (Entity *)
                (*by_num_outputs_dictionary)
                [the_component→num_outputs()];

            // have to leaf dictionary
            leaf_dictionary→Remove(the_component→component_name());
            leaf_dictionary→putObject();
            if(leaf_dictionary→Cardinality()==0)
            {
                by_num_outputs_dictionary→
                    Remove(the_component→num_outputs());
                by_num_outputs_dictionary→putObject();

                if(by_num_outputs_dictionary→Cardinality()==0);
                {
                    by_num_unrecognized_dictionary→
                        Remove(the_component→num_generic_types());
                    by_num_unrecognized_dictionary→putObject();

                    if(by_num_unrecognized_dictionary→Cardinality()==0)
                    {
                        by_num_inputs_dictionary→
                            Remove(the_component→num_inputs());
                        by_num_inputs_dictionary→putObject();
                        by_num_unrecognized_dictionary→deleteObject(TRUE);
                    };
                    by_num_outputs_dictionary→deleteObject(TRUE);
                }
            }
        }
    }
}

```

```

SB_COMPONENT_DICTIONARY *SB_OPERATOR_COMPONENT_LIBRARY::
query(SB_OPERATOR_COMPONENT *query_component)
{
    SB_COMPONENT_DICTIONARY *query_result=new SB_COMPONENT_DICTIONARY();

    Dictionary *leaf_dictionary;
    Dictionary *by_num_inputs_dictionary;
    Dictionary *by_num_unrecognized_dictionary;
    Dictionary *by_num_outputs_dictionary;

    // get the correct state_dictionary to start the query

    if(query_component->states()==TRUE)
    {
        by_num_inputs_dictionary=state_dictionary();
    }
    else
    {
        by_num_inputs_dictionary=non_state_dictionary();
    };

    // have correct state dictionary so now find correct
    // input dictionary

    // for inputs must match exactly so only get one dictionary

    if((by_num_inputs_dictionary->isIndex(query_component->num_inputs()))==TRUE)
    {
        by_num_unrecognized_dictionary=
            (Dictionary *) (Entity *) (*by_num_inputs_dictionary
                [query_component->num_inputs()]);
        // got the correct unrecognized_dictionary so iterate over it for
        // the output dictionaries
        DictionaryIterator next_outputs_dict=
            DictionaryIterator(by_num_unrecognized_dictionary,
                FALSE,
                query_component->
                num_unrecognized_types());

        // loop through all of the unrecognized types that are valid

        while(next_outputs_dict.moreData())

```

```

{
    by_num_outputs_dictionary=(Dictionary *)(Entity *)
        next_outputs_dict();

    // got the outputs dictionary so now get the leaf dictionary

    DictionaryIterator next_leaf_dict=
        DictionaryIterator(by_num_outputs_dictionary,
            FALSE,
            query_component—
            num_outputs());

    while(next_leaf_dict.moreData())
    {
        leaf_dictionary=(Dictionary *)(Entity *)
            next_leaf_dict();

        DictionaryIterator next_component=
            DictionaryIterator(leaf_dictionary);

        // got an output dictionary so iterate
        // through it and put the components in the
        // return result dictionary

        while(next_component.moreData())
        {
            SB_OPERATOR_COMPONENT *the_component=
                (SB_OPERATOR_COMPONENT *)(Entity *)next_component();

            if(query_component—filter(the_component)==TRUE)
            {
                query_result—add(the_component);
            }
        }
    }
};

// add code to interface to semantic check routine here

return query_result;

};

void SB_OPERATOR_COMPONENT_LIBRARY::list(ofstream& outstream)
{

```

```
operator_component_dictionary()→printOn(outstream);  
};
```

```

#include "sball.hxx"

#include "sbextern.h"

SB_RECOGNIZED_TYPES::SB_RECOGNIZED_TYPES(char *file) : Object()
{
    char new_type_name[256];
    int count=0;
    int flag_value;
    int case_sensitive_int;

    Dictionary *new_name_dictionary=new
        Dictionary(OC_string,OC_integer,FALSE,FALSE);

    the_name_dictionary=new_name_dictionary->findTRef();

    ifstream type_defs(file);

    type_defs >> case_sensitive_int;
    if(case_sensitive_int==0)
    {
        case_sensitive=FALSE;
    }
    else
    {
        case_sensitive=TRUE;
    };

    Boolean done_flag=FALSE;
    while(done_flag==FALSE)
    {

        count++;
        type_defs >> new_type_name;
        if(strcmp(new_type_name,"~")!=0)
        {
            char *store_type_name;
            if(!case_sensitive)
            {
                store_type_name=convert_to_upper(new_type_name);
            }
            else
            {
                store_type_name=new_type_name;
            };

            name_dictionary()->Insert(store_type_name,count);

        }
        else
        {

```

```

        done_flag=TRUE;
        count--;
    };

};

array_size=count;

Array *new_row_array=new Array(OC_array,count,1);
the_row_array=new_row_array→findTRef();

int row_count=1;

for(row_count=1;row_count < array_size+1; row_count++)
{
    Array *new_column_array=new Array(OC_integer,count,1);
    int i;
    for(i=1;(i < count+1);i++)
    {
        type_defs >> flag_value;
        new_column_array→setElement(i,flag_value);
    }
    row_array()→setElement(row_count,new_column_array);
};

};

SB_RECOGNIZED_TYPES::SB_RECOGNIZED_TYPES(APL *theAPL) : Object(theAPL)
{
}

void SB_RECOGNIZED_TYPES::Destroy(Boolean aborted)
{
    int row_count;

    name_dictionary()→Destroy(aborted);

    for(row_count=1;row_count < array_size+1; row_count++)
    {
        Array &row=*(row_array());
        ((Array *) (Entity *) row[row_count])→Destroy(aborted);
    };
    row_array()→Destroy(aborted);

    delete the_name_dictionary;
    delete the_row_array;

    Object::Destroy(aborted);
};

```

```

void SB_RECOGNIZED_TYPES::deleteObject(Boolean deallocate)
{
    int row_count;

    name_dictionary()→deleteObject(FALSE);

    for(row_count=1;row_count < array_size+1; row_count++)
    {
        Array &row=*(row_array());
        ((Array *)(Entity *)row[row_count])→deleteObject(FALSE);
    };
    row_array()→deleteObject(FALSE);

    Object::deleteObject(deallocate);
};

void SB_RECOGNIZED_TYPES::putObject(Boolean deallocate)
{
    int row_count;

    name_dictionary()→putObject(deallocate);

    for(row_count=1;row_count < array_size+1; row_count++)
    {
        Array &row=*(row_array());
        ((Array *)(Entity *)row[row_count])→putObject(deallocate);
    };
    row_array()→putObject(deallocate);

    Object::putObject(deallocate);
};

Type *SB_RECOGNIZED_TYPES::getDirectType()
{
    return SB_RECOGNIZED_TYPES_OType;
};

Array *SB_RECOGNIZED_TYPES::row_array()
{
    return (Array *)(Entity *)the_row_array→Binding();
};

Dictionary *SB_RECOGNIZED_TYPES::name_dictionary()
{
    return (Dictionary *)(Entity *)the_name_dictionary→Binding();
};

Boolean SB_RECOGNIZED_TYPES::map(int in_map,int out_map)
{
    Boolean return_flag=FALSE;

    Array& rows=*(row_array());

```

```

Array& column=((Array *)(Entity *)rows[in_map]);

if( int(column[out_map])==1)
{
    return_flag=TRUE;
};
return return_flag;
};

int SB_RECOGNIZED_TYPES::type_number(char *type_id)
{
    int return_value=SB_UNRECOGNIZED_TYPE;
    char *search_id;

    if(!case_sensitive)
    {
        search_id=convert_to_upper(type_id);
    }
    else
    {
        search_id=type_id;
    };

    if(name_dictionary()—isIndex(search_id))
    {
        return_value=int(name_dictionary()—getIntegerElement(search_id));
    };

    return return_value;
}

char *SB_RECOGNIZED_TYPES::convert_to_upper(char *type_id)
{
    int i;
    char *new_id=new char[strlen(type_id)+1];
    for(i=0;i<strlen(type_id);i++)
    {
        if(islower(type_id[i]))
        {
            new_id[i]=type_id[i]+'A'-'a';
        }
        else
        {
            new_id[i]=type_id[i];
        };
    };
    new_id[i]=NULL;
    return new_id;
}

```

};

```

# include "sball.hxx"

# include "sbextern.h"

SB_TYPE_NAME::SB_TYPE_NAME(APL *theAPL) : Object(theAPL)
{
};

SB_TYPE_NAME::SB_TYPE_NAME(char *new_id, SB_ID_DECL_DICTIONARY
*new_dictionary) : Object()
{

    the_id=new char[strlen(new_id)+1];
    strcpy(the_id,new_id);

    if(new_dictionary==NULL)
    {
        new_dictionary= new SB_ID_DECL_DICTIONARY();
    };
    the_id_decl_dictionary=new_dictionary->findTRef();

    Boolean found_flag=FALSE;
    SB_ID_DECL *base_type_decl;
    DictionaryIterator next_id_decl=id_decl_dictionary()->id_iterator();
    while(next_id_decl.moreData() && !found_flag)
    {

        SB_ID_DECL *the_type_decl=(SB_ID_DECL *) (Entity *) next_id_decl();
        if((SB_MAIN_LIBRARY->recognized_types())->
            type_number(the_type_decl->id())==SB_BASE_TYPE)
        {
            // found the base type decl
            found_flag=TRUE;
            base_type_decl=the_type_decl;
        };
    };
    if(found_flag)
    {
        // has a base type defined so look it up
        the_base_type_id=(base_type_decl->type_name())->id();
    }
    else
    {
        the_base_type_id=the_id;
    };
    the_base_type_code=(SB_MAIN_LIBRARY->
        recognized_types())->type_number(the_base_type_id);
    the_type_code=(SB_MAIN_LIBRARY->
        recognized_types())->type_number(the_id);
};

```

```

void SB_TYPE_NAME::Destroy(Boolean aborted)
{
    if(the_id≠NULL)
    {
        delete the_id;
    };

    if(the_base_type_id)
    {
        delete the_base_type_id;
    };

    id_decl_dictionary()→Destroy(aborted);

    delete the_id_decl_dictionary;

    Object::Destroy(aborted);
};

void SB_TYPE_NAME::deleteObject(Boolean deallocate)
{
    id_decl_dictionary()→deleteObject(FALSE);

    Object::deleteObject(deallocate);
};

void SB_TYPE_NAME::putObject(Boolean deallocate)
{
    id_decl_dictionary()→putObject(deallocate);

    Object::putObject(deallocate);
};

SB_ID_DECL_DICTIONARY *SB_TYPE_NAME::id_decl_dictionary()
{
    return (SB_ID_DECL_DICTIONARY *)(the_id_decl_dictionary→Binding());
};

Type *SB_TYPE_NAME::getDirectType()
{
    return SB_TYPE_NAME_OType;
};

Boolean SB_TYPE_NAME::operator ==(Entity& other_type)
{
    Boolean return_flag=FALSE;

```

```

char *this_base_id=this→base_type_id();
char *this_id=this→id();
char *other_base_id=((SB_TYPE_NAME &)other_type).base_type_id();
char *other_id=((SB_TYPE_NAME &)other_type).id();

if(strcmp(this_base_id,other_base_id)==0 &&
   (strcmp(this_id,other_id)==0))
{
    return_flag=TRUE;
};
return return_flag;
};

```

```

Boolean SB_TYPE_NAME::operator ≥(Entity& other_type)
{

```

```

    Boolean return_flag=FALSE;
    char *this_base_id=this→base_type_id();
    char *this_id=this→id();
    char *other_base_id=((SB_TYPE_NAME &)other_type).base_type_id();
    char *other_id=((SB_TYPE_NAME &)other_type).id();

    if( strcmp(this_base_id,other_base_id) < 0)
    {
        return_flag=FALSE;
    }
    else if(strcmp(this_base_id,other_base_id) > 0)
    {
        return_flag=TRUE;
    }
    else if(strcmp(this_id,other_id)< 0)
    {
        return_flag=FALSE;
    }
    else
    {
        return_flag=TRUE;
    };
    return return_flag;
};

```

```

void SB_TYPE_NAME::printOn(ofstream& outstream)
{
    outstream << this→the_id;
    outstream << " = " << type_code();
    outstream << " ( " << the_base_type_id << " = ";
    outstream << base_type_code() << " )";

    outstream << "[ ";

```

```

    id_decl_dictionary()→printOn(outstream);
    outstream << "]",

};

ListIterator SB_TYPE_NAME::decl_iterator()
{
    return id_decl_dictionary()→order_iterator();
};

int SB_TYPE_NAME::num_decl()
{
    int return_value;

    if(id_decl_dictionary()==NULL)
    {
        return_value=0;
    }
    else
    {
        return_value=id_decl_dictionary()→num();
    }
};

    return return_value;
};

char *SB_TYPE_NAME::id()
{
    return the_id;
};

char *SB_TYPE_NAME::base_type_id()
{
    return the_base_type_id;
};

int SB_TYPE_NAME::base_type_code()
{
    return the_base_type_code;
};

int SB_TYPE_NAME::type_code()
{
    return the_type_code;
};

Boolean SB_TYPE_NAME::recognized()
{
    return Boolean(base_type_code()!=SB_UNRECOGNIZED_TYPE);
};

```

};

```

# include "sball.hxx"
# include "sbextern.h"

SB_TEXT_OBJECT::SB_TEXT_OBJECT(APL *theAPL) : Object(theAPL)
{
};

SB_TEXT_OBJECT::SB_TEXT_OBJECT() : Object()
{
    the_text=new char[1];
    strcpy(the_text,"");
};

Type *SB_TEXT_OBJECT::getDirectType()
{
    return SB_TEXT_OBJECT_OType;
};

void SB_TEXT_OBJECT::Destroy(Booleam aborted)
{
    delete the_text;
    Object::Destroy(aborted);
};

void SB_TEXT_OBJECT::deleteObject(Booleam deallocate)
{
    Object::deleteObject(deallocate);
};

void SB_TEXT_OBJECT::putObject(Booleam deallocate)
{
    Object::putObject(deallocate);
};

void SB_TEXT_OBJECT::append(ifstream& instream)
{
    ostringstream buffer;

    while(!instream.eof())
    {
        char text=instream.get();
        if(text≠EOF)
        {
            buffer.put(text);
        }
    };
    buffer.put(NULL);
    the_text=buffer.str();
};

void SB_TEXT_OBJECT::append(char *instring)

```

```

{
    the_text=new char[strlen(instring)+1];
    strcpy(the_text,instring);
};

void SB_TEXT_OBJECT::text(ofstream& outstream)
{
    outstream << the_text;
};

char *SB_TEXT_OBJECT::text()
{
    return the_text;
};

```

```

# include "sball.hxx"

# include "sbextern.h"

SB_TYPE_USAGE::SB_TYPE_USAGE(APL *theAPL) : Object(theAPL)
{
};

SB_TYPE_USAGE::SB_TYPE_USAGE(char *new_type_id,
                             SB_TYPE_NAME *new_type_name):
    Object()
{

    the_type_id=new char[strlen(new_type_id)+1];
    strcpy(the_type_id,new_type_id);

    the_type_name=new_type_name→findTRef();
    the_times_used=0;

};

void SB_TYPE_USAGE::Destroy(Boolean aborted)
{
    if(the_type_id)
    {
        delete the_type_id;
    };

    type_name()→Destroy(aborted);

    Object::Destroy(aborted);
};

void SB_TYPE_USAGE::deleteObject(Boolean deallocate)
{
    type_name()→deleteObject(FALSE);

    Object::deleteObject(deallocate);
};

void SB_TYPE_USAGE::putObject(Boolean deallocate)
{
    type_name()→putObject(deallocate);

    Object::putObject(deallocate);
};

char *SB_TYPE_USAGE::type_id()
{

```

```

    return the_type_id;
};

SB_TYPE_NAME *SB_TYPE_USAGE::type_name()
{
    return ((SB_TYPE_NAME *) (the_type_name→Binding()));
};

void SB_TYPE_USAGE::used()
{
    the_times_used++;
};

int SB_TYPE_USAGE::times_used()
{
    return (the_times_used);
};

Type *SB_TYPE_USAGE::getDirectType()
{
    return SB_TYPE_USAGE_OType;
};

void SB_TYPE_USAGE::printOn(ofstream& outstream)
{
    outstream << the_type_id << " used " << the_times_used << " ";
    type_name()→printOn(outstream);
};

char *SB_TYPE_USAGE::base_type_id()
{
    return type_name()→base_type_id();
};

```

```

# include "sball.hxx"

# include "sbextern.h"

SB_TYPE_USAGE_DICTIONARY::SB_TYPE_USAGE_DICTIONARY(APL *theAPL) :
Object(theAPL)
{
};

SB_TYPE_USAGE_DICTIONARY::SB_TYPE_USAGE_DICTIONARY() : Object()
{

    Dictionary *new_dictionary_by_base_type= new Dictionary(OC_string,
                                                             SB_TYPE_USAGE_OType,
                                                             TRUE,
                                                             TRUE);

    the_dictionary_by_base_type=new_dictionary_by_base_type→findTRef();

    Dictionary *new_dictionary_by_type_id= new Dictionary(OC_string,
                                                           SB_TYPE_USAGE_OType,
                                                           TRUE,
                                                           FALSE);

    the_dictionary_by_type_id=new_dictionary_by_type_id→findTRef();
    Dictionary *new_dictionary_by_times_used=new Dictionary(OC_integer,
                                                             SB_TYPE_USAGE_OType,
                                                             TRUE,
                                                             TRUE);

    the_dictionary_by_times_used=new_dictionary_by_times_used→findTRef();

};

void SB_TYPE_USAGE_DICTIONARY::Destroy(Boolean aborted)
{

    DictionaryIterator next_decl=type_id_iterator();

    while(next_decl.moreData())
    {
        ((SB_TYPE_USAGE *)(Entity *)next_decl())→Destroy(aborted);
    };
};

```

```

dictionary_by_base_type()→Destroy(aborted);
dictionary_by_type_id()→Destroy(aborted);
dictionary_by_times_used()→Destroy(aborted);

delete the_dictionary_by_base_type;
delete the_dictionary_by_type_id;
delete the_dictionary_by_times_used;

Object::Destroy(aborted);

};

void SB.TYPE_USAGE.DICTIONARY::deleteObject(Boolean deallocate)
{
    DictionaryIterator next_decl=type_id_iterator();

    while(next_decl.moreData())
    {
        ((SB.TYPE_USAGE *)(Entity *)next_decl())→deleteObject(FALSE);
    };

    dictionary_by_base_type()→deleteObject(FALSE);
    dictionary_by_type_id()→deleteObject(FALSE);
    dictionary_by_times_used()→deleteObject(FALSE);

    Object::deleteObject(deallocate);

};

void SB.TYPE_USAGE.DICTIONARY::putObject(Boolean deallocate)
{
    DictionaryIterator next_decl=type_id_iterator();

    while(next_decl.moreData())
    {
        ((SB.TYPE_USAGE *)(Entity *)next_decl())→putObject(deallocate);
    };

    dictionary_by_base_type()→putObject(deallocate);
    dictionary_by_type_id()→putObject(deallocate);
    dictionary_by_times_used()→putObject(deallocate);

    Object::putObject(deallocate);

};

Type *SB.TYPE_USAGE.DICTIONARY::getDirectType()

```

```

{
    return SB_TYPE_USAGE_DICTIONARY_OType;
};

Dictionary *SB_TYPE_USAGE_DICTIONARY::dictionary_by_base_type()
{
    return (Dictionary *) (the_dictionary_by_base_type → Binding());
};

Dictionary *SB_TYPE_USAGE_DICTIONARY::dictionary_by_type_id()
{
    return (Dictionary *) (the_dictionary_by_type_id → Binding());
};

Dictionary *SB_TYPE_USAGE_DICTIONARY::dictionary_by_times_used()
{
    return (Dictionary *) (the_dictionary_by_times_used → Binding());
};

Boolean SB_TYPE_USAGE_DICTIONARY::add_type(char *type_id, SB_TYPE_NAME
*type_name)
{
    Boolean return_flag;

    if(dictionary_by_type_id() → isIndex(type_id) == FALSE)
    {
        // ID NOT YET USED
        SB_TYPE_USAGE *new_usage = new SB_TYPE_USAGE(type_id, type_name);

        dictionary_by_type_id() → Insert(type_id, new_usage);
        dictionary_by_base_type() → Insert(type_name → base_type_id(), new_usage);
        dictionary_by_times_used() → Insert(0, new_usage);

        return_flag = TRUE;
    }
    else
    {
        // id already in use so can not insert
        return_flag = FALSE;
    };

    return return_flag;
};

Boolean SB_TYPE_USAGE_DICTIONARY::add_type(SB_TYPE_USAGE *type_usage)
{
    Boolean return_flag;

```

```

if(dictionary_by_type_id()→isIndex(type_usage→type_id())==FALSE)
{
    // ID NOT YET USED

    dictionary_by_type_id()→Insert(type_usage→type_id(),type_usage);
    dictionary_by_base_type()→Insert(type_usage→base_type_id(),type_usage);
    dictionary_by_times_used()→Insert(0,type_usage);

    return_flag=TRUE;
}
else
{
    // id already in use so can not insert
    return_flag=FALSE;
};

return return_flag;

};

Boolean SB.TYPE_USAGE_DICTIONARY::append(SB.TYPE_USAGE_DICTIONARY
*dictionary)
{
    Boolean return_flag=TRUE;

    DictionaryIterator next_id=dictionary→type_id_iterator();

    while(next_id.moreData() && return_flag==TRUE)
    {
        SB.TYPE_USAGE *the_usage=(SB.TYPE_USAGE *)(Entity *)next_id();
        return_flag=add_type(the_usage);
    };
    /* next_id.Reset();
    while(next_id.moreData())
    {
        SB.TYPE_USAGE *the_usage=(SB.TYPE_USAGE *)(Entity *)next_id();
        dictionary->remove_usage(the_usage);
    };

    dictionary->Destroy(FALSE);*/

    return return_flag;
};

void SB.TYPE_USAGE_DICTIONARY::remove_usage(SB.TYPE_USAGE *the_usage)
{
    dictionary_by_times_used()→Remove(the_usage→times_used(),the_usage);
    dictionary_by_type_id()→Remove(the_usage→type_id(),the_usage);
    dictionary_by_base_type()→Remove(the_usage→base_type_id(),the_usage);
}

```

```

};

void SB_TYPE_USAGE_DICTIONARY::printOn(ofstream& outstream)
{
    DictionaryIterator &next_decl=
        *(new DictionaryIterator(dictionary_by_base_type()));

    while(next_decl.moreData())
    {
        ((SB_TYPE_USAGE *)(Entity *) next_decl())->printOn(outstream);
        outstream << "\n";
    }
};

int SB_TYPE_USAGE_DICTIONARY::num()
{
    return (int)(dictionary_by_base_type()-Cardinality());
};

DictionaryIterator SB_TYPE_USAGE_DICTIONARY::type_id_iterator()
{
    return DictionaryIterator(dictionary_by_type_id());
};

DictionaryIterator SB_TYPE_USAGE_DICTIONARY::base_type_iterator()
{
    return DictionaryIterator(dictionary_by_base_type());
};

DictionaryIterator SB_TYPE_USAGE_DICTIONARY::times_used_iterator()
{
    return DictionaryIterator(dictionary_by_times_used());
};

Boolean SB_TYPE_USAGE_DICTIONARY::update(SB_TYPE_NAME *type_name)
{
    Boolean return_flag=FALSE;
    if(dictionary_by_type_id()-isIndex(type_name->id()))
    {
        // this type id is in the list so update its usage
        ((SB_TYPE_USAGE *)dictionary_by_type_id()-
            getElement(type_name->id()))->used();
        return_flag=TRUE;
    }
    return return_flag;
};

```

APPENDIX C - PARSER GENERATION INPUT FILES

A. LEX INPUT

```
%{  
# include "y.tab.h";  
int line_number=1;  
%}
```

```
a      [aA]  
b      [bB]  
c      [cC]  
d      [dD]  
e      [eE]  
f      [fF]  
g      [gG]  
h      [hH]  
i      [iI]  
j      [jJ]  
k      [kK]  
l      [lL]  
m      [mM]  
n      [nN]  
o      [oO]  
p      [pP]  
q      [qQ]  
r      [rR]  
s      [sS]  
t      [tT]  
u      [uU]  
v      [vV]  
w      [wW]  
x      [xX]  
y      [yY]  
z      [zZ]  
space  [ ]  
%%
```

```
{t}{y}{p}{e}  
{s}{p}{e}{c}{i}{f}{i}{c}{a}{t}{i}{o}{n}  
{e}{n}{d}  
{g}{e}{n}{e}{r}{i}{c}
```

```
return(TYPE);  
return(SPECIFICATION);  
return(END);  
return(GENERIC);
```

{o}{p}{e}{r}{a}{t}{o}{r}	return(OPERATOR);
{i}{n}{p}{u}{t}	return(INPUT);
{o}{u}{t}{p}{u}{t}	return(OUTPUT);
{s}{t}{a}{t}{e}{s}	return(STATES);
{e}{x}{c}{e}{p}{t}{i}{o}{n}{s}	return(EXCEPTIONS);
{b}{y}{space}{r}{e}{q}{u}{i}{r}{e}{m}{e}{n}{t}{s}	return(BY_REQ);
{d}{e}{s}{c}{r}{i}{p}{t}{i}{o}{n}	return(DESCRIPTION);
{a}{x}{i}{o}{m}{s}	return(AXIOMS);
{k}{e}{y}{w}{o}{r}{d}{s}	return(KEYWORDS);
{m}{a}{x}{i}{m}{u}{m}{space}{e}{x}{c}{e}{u}{t}{i}{o}{n}{space}{t}{i}{m}{e}	return(MAX_EXEC_TIME);
{m}{i}{c}{r}{o}{s}{e}{c}	return(MICROSEC);
{m}{s}	return(MS);
{s}{e}{c}	return(SEC);
{m}{i}{n}	return(MIN);
{h}{o}{u}{r}{s}	return(HOURS);
{a}{n}{d}	return(AND);
{o}{r}	return(OR);
{x}{o}{r}	return(XOR);
{t}{r}{u}{e}	return(TRUE);
{f}{a}{l}{s}{e}	return(FALSE);
{n}{o}{t}	return(NOT);
"<"	return('<');
">"	return('>');
"="	return('=');
">="	return(GTE);
"<="	return(LTE);
"!="	return(NEQV);
"+"	return('+');
"-"	return('-');
"&"	return('&');
"*"	return('*');
"/"	return('/');
{m}{o}{d}	return(MOD);
{r}{e}{m}	return(REM);
"**"	return(EXP);
[0-9][0-9]*	return(INTEGER_LITERAL);
[0-9][0-9]*"."[0-9]*	return-REAL_LITERAL);
""["^"]*""	return(STRING_LITERAL);
"."	return('.');
":"	return(':');
","	return(',');
"["	return('[');
"]"	return(']');
"("	return('(');
)"	return(')');
"{"[^"]*"}	{
	char *temp=yytext;
	while(*temp!=NULL)
	{
	if(*temp=='\n')

[a-zA-Z][a-zA-Z_0-9]*
[\\
"n"

```
        {  
            line_number++;  
        };  
        temp++;  
    };  
    return(TEXT_BLOCK);  
};  
        return(ID);  
  
    {  
        line_number++;  
    }  
};
```

B. YACC INPUT

%start component

```
%union {
    void* OBJECT_POINT;
}
```

%token ID TYPE SPECIFICATION END GENERIC

%token OPERATOR INPUT OUTPUT

%token STATES EXCEPTIONS BY_REQ DESCRIPTION AXIOMS

%token TEXT_BLOCK KEYWORDS

%token MOD GTE LTE

%token MS MICROSEC SEC HOURS MIN AND OR XOR

%token NEQV REM EXP

%token INITIALLY MAX_EXEC_TIME

%token INTEGER_LITERAL TRUE FALSE REAL_LITERAL STRING_LITERAL

%token NOT ABS

%type <OBJECT_POINT> type_spec optional_generic_specs

%type <OBJECT_POINT> optional_type_decl optional_operator_decl

%type <OBJECT_POINT> operator data_type type_name generic_attributes

%type <OBJECT_POINT> input_attributes output_attributes exceptions_attributes

%{

#include <stream.h> // c++ specific io routines

#include "sball.hxx"

/* this code allows the c++ compiler to use the c code generated by lex as standard c code */

extern "C--"

```
{
    extern int yylex();
    extern int line_number;
    extern char yytext[];
    #include <string.h>
    #include <ctype.h>
    #include <stdlib.h>
    #include <stdio.h>
}
```

extern int yyerror(char *);

extern SB_COMPONENT* YYPARSE_component; // global pointer to the main library object

extern Boolean YYPARSE_query_flag;

// predeclare functions for internal stacks

```

void      push_object(void *new_object);
void      *top_object();
void      *pop_object();
void      push_id(char *new_id);
char      *top_id();
char      *pop_id();
void      push_rid(char *new_id);
char      *top_rid();
char      *pop_rid();

```

```
// declare global variables for the parser
```

```

Boolean   SB_COMPONENT_ADT_FLAG=FALSE;
char      *psdl_type_name;

```

```
%}
```

```
%%
```

```
component:
```

```

    data_type
    {
        YYPARSE_component=(SB_COMPONENT *)$1;
    }
    |
    operator
    {
        YYPARSE_component=(SB_COMPONENT *)$1;
    };

```

```
data_type:
```

```

    TYPE
    push_ID
    {
        psdl_type_name=top_id();
        push_object(new SB_ADT_COMPONENT(pop_id()));
        SB_COMPONENT_ADT_FLAG=TRUE;
    }
    type_spec
    {
        $$=(SB_COMPONENT *)pop_object();
    };

```

```
type_spec:
```

```

    SPECIFICATION
    optional_generic_specs
    optional_type_decl
    optional_operator_decl

```

functionality

END

```
{
    if($2!=NULL)
    {
        ((SB_ADT_COMPONENT *)top_object()->
            insert_generics((SB_TYPE_USAGE_DICTIONARY *)$2);
    };
    if($3!=NULL)
    {
        ((SB_ADT_COMPONENT *)top_object()->
            insert_adt_usage((SB_TYPE_USAGE_DICTIONARY *)$3);
    };
    if($4!=NULL)
    {
        ((SB_ADT_COMPONENT *)top_object()->
            insert_operators((SB_ADT_OPERATOR_DICTIONARY *)$4);
    };
};
```

optional_generic_specs:

GENERIC

push_new_SB_TYPE_USAGE_DICTIONARY

list_of_type_decl

```
{
    $$=(SB_TYPE_USAGE_DICTIONARY *)pop_object();
};
|
/*optional*/
{ $$=NULL;};
```

optional_type_decl:

push_new_SB_TYPE_USAGE_DICTIONARY

list_of_type_decl

```
{
    $$=(SB_TYPE_USAGE_DICTIONARY *)pop_object();
}
|
/*optional*/
{ $$=NULL;};
```

optional_operator_decl:

push_new_SB_ADT_OPERATOR_DICTIONARY

operator_list

```
{
    $$=(SB_ADT_OPERATOR_DICTIONARY *)pop_object();
}
|
```

```

/*optional*/
{ $$=NULL; };

list_of_type_decl:

    list_of_type_decl
    ','
    type_decl
    |
    type_decl;

type_decl:

    push_id_list_start
    id_list
    ','
    type_name
    {
        // must use another stack in order to reverse the order back
        // to the original order in the declaration
        while(top_id()!=BOTTOM_ID)
        {
            push_rid(pop_id());
        }
        pop_id(); // pop off the BOTTOM_ID MARKER
        while(top_rid()!=NULL)
        {
            ((SB_TYPE_USAGE_DICTIONARY *)top_object())->
                add_type(pop_rid(),(SB_TYPE_NAME *)$4);
        }
    };

id_list:

    id_list
    ','
    push_ID
    |
    push_ID;

type_name:

    push_ID
    {
        $$=new SB_TYPE_NAME(pop_id(),(SB_ID_DECL_DICTIONARY*)NULL);
    };
    |
    push_ID
    '['
    push_new_SB_ID_DECL_DICTIONARY
    list_of_id_decl
    ']'

```

```

{
    $$=new SB_TYPE_NAME(pop_id(),(SB_ID_DECL_DICTIONARY*)pop_object());
};

list_of_id_decl:

    list_of_id_decl
    ';'
    id_decl
    |
    id_decl;

id_decl:

    push_id_list_start
    id_list
    ';'
    type_name
    {
        // must use another stack in order to reverse the order back
        // to the original order in the declaration
        while(top_id()!=BOTTOM_ID)
        {
            push_rid(pop_id());
        };
        pop_id(); // pop off the BOTTOM_ID MARKER
        while(top_rid()!=NULL)
        {
            ((SB_ID_DECL_DICTIONARY *)top_object())->
                add_decl(pop_rid(),(SB_TYPE_NAME *)$4);
        };
    };

operator_list:

    operator
    {
        ((SB_ADT_OPERATOR_DICTIONARY*)top_object())->add((SB_ADT_OPERATOR*)$1);
    };
    |
    operator
    operator_list
    {
        ((SB_ADT_OPERATOR_DICTIONARY*)top_object())->
            add((SB_ADT_OPERATOR*)$1);
    };

functionality:

    keywords
    informal_desc

```

formal_desc

keywords:

```
/*optional*/
|
KEYWORDS
push_id_list_start
id_list
{
    while(top_id()!=BOTTOM_ID)
    {
        (((SB_COMPONENT *)top_object())->keyword_dictionary())->add(pop_id());
    };
    pop_id(); // remove the bottom_id
};
```

informal_desc:

```
/*optional*/
|
DESCRIPTION
TEXT_BLOCK
{
    char *the_text=new char[strlen(yytext)+1];
    // put all but the opening { and closing } into the_text
    int i;
    for(i=1;i< strlen(yytext)-1;i++)
    {
        the_text[i-1]=yytext[i];
    };
    the_text[i-1]=NULL;
    (((SB_COMPONENT *)top_object())->informal_description())->append(the_text);
    delete the_text;
}
```

formal_desc:

```
{ /*optional*/ }
|
AXIOMS
TEXT_BLOCK
{
    char *the_text=new char[strlen(yytext)+1];
    // put all but the opening { and closing } into the_text
    int i;
    for(i=1;i< strlen(yytext)-1;i++)
    {
        the_text[i-1]=yytext[i];
    };
    the_text[i-1]=NULL;
```

```

        (((SB_COMPONENT *)top_object())->formal_description())->append(the_text);
delete the_text;
}

```

operator:

```

OPERATOR
push_ID
{
    if(SB_COMPONENT_ADT_FLAG==TRUE)
    {
        // concatonate the psdl_type name and the operator name to ensure
        // a unique name in the database for the new component
        char *component_name=new
        char[strlen(psdl_type_name)+
        strlen(top_id())+2];
        strcpy(component_name,psdl_type_name);
        strcat(component_name,".");
        strcat(component_name,pop_id());
        push_object(new SB_ADT_OPERATOR(component_name));
        delete component_name;
    }
    else
    {
        push_object(new SB_OPERATOR_COMPONENT(pop_id()));
    };
}
operator_spec
{
    $$=pop_object();
};

```

operator_spec:

```

SPECIFICATION
operator_interface
functionality
END

```

operator_interface:

```

{ /*empty */
|
attribute
req_trace
operator_interface

```

attribute:

```

generic_attributes
{

```

```

        ((SB_OPERATOR
*)top_object()->insert_generics((SB_TYPE_USAGE_DICTIONARY *)$1);
    };
    |
    input_attributes
    {
        ((SB_OPERATOR *)top_object()->add_inputs((SB_ID_DECL_DICTIONARY *)$1);
    };
    |
    output_attributes
    {
        ((SB_OPERATOR *)top_object()->
            add_outputs((SB_ID_DECL_DICTIONARY *)$1);
    };
    |
    state_attributes
    |
    exceptions_attributes
    {
        ((SB_OPERATOR *)top_object()->
            add_exceptions((SB_EXCEPTION_DICTIONARY *)$1);
    };
    |
    max_execution_attribute;

req_trace:
    { /*empty*/ }
    |
    BY_REQ
    id_list

generic_attributes:

    GENERIC
    push_new_SB_TYPE_USAGE_DICTIONARY
    list_of_type_decl
    {
        $$=(SB_TYPE_USAGE_DICTIONARY *)pop_object();
    };

input_attributes:

    INPUT
    push_new_SB_ID_DECL_DICTIONARY
    list_of_id_decl
    {
        $$=(SB_ID_DECL_DICTIONARY *)pop_object();
    };

output_attributes:

```

OUTPUT

```
push_new_SB_ID_DECL_DICTIONARY
list_of_id_decl
{
    $$=(SB_ID_DECL_DICTIONARY *)pop_object();
};
```

state_attributes:

```
STATES
list_of_id_decl
{
    // through away the list of type decl since it is not used
    pop_object();
}
INITIALLY
initial_expression_list
;
```

exceptions_attributes:

```
EXCEPTIONS
push_new_SB_EXCEPTION_DICTIONARY
push_id_list_start
id_list
{
    while(top_id()!=BOTTOM_ID)
    {
        ((SB_EXCEPTION_DICTIONARY *)top_object())->add(pop_id());
    };
    pop_id(); // pop off the bottom marker
    $$=(SB_EXCEPTION_DICTIONARY *)pop_object();
};
```

max_execution_attribute:

```
MAX_EXEC_TIME time;
```

time:

```
INTEGER_LITERAL MICROSEC
|
INTEGER_LITERAL MS
|
INTEGER_LITERAL SEC
|
INTEGER_LITERAL MIN
|
INTEGER_LITERAL HOURS;
```

initial_expression_list:

```
initial_expression_list
```

```

::
initial_expression
|
initial_expression;

initial_expression:
TRUE
|
FALSE
|
INTEGER_LITERAL
|
REAL_LITERAL
|
STRING_LITERAL
|
ID
|
type_name
::
ID
|
type_name
::
ID
'| initial_expression_list '|
|
'| initial_expression '|
|
initial_expression
log_op
initial_expression
initial_expression
rel_op
initial_expression
| '-'
initial_expression
|
'| '+'
initial_expression
|
initial_expression
bin_add_op
initial_expression
|
initial_expression
bin_mul_op
initial_expression
|
initial_expression
EXP

```

```

    initial_expression
    |
    NOT
    initial_expression
    |
    ABS
    initial_expression;

log_op:
    AND
    |
    OR
    |
    XOR;

rel_op:
    '='
    |
    '>'
    |
    GTE
    |
    LTE
    |
    NEQV;

bin_add_op:
    '+'
    |
    '-'
    |
    '&';

bin_mul_op:
    '*'
    |
    '/'
    |
    MOD
    |
    REM;

push_ID:
    ID
    {
        char *new_id=new char[strlen(yytext)+1];
        strcpy(new_id,yytext);
        push_id(new_id);
    };

```

```

push_new_SB_EXCEPTION_DICTIONARY:
{
    push_object(new SB_EXCEPTION_DICTIONARY());
};

push_new_SB_TYPE_USAGE_DICTIONARY:
{
    push_object(new SB_TYPE_USAGE_DICTIONARY());
};

push_new_SB_ADT_OPERATOR_DICTIONARY:
{
    push_object(new SB_ADT_OPERATOR_DICTIONARY());
};

push_id_list_start:
{
    push_id(BOTTOM_ID);
};

push_new_SB_ID_DECL_DICTIONARY:
{
    push_object(new SB_ID_DECL_DICTIONARY());
};

%%

/* define the id stack and the object pointer stack */

typedef struct OBJECT_STACK_RECORD
{
    void *object_point;
    OBJECT_STACK_RECORD *next_record;
} OBJECT_STACK_RECORD;

typedef struct ID_STACK_RECORD
{
    char *id;
    ID_STACK_RECORD *next_record;
} ID_STACK_RECORD;

typedef struct NAME_STACK_RECORD
{
    char *name;
    NAME_STACK_RECORD *next_record;
} NAME_STACK_RECORD;

```

```

char BOTTOM_ID_MARKER[2]="#"; /* used as a bottom of id_list marker */
char *BOTTOM_ID=(char *)BOTTOM_ID_MARKER;

OBJECT_STACK_RECORD *top_object_in_stack=NULL;
ID_STACK_RECORD *top_id_in_stack=NULL;
ID_STACK_RECORD *top_rid_in_stack=NULL;
NAME_STACK_RECORD *top_name_in_stack=NULL;

void push_object(void *new_object)
{
    OBJECT_STACK_RECORD *new_object_record=new OBJECT_STACK_RECORD;
    new_object_record->object_point=new_object;
    new_object_record->next_record=top_object_in_stack;
    top_object_in_stack=new_object_record;
};

void *top_object()
{
    void *return_object=NULL;
    if(top_object_in_stack!=NULL)
    {
        return_object=top_object_in_stack->object_point;
    }
    else
    {
        cerr << "error in object stack tried to view top object that was null\n";
    };
    return return_object;
};

void *pop_object()
{
    void *return_object=NULL;
    OBJECT_STACK_RECORD *temp_point=top_object_in_stack;
    if(top_object_in_stack!=NULL)
    {
        return_object=top_object_in_stack->object_point;
        top_object_in_stack=top_object_in_stack->next_record;
        delete temp_point;
    }
    else
    {
        cerr << "PARSER: error in object stack read past end\n";
    };
    return return_object;
};

void push_rid(char *new_id)
{
    ID_STACK_RECORD *new_id_record=new ID_STACK_RECORD;
    new_id_record->id=new_id;

```

```

    new_id_record->next_record=top_rid_in_stack;
    top_rid_in_stack=new_id_record;
};

char *top_rid()
{
    char *return_id=NULL;
    if(top_rid_in_stack!=NULL)
    {
        return_id=top_rid_in_stack->id;
    };
    return return_id;
};

char *pop_rid()
{
    char *return_id=NULL;
    if(top_rid_in_stack!=NULL)
    {
        return_id=top_rid_in_stack->id;
        ID_STACK_RECORD *temp_point=top_rid_in_stack;
        top_rid_in_stack=top_rid_in_stack->next_record;
        delete temp_point;
    }
    else
    {
        cerr << "PARSER: error in rid stack read past end\n";
    };
    return return_id;
};

void push_id(char *new_id)
{
    ID_STACK_RECORD *new_id_record=new ID_STACK_RECORD;
    new_id_record->id=new_id;
    new_id_record->next_record=top_id_in_stack;
    top_id_in_stack=new_id_record;
};

char *top_id()
{
    char *return_id=NULL;
    if(top_id_in_stack!=NULL)
    {
        return_id=top_id_in_stack->id;
    }
    else
    {
        cerr << "PARSER: error in id stack looked at NULL record\n";
    };
    return return_id;
};

```

```

};

char *pop_id()
{
    char *return_id=NULL;
    if(top_id_in_stack!=NULL)
    {
        return_id=top_id_in_stack->id;
        ID_STACK_RECORD *temp_point=top_id_in_stack;
        top_id_in_stack=top_id_in_stack->next_record;
        delete temp_point;
    }
    else
    {
        cerr << "PARSER: error in id stack read past end\n";
    };
    return return_id;
};

```

APPENDIX D - INTEGRATING ADA COMPONENTS INTO CAPS

Once a reusable component has been retrieved it must be transformed for use in the prototype being developed. As previously discussed, this requires that the names for parameters, streams, operators, and types be changed to match those of the query component. Along with this transformation the execution support system expects several naming conventions to be followed for Ada components. This appendix will discuss how an Ada reusable component can be transformed into the domain of the query component as well as the naming conventions the execution support system expects for Ada components. An example of the transformation process is also included.

A. ADA REUSABLE COMPONENT NAMING CONVENTIONS

The execution support system requires that all Ada reusable components are implemented via packages. To simplify the process of identifying package names the following conventions are used.

1. Operators

An operator with an ID of `operator_name` will be implemented in a package named `operator_name_pkg`. The operator itself will be implemented by the procedure `operator_name_pkg.operator_name`.

2. Types

A PSDL type with an ID of `psdl_type_name` will be implemented in a package named `psdl_type_name_pkg`.

B. EXAMPLE

1. Query Specification

```
type integer_set
specification

    integer_set : ADT

    operator create
    specification
        output
            the_set : integer_set
    end
    operator insert
    specification
        input
            x : integer,
            in_set : integer_set
        output
            out_set : integer_set
    end
    operator remove
    specification
        input
            x : integer,
            in_set : integer_set
        output
            out_set : integer_set
    end
    operator member
    specification
        input
            x : integer,
            in_set : integer_set
        output
            result : boolean
    end
end
```

2. Library Specification Located As A Match

```
type set
specification
  generic t : GENERIC_TYPE,
         block_size : GENERIC_VALUE,
         eq : GENERIC_PROCEDURE
```

```
set : ADT
```

```
operator empty
specification
  output
    s : set
end
```

```
operator add
specification
  input
    x : t,
    si : set
  output
    so : set
end
```

```
operator remove
specification
  input
    x : t,
    si : set
  output
    so : set
end
```

```
operator member
specification
  input
    x : t,
    s : set
  output
    v : boolean
end
```

```
operator union
specification
  input
    s1,s2 : set
  output
    s3 : set
end
```

```

operator difference
specification
    input      s1,s2 : set
    output     s3 : set
end

operator intersection
specification
    input      s1,s2 : set
    output     s3 : set
end

operator size
specification
    input      s : set
    output     v : natural
end

operator equal
specification
    input      s1,s2 : set
    output     v : boolean
end

operator subset
specification
    input      s1,s2 : set
    output     v : boolean
end

keywords SET

description ( SET ADT WITH OPERATIONS FOR EMPTY, ADD, SUBSET, EQUAL )
end

```

3. Matching Map

TYPE set -> integer_set

MAP

set -> integer_set

GENERIC

t -> integer

eq -> UNDEFINED

OPERATOR empty -> create

MAP

OUTPUT

s : set -> new_set : integer_set

END

OPERATOR add -> insert

MAP

INPUT

x : t -> x : integer,

si : set -> in_set : integer_set

OUTPUT

so : set -> out_set : integer_set

END

OPERATOR remove -> remove

MAP

INPUT

x : t -> x : integer,

si : set -> in_set : integer_set

OUTPUT

so : set -> out_set : integer_set

END

OPERATOR member -> member

MAP

INPUT

x : t -> x : integer,

si : set -> in_set : integer_set

OUTPUT

v : boolean -> result : boolean

END

END

The grammar that defines this mapping language is included in this Appendix.

4. Generated Ada Code

From this matching map the following Ada specification and implementation can be generated to implement integer_set.

C. ADA SPECIFICATION

```
with sb_set_pkg;
package integer_set_pkg is
  type integer_set is private;

  procedure create(new_set : out integer_set);

  procedure insert(x : in integer;
                  in_set : in integer_set;
                  out_set : out integer_set);

  procedure remove(x : in integer;
                  in_set : in integer_set;
                  out_set : out integer_set);

  procedure member(x : in integer;
                  in_set : in integer_set;
                  result : out boolean);

private
  package sb_set_pkg_to_integer_set_pkg is new sb_set_pkg(integer,UNDEFINED);
  integer_set is subtype sb_set_pkg_to_integer_set_pkg.set;
end integer_set_pkg;
```

D. ADA IMPLEMENTATION

```
package body integer_set_pkg is

  procedure create(new_set : out integer_set) is
  begin
    sb_set_pkg_to_integer_set_pkg.empty(new_set);
  end;

  procedure insert(x : in integer;
                  in_set : in integer_set;
                  out_set : out integer_set) is
  begin
    sb_set_pkg_to_integer_set_pkg.insert(x,in_set,out_set);
  end;

  procedure remove(x : in integer;
                  in_set : in integer_set;
                  out_set : out integer_set) is
  begin
    sb_set_pkg_to_integer_set_pkg.remove(x,in_set,out_set);
  end;

  procedure member(x : in integer;
                  in_set : in integer_set;
                  result : out boolean) is
  begin
    sb_set_pkg_to_integer_set_pkg.member(x, in_set,result);
  end;
end integer_set_pkg;
```

E. PSDL MATCHING MAP GRAMMER (YACC INPUT FORMAT)

```
%start psdl_map
%token ID OPERATOR MAPS_TO MAP END TYPE GENERIC COMMA INPUT OUTPUT
%token COLON UNDEFINED
```

```
%%
```

```
psdl_map:
    operator_map
    |
    type_map
```

```
operator_map:
    OPERATOR
    library_id
    MAPS_TO
    query_id
    MAP
    operator_attributes
    END
```

```
type_map:
    TYPE
    library_id
    MAPS_TO
    query_id
    MAP
    type_attributes
    END
```

```
type_attributes:
    adt_map_list
    generic_map_list
    operator_map_list
```

```
adt_map_list:
    { /*OPTIONAL*/ }
    |
    map_type_list
```

```
generic_map_list:
    { /*OPTIONAL*/ }
    |
    GENERIC
    map_type_list
```

operator_map_list:
operator_map_list
COMMA
operator_map
|
operator_map

operator_attributes:
generic_map_list
input_map_list
output_map_list
exception_map_list

input_map_list:
{/*OPTIONAL*/}
|
INPUT
map_decl_list

output_map_list:
{/*OPTIONAL*/}
|
OUTPUT
map_decl_list

exception_map_list:
{/*OPTIONAL*/}
|
EXCEPTION
map_exception_list

map_decl_list:
map_decl_list
COMMA
map_decl
|
map_decl

map_decl:
library_id
COLON
library_type_id
MAPS_TO
query_id
COLON
query_type_id
|
library_id
MAPS_TO
UNDEFINED

```

map_type_list:
  map_type_list
  COMMA
  map_type
  |
  map_type

map_type:
  library_type_id
  MAPS_TO
  query_type_id
  |
  library_type_id
  MAPS_TO
  UNDEFINED

map_exception_list:
  map_exception_list
  COMMA
  map_exception
  |
  map_exception

map_exception:
  library_id
  MAPS_TO
  query_id
  |
  library_id
  MAPS_TO
  UNDEFINED
  |
  UNDEFINED
  MAPS_TO
  query_id

library_id:
  id

query_id:
  id

library_type_id:
  id

query_type_id:
  id

id:
  ID

```

APPENDIX E - COMMAND LINE INTERFACE SPECIFICATION

The software base has been implemented with a command line interface to simplify its integration into CAPS. This Appendix provides a detailed specification of the command line interface for the software base.

The following is a list of the software base commands available and how to use them. Each command is prefixed by the name of the software base executable (i.e. `caps_software_base`).

A. MAKE NEW SOFTWARE BASE LIBRARY

1. Command

```
ml library_name type_matching_rule_file
```

2. Description

Creates a new reusable component library within the software base named `library_name` using the type matching rules specified in the file `type_matching_rule_file`. See Appendix A for a description of the contents of the type matching rule file.

3. Example

```
%caps_software_base ml Ada Ada_rule_file
```

B. DELETE SOFTWARE BASE LIBRARY

1. Command

```
dl library_name
```

2. Description

Deletes the library **library_name** from the software base and all of the components in this library.

3. Example

```
%caps_software_base dl Ada
```

C. ADD A COMPONENT TO A SOFTWARE BASE LIBRARY

1. Command

```
ca library_name psdl_file imp_spec imp_body
```

2. Description

Adds the component specified in **psdl_file** to the library named **library_name**.

The implementation source code is in the files **imp_spec** and **imp_body**.

3. Example

```
%caps_software_base ca Ada sb_set.psdl sb_set.spec.a sb_set.body.a
```

D. DELETE A COMPONENT FROM A SOFTWARE BASE LIBRARY

1. Command

`cd library_name component_name`

2. Description

Deletes the component named **component_name** from the library named **library_name**.

E. UPDATE A COMPONENT IN A SOFTWARE BASE LIBRARY

1. Example

`%caps_software_base cd Ada sb_set`

2. Command

`cu library_name psdl_file imp_spec imp_body`

3. Description

Update the existing component specified in **psdl_file** with the new specification given in **psdl_file** and the new implementation given in the files **imp_spec** and **imp_body**.

4. Example

`%caps_software_base cu Ada set.psdl new_set.spec.a new_set.body.a`

F. VIEW A COMPONENT IN A SOFTWARE BASE LIBRARY

1. Command

```
cv library_name component_name sb_set.psdl sb_set.spec.a sb_set.body.a
```

2. Description

Generate text files for viewing the component **component_name** which is in library **library_name**. The text is written to the files specified by **psdl_file**, **imp_spec**, **imp_body**.

3. Example

```
%caps_software_base cv Ada sb_set.psdl sb_set.spec.a sb_set.body.a
```

G. LIST OF KEYWORDS IN A SOFTWARE BASE LIBRARY

1. Command

```
kwl library_name output_file
```

2. Description

Generate a list of keywords defined in library **library_name**. This list is provided to allow the formulation of keyword queries or when selecting keywords for a new component. The list of keywords is written to the file **output_file**.

3. Example

```
%caps_software_base kwl Ada keyword_list
```

H. LIST OF COMPONENTS IN A SOFTWARE BASE LIBRARY

1. Command

`cl library_name output_file`

2. Description

Generate a list of component names defined in library **library_name**. This list can be used for named look up of components in the software base. The list is written to the file **output_file**.

3. Example

`%caps_software_base cl Ada component_list`

I. LIST OF PSDL TYPES IN A SOFTWARE BASE LIBRARY

1. Command

`tl library_name output_file`

2. Description

Generate a list of PSDL type components defined in library **library_name**. This list can be used for named look up of type components in the software base. The list is written to the file **output_file**.

3. Example

`%caps_software_base tl Ada type_list`

J. LIST OF PSDL OPERATORS IN A SOFTWARE BASE LIBRARY

1. Command

`ol library_name output_file`

2. Description

Generate a list of PSDL operator components defined in library **library_name**. This list can be used for named look up of components in the software base. The list is written to the file **output_file**.

3. Example

`%caps_software_base ol Ada operator_list`

K. KEYWORD QUERY

1. Command

`kwq library_name keyword_list output_file`

2. Description

Perform a keyword query on library **library_name** using the keywords in the file **keyword_list** and write the output to **output_file**. The output file contains the component name, the percentage of keywords matched, and the first line of the description of the component.

3. Example

`%caps_software_base kwq Ada query_keyword_list result_list`

L. COMPONENT QUERY

1. Command

```
cq library_name psdl_file output_file
```

2. Description

Performs a query by specification on the library **library_name** using the PSDL specification in **psdl_file** for the query. Writes the output to **output_file**. The output file contains the name of the component, the percentage score from semantic matching, and the first line of the description for the component.

3. Example

```
%caps_software_base cq Ada query.file result_file
```

M. GENERATE MATCHING MAP

1. Command

```
cgm library_name psdl_file component_name output_file
```

2. Description

Generate a matching map of how the component **component_name** matches the PSDL specification in **psdl_file** and writes the map to the file **output_file**. This function is currently not implemented.

3. Example

```
%caps_software_base cgm Ada query.psd result_file
```

N. COMPONENT DIAGNOSTICS

1. Command

`cdiag library_name component_name output_file`

2. Description

Creates a text file that contains diagnostic information about the component.

3. Example

`%caps_software_base cdiag Ada sb_set`

APPENDIX F - CAPS SOFTWARE BASE GRAPHICAL USER INTERFACE USERS MANUAL

A. BASIC MOUSE TECHNIQUES

1. Clicking

Clicking the mouse means moving the cursor to the desired location and pressing the left mouse button and then releasing it.

2. Double Clicking

Double clicking means clicking the mouse on an item twice in rapid succession. This technique is often used to select an item from a list of items.

3. Dragging

Dragging an item is accomplished by moving the cursor to the desired item and pressing the left mouse button. While holding the button down the item can be moved (dragged) to the desired location. To complete the operation simply release the mouse button.

4. Push Buttons

A button is pushed by clicking the left mouse button while the mouse cursor is over the button. Pushing a button will cause the labeled action to take place.

5. Pull Down Menus

A pull down menu is selected by clicking the left mouse button on the menu title and holding the button down. A list of the available options for this menu will be displayed. To select one of the available options move the mouse to the option (a highlighted bar will follow the mouse) and let up on the mouse button. Letting up on the mouse button anywhere outside of the pull down menu's option list will take no action.

6. Scrolling

1. Click on scroll bar arrows to scroll display one line in the desired direction.
2. Click above or below the scroll bar display icon to move up or down a page of information.
3. Drag the scroll bar display icon to the desired position in the view.
4. Use the middle mouse button to get the "Grabber Hand" which can be used to move the display. This method can be used on all scrolling views even if there is no scroll bar. This is useful for string editors that have no scroll bars.

7. Sizing Windows

All of the windows in the Software Base user interface have been designed to allow resizing to user preferences. The method used to resize a window depends on the version of X11 window manager that is in use. The examples in this manual are for the OpenWindows manager.

To resize a windows, In OpenWindows, simple drag any corner of the window in the desired direction and the window will be resized.

B. STARTING THE SOFTWARE BASE GRAPHICAL USER INTERFACE

The Software Base graphical user interface is started by executing the command `softbase.exe` from the command line. The path for the CAPS executables must be in your path. Currently you must be either working on `suns5` or `rxterm'd` to `suns5` to use this interface. This is due to the requirement that the Interviews 3.0 libraries must be mounted for the interface to execute. These libraries are currently only mounted on `suns5`.

C. CAPS SOFTWARE BASE MAIN MENU (FIGURE F.1)

The CAPS Software Base main menu is the top level of the Software Base Browsing System. From here all options of the Software Base are available. These options are to add new components, update existing components, delete components, browse by keyword, browse operators, browse types, query for a given specification, and get on-line help (The help system is currently not implemented).

These options are organized into four categories: File, Browse, Query, and Help. These categories make up the main menu for the system.

1. File

The file option is a pull down menu of operations. These operations are Add Component, Update Component, and Quit. See section A.5 for details on how to select pull-down menu items.

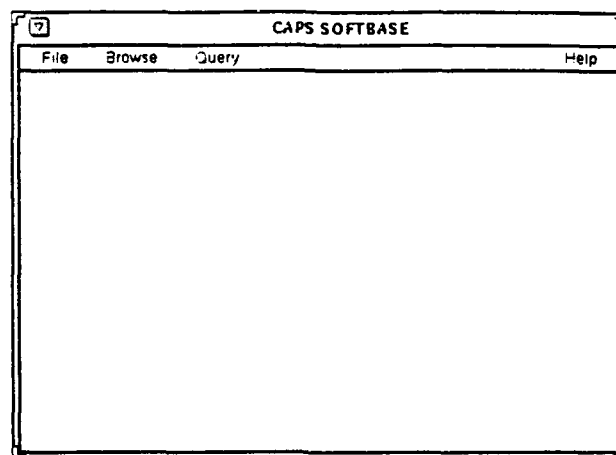


Figure - F.1 Main Menu

a. Add Component

When this option is chosen the user will be prompted for the input files by the input file selection window. See section D for a detailed description of how to use this window.

Once the input files have been selected the system attempts to add the component to the software base. If an error occurs an error display will be provided. If no errors have occurred the input window will be removed from the screen to indicate a successful addition.

b. Update Component

This option is used to provide an updated version of an existing component. The method used is the same as for adding components except that the new PSDL specification and implementation files will replace those that are currently in the software base.

c. Quit

This option quits the Software base Browsing system.

2. Browse

The browse option is a pull-down menu of browsing operations. These operations are By Keyword, By Operator, and By Type. See section I for details on how to select pull-down menu items.

a. By Keyword

Browsing by keyword means that the user will provide a list of desired keywords via the Keyword Selection Window and then will be given a Component Selection Window containing of those components in the software base which are members of at least one of those keyword categories.

The contents of the Component Selection Window are ordered such that those components which are members of more of the desired keyword categories are first. See section E for more information on using the Keyword Selection Window and section F for use of the Component Selection Window.

b. By Operator

Browsing by operator provides a Component Selection Window containing all of the operator components in the software base. These components are ordered alphabetically. See section F for details of how to use the Component Selection Window.

c. By Type

Browsing by operator provides a Component Selection Window containing all of the type components in the software base. These components are ordered alphabetically. See section F for details of how to use the Component Selection Window.

3. Query

The query option allows for a query of the software base based on a given PSDL specification. The user is prompted for a query specification with the Query Specification Window. If any components were found that match the query specification then a Component Selection Window is provided with all of the names of the matches in it. See section VII for details on the Component Selection Window.

4. Help

This option provides an on-line version of this manual (not implemented).

D. INPUT FILE SELECTION WINDOW (FIGURE F.2)

Inputs to the software base are made up of three text files. The PSDL specification, the implementation specification, and the implementation body. The input file selection window allows the selection of each of these files.

The file selection boxes show the current working directory in the Directory Box, and all of the files in that directory in the rest of the box. Double clicking on a file in one of the file selection boxes selects that file. Double clicking on a directory in a file

selection box will change to that directory. A new directory can be entered manually by typing it in the Directory box. The name of the file can be entered manually by typing it into the File Name box.

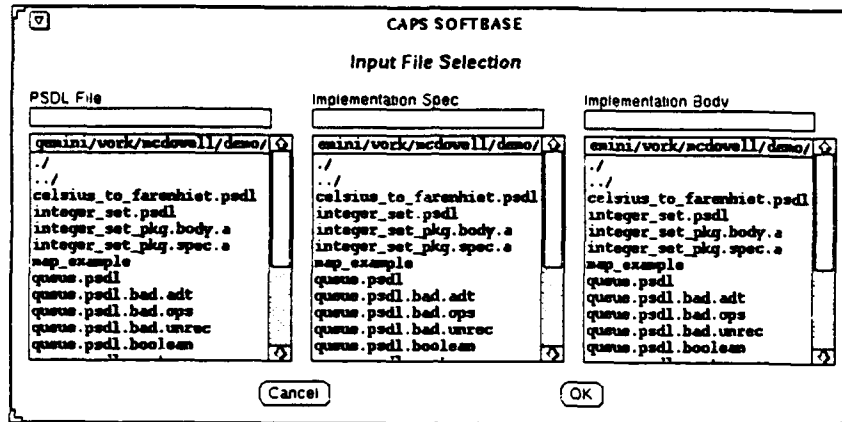


Figure - F.2 Input File Selector

Once all of the input files have been entered pushing, OK will cause the input files to be processed. Pushing Cancel will cause the input operation to terminate.

E. KEYWORD SELECTION WINDOW (FIGURE F.3)

This window allows the selection of keywords for a keyword search of the software base. All keyword categories in the software base are list in the left hand Box. Double clicking on a keyword will add it to the Keyword Selected box on the right hand side. Double clicking on keywords in the Selected box will remove them. Once the desired keywords have been selected pushing OK will start the search of the software base. Pushing Cancel will abort the search.

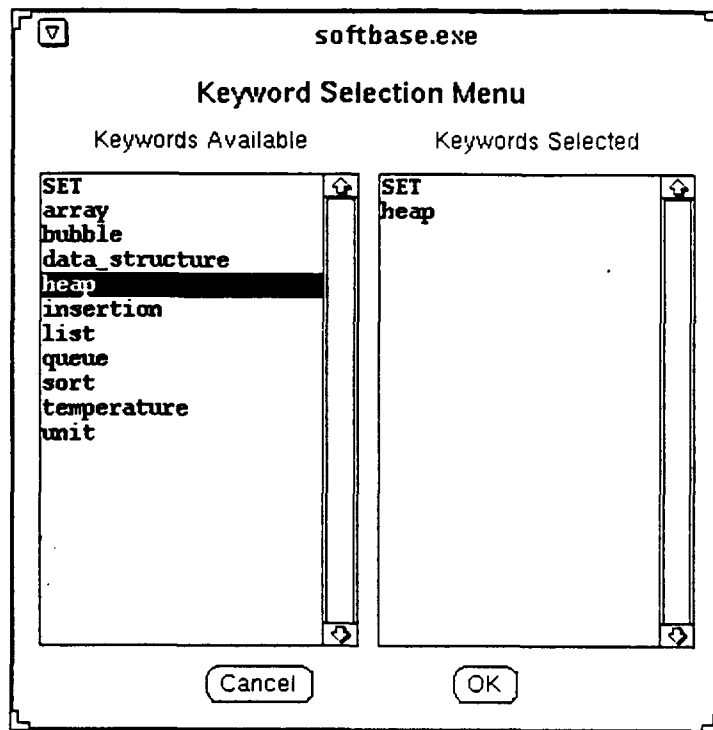


Figure - F.3 Keyword Selector

F. COMPONENT SELECTION WINDOW (FIGURE F.4)

This window displays a list of component names and a one line description of each component. Double clicking on a component will bring up a view of that components PSDL specification. See section VIII for details on using this view. This window is not removed automatically when a component is selected for viewing so that multiple components can be viewed simultaneously. To remove this window from the display push the cancel button. Pressing the OK button will view the currently selected component.

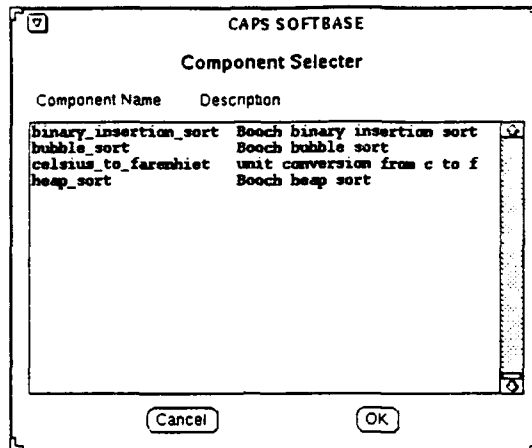


Figure - F.4 Component Selector

G. PSDL SPECIFICATION VIEWING WINDOW (FIGURE F.5)

This window displays the PSDL specification for a given component and allows various actions to take place on that component. The action available are: printing the specification, saving the specification to a file, deleting the component from the software base, viewing the components Ada specification, and searching for a given text string in the specification.

1. File

This is a pull-down menu for the print, save, delete, and quit view operations.

a. Print

Causes a printout of the specification to be spooled to the default printer.

b. Save As

Prompts the user for a file name and saves the specification to that file.

c. Delete

The system verifies that no other components are dependant on this component and if not allows the user to confirm that they wish to remove this component from the software base.

d. Quit View

Removes this view from the display.

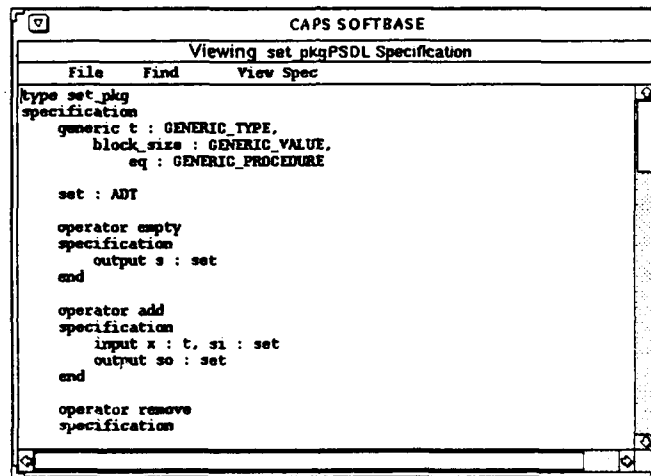


Figure - A.5 PSDL Specification Viewing Window

2. Find

Prompts for text to search for and if found repositions the cursor to that text (not implemented).

3. View Ada Specification

Provides the Ada Specification Viewing Window for this component.

H. ADA SPECIFICATION VIEWING WINDOW (FIGURE F.6)

This window displays the Ada specification for a given component and allows various actions to take place on that component. The action available are: printing the specification, saving the specification to a file, viewing the components Ada body, and searching for a given text string in the specification.

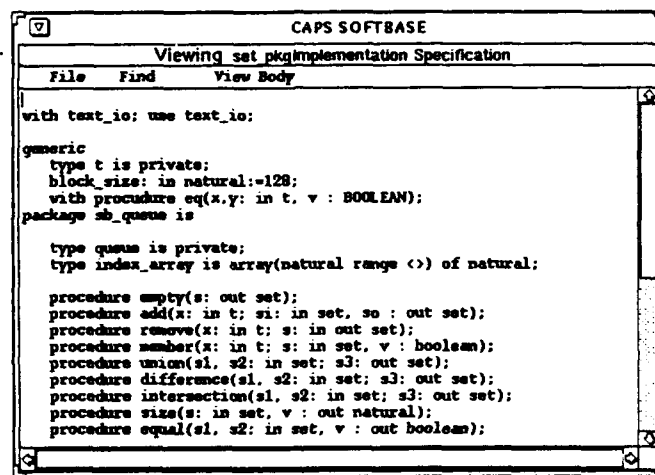


Figure - F.6 Ada Specification Viewing Window

1. File

This is a pull-down menu for the print, save, and quit view operations.

a. Print

Causes a printout of the specification to be spooled to the default printer.

b. Save As

Prompts the user for a file name and saves the specification to that file.

c. Quit View

Removes this view from the display.

2. Find

Prompts for text to search for and if found repositions the cursor to that text (not implemented).

3. View Ada Body

Provides the Ada Body Viewing Window for this component.

I. ADA BODY VIEWING WINDOW (FIGURE 7)

This window displays the Ada body for a given component and allows various actions to take place on that component. The action available are: printing the body, saving the body to a file, and searching for a given text string in the body.

1. File

This is a pull-down menu for the print, save, and quit view operations.

a. Print

Causes a printout of the body to be spooled to the default printer.

b. Save As

Prompts the user for a file name and saves the body to that file.

c. Quit View

Removes this view from the display.

2. Find

Prompts for text to search for and if found repositions the cursor to that text
(not implemented).

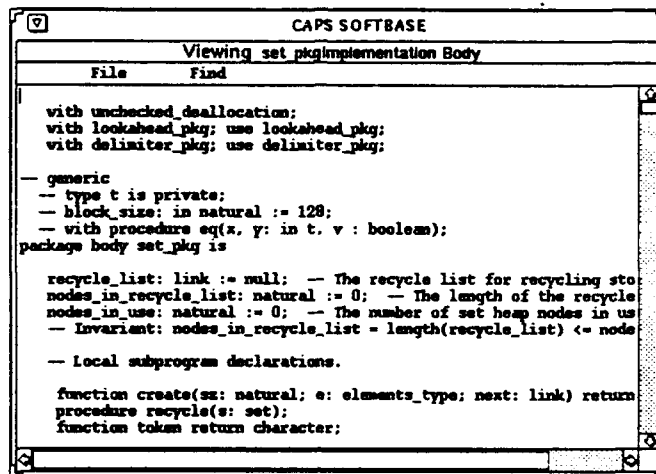


Figure - F.7 Ada Body Viewing Window

APPENDIX - G SOFTWARE BASE GRAPHICAL USER INTERFACE SOURCE CODE

All of the classes with the extension of "-core" in their name are implemented with code generated by the ibuild tool which is part of InterViews 3.0b [Ref 19]. This code is not presented here since it was machine generated. The class definitions for these "core classes" are included since the leaf classes inherit from the "core classes". Ibuild generated skeletons for the leaf classes. All that was required to implement this GUI was the addition of the code for each of the leaf class methods.

```

#ifndef SB_main_menu_core.h
#define SB_main_menu_core.h

#include <InterViews/scene.h>

class MenuItem;

class SB_main_menu_core : public MonoScene {
public:
    SB_main_menu_core(const char*);
    virtual void _AddComponent();
    virtual void _UpdateComponent();
    virtual void quit();
    virtual void browse_by_keyword();
    virtual void browse_by_type();
    virtual void browse_by_operator();
    virtual void query();
    virtual void _Help();
protected:
    Interactor* Interior();
protected:
    MenuItem* the_menu_file_quit;
    MenuItem* the_menu_browse_keyword;
    MenuItem* the_menu_browse_type;
    MenuItem* the_menu_browse_by_operator;
    MenuItem* the_menu_query;
};

#endif

```

```

#ifndef SB_main_menu_h
#define SB_main_menu_h

#include "SB_main_menu-core.h"

class SB_main_menu : public SB_main_menu_core {
public:
    SB_main_menu(const char*);

    virtual void _AddComponent();
    virtual void _UpdateComponent();
    virtual void quit();
    virtual void browse_by_keyword();
    virtual void browse_by_type();
    virtual void browse_by_operator();
    virtual void query();
    virtual void _Help();
};

#endif

```

```

#ifndef body_viewer_core.h
#define body_viewer_core.h

#include <InterViews/scene.h>

class HBox;
class Message;
class MenuBar;
class PulldownMenu;
class TextEditor;
class ButtonState;

class body_viewer_core : public MonoScene {
public:
    body_viewer_core(const char*);
    virtual void _Saveas();
    virtual void _Print();
    virtual void _QuitView();
    virtual void _Find();
protected:
    Interactor* Interior();
protected:
    HBox* the_file_name;
    Message* default_message;
    MenuBar* menu_bar;
    PulldownMenu* file_menu;
    TextEditor* the_editor;
};

#endif

```

```

#ifndef body_viewer_h
#define body_viewer_h

#include "body_viewer-core.h"

class body_viewer : public body_viewer_core {
private:
    char* the_string;
    char* component_name;
    TextBuffer *the_buffer;

public:
    body_viewer(const char*component_name, char *body_file);

    virtual void _Saveas();
    virtual void _Print();
    virtual void _QuitView();
    virtual void _Find();
};

#endif

```

```

#ifndef component_selector_core_h
#define component_selector_core_h

#include <InterViews/scene.h>

class StringBrowser;
class ButtonState;
class PushButton;

class component_selector_core : public MonoScene {
public:
    component_selector_core(const char*);
    virtual void selected();
    virtual void cancel();
    virtual void okay();
protected:
    Interactor* Interior();
protected:
    ButtonState* the_browser_BS;
    ButtonState* the_cancel_BS;
    ButtonState* the_ok_BS;
    StringBrowser* the_browser;
    PushButton* cancel_button;
    PushButton* ok_button;
};

#endif

```

```

#ifndef component_selector.h
#define component_selector.h

#include "component_selector-core.h"
#include <stream.h>

class component_selector : public component_selector_core {
public:
    component_selector(const char*);

    void Insert_components();
    void Insert_components( char *file_name);
    virtual void selected();
    virtual void cancel();
    virtual void okay();
};

#endif

```

```

#ifndef delete_warning_core.h
#define delete_warning_core.h

#include <InterViews/dialog.h>

class ButtonState;

class delete_warning_core : public Dialog {
public:
    delete_warning_core(const char*);

protected:
    Interactor* Interior();
protected:
};

#endif

```

```
#ifndef delete_warning_h
#define delete_warning_h

#include "delete_warning-core.h"

class delete_warning : public delete_warning_core {
public:
    delete_warning(const char*);

};

#endif
```

```

#ifndef dependancy_selector_core.h
#define dependancy_selector_core.h

#include <InterViews/scene.h>

class Message;
class StringBrowser;
class ButtonState;
class PushButton;

class dependancy_selector_core : public MonoScene {
public:
    dependancy_selector_core(const char*);
    virtual void new_selection();
    virtual void remove_selection();
    virtual void cancel();
    virtual void okay();
protected:
    Interactor* Interior();
protected:
    ButtonState* the_choice_BS;
    ButtonState* the_selected_BS;
    ButtonState* cancel_BS;
    ButtonState* okay_BS;
    Message* the_name;
    Message* the_choice_message;
    StringBrowser* choice_browser;
    Message* the_selected_message;
    StringBrowser* selected_browser;
    PushButton* the_cancel_button;
    PushButton* the_ok_button;
};

#endif

```

```

#ifndef dependancy_selector.h
#define dependancy_selector.h

#include "dependancy_selector-core.h"

class dependancy_selector : public dependancy_selector_core {
public:
    dependancy_selector(const char*);

    void Insert_components();
    virtual void new_selection();
    virtual void remove_selection();
    virtual void cancel();
    virtual void okay();
};

#endif

```

```

#ifndef error_viewer_core.h
#define error_viewer_core.h

#include <InterViews/scene.h>

class HBox;
class Message;
class TextEditor;
class ButtonState;

class error_viewer_core : public MonoScene {
public:
    error_viewer_core(const char*);
    virtual void ok_action();
protected:
    Interactor* Interior();
protected:
    ButtonState* ok_BS;
    HBox* the_file_name;
    Message* default_message;
    TextEditor* the_editor;
};

#endif

```

```

#ifndef error_viewer_h
#define error_viewer_h
#include <InterViews/textbuffer.h>
#include "error_viewer-core.h"

class error_viewer : public error_viewer_core {
private:
    TextBuffer *the_buffer;
public:
    error_viewer(const char*name, char *error_file);
    void ok_action();

};

#endif

```

```

#ifndef input_file_selector_core.h
#define input_file_selector_core.h

#include <InterViews/scene.h>

class StringEditor;
class ButtonState;
class FileBrowser;

class input_file_selector_core : public MonoScene {
public:
    input_file_selector_core(const char*);
    virtual void new_psdL_file_name();
    virtual void update_psdL_dir();
    virtual void psdL_selected();
    virtual void new_spec_file_name();
    virtual void update_ada_spec_dir();
    virtual void spec_selected();
    virtual void new_body_file_name();
    virtual void update_ada_body_dir();
    virtual void body_selected();
    virtual void cancel();
    virtual void okay();
protected:
    Interactor* Interior();
protected:
    ButtonState* psdL_file_name_BS;
    ButtonState* psdL_dir_BS;
    ButtonState* psdL_BS;
    ButtonState* spec_file_name_BS;
    ButtonState* ada_spec_dir_BS;
    ButtonState* spec_files_BS;
    ButtonState* body_file_name_BS;
    ButtonState* ada_body_dir_BS;
    ButtonState* body_files_BS;
    ButtonState* cancel_BS;
    ButtonState* okay_BS;
    StringEditor* psdL_file_name;
    StringEditor* psdL_dir;
    FileBrowser* psdL_files;
    StringEditor* spec_file_name;
    StringEditor* ada_spec_dir;
    FileBrowser* spec_files;
    StringEditor* body_file_name;
    StringEditor* ada_body_dir;
    FileBrowser* body_files;
};

#endif

```

```

#ifndef input_file_selector.h
#define input_file_selector.h

#include "input_file_selector-core.h"

class input_file_selector : public input_file_selector_core {
public:
    input_file_selector(const char*);

    virtual void new_psdL_file_name();
    virtual void update_pdsL_dir();
    virtual void psdL_selected();
    virtual void new_spec_file_name();
    virtual void update_ada_spec_dir();
    virtual void spec_selected();
    virtual void new_body_file_name();
    virtual void update_ada_body_dir();
    virtual void body_selected();
    virtual void cancel();
    virtual void okay();
};

#endif

```

```

#ifndef keyword_selector_core.h
#define keyword_selector_core.h

#include <InterViews/scene.h>

class Message;
class StringBrowser;
class ButtonState;
class PushButton;

class keyword_selector_core : public MonoScene {
public:
    keyword_selector_core(const char*);
    virtual void new_selection();
    virtual void remove_selection();
    virtual void cancel();
    virtual void okay();
protected:
    Interactor* Interior();
protected:
    ButtonState* the_choice_BS;
    ButtonState* the_selected_BS;
    ButtonState* cancel_BS;
    ButtonState* ok_BS;
    Message* the_name;
    Message* the_choice_message;
    StringBrowser* choice_browser;
    Message* the_selected_message;
    StringBrowser* selected_browser;
    PushButton* the_cancel_button;
    PushButton* the_ok_button;
};

#endif

```

```

#ifndef keyword_selector.h
#define keyword_selector.h

#include "keyword_selector-core.h"

class keyword_selector : public keyword_selector_core {
public:
    keyword_selector(const char*);

    void Insert_keywords();
    virtual void new_selection();
    virtual void remove_selection();
    virtual void cancel();
    virtual void okay();
};

#endif

```

```

#ifndef psdl_viewer_core.h
#define psdl_viewer_core.h

#include <InterViews/scene.h>

class HBox;
class Message;
class MenuBar;
class PulldownMenu;
class MenuItem;
class TextEditor;
class ButtonState;

class psdl_viewer_core : public MonoScene {
public:
    psdl_viewer_core(const char*);
    virtual void _Saveas();
    virtual void _Print();
    virtual void _Delete();
    virtual void _QuitView();
    virtual void _Find();
    virtual void view_spec_action();
protected:
    Interactor* Interior();
protected:
    HBox* the_file_name;
    Message* default_message;
    MenuBar* menu_bar;
    PulldownMenu* file_menu;
    MenuItem* view_spec;
    TextEditor* the_editor;
};

#endif

```

```

#ifndef psdl_viewer_h
#define psdl_viewer_h

#include "psdl_viewer-core.h"
#include <InterViews/textbuffer.h>
class psdl_viewer : public psdl_viewer_core
{
    private:
        char *component_name;
        char *psdl_file;
        char *spec_file;
        char *body_file;
        char *the_string;

        TextBuffer *the_buffer;

    public:
        psdl_viewer(const char*);

        virtual void _Saveas();
        virtual void _Print();
        virtual void _Delete();
        virtual void _QuitView();
        virtual void _Find();
        virtual void view_spec_action();
        virtual void Handle(Event&);
};

#endif

```

```

#ifndef query_file_dialog_core.h
#define query_file_dialog_core.h

#include <InterViews/dialog.h>

class StringEditor;
class ButtonState;
class FileBrowser;

class query_file_dialog_core : public Dialog {
public:
    query_file_dialog_core(const char*);

    virtual void file_name_action();
    virtual void directory_action();
    virtual void file_browser_action();
    virtual void cancel_action();
    virtual void okay_action();
protected:
    Interactor* Interior();
protected:
    ButtonState* dialog_BS;
    ButtonState* the_file_name_BS;
    ButtonState* directory_name_BS;
    ButtonState* file_browser_BS;
    ButtonState* cancel_BS;
    ButtonState* okay_BS;
    StringEditor* the_file_name;
    StringEditor* directory_name;
    FileBrowser* the_file_browser;
};

#endif

```

```

#ifndef query_file_dialog_h
#define query_file_dialog_h

#include "query_file_dialog-core.h"

class query_file_dialog : public query_file_dialog_core {
public:
    query_file_dialog(const char*);

    virtual void file_name_action();
    virtual void directory_action();
    virtual void file_browser_action();
    virtual void cancelAction();
    virtual void okay_action();
    const char* file_name();
};

#endif

```

```

#ifndef save_browser_dialog_core.h
#define save_browser_dialog_core.h

#include <Interviews/dialog.h>

class StringEditor;
class ButtonState;
class FileBrowser;

class save_browser_dialog_core : public Dialog {
public:
    save_browser_dialog_core(const char*);

    virtual void update_file_name();
    virtual void update_directory();
    virtual void file_browser_action();
    virtual void cancel_action();
    virtual void ok_action();
protected:
    Interactor* Interior();
protected:
    ButtonState* dialog_BS;
    ButtonState* file_name_BS;
    ButtonState* new_directory_BS;
    ButtonState* file_browser_BS;
    ButtonState* cancel_BS;
    ButtonState* ok_BS;
    StringEditor* the_file_name;
    StringEditor* the_directory_name;
    FileBrowser* the_file_browser;
};

#endif

```

```

#ifndef save_browser_dialog_h
#define save_browser_dialog_h

#include "save_browser_dialog-core.h"

class save_browser_dialog : public save_browser_dialog_core
{
private:
    char *default_name;

public:
    save_browser_dialog(const char*name);

    virtual void file_browser_action();
    virtual void update_file_name();
    virtual void update_directory();
    virtual void cancel_action();
    virtual void ok_action();
    const char *file_name();
};

#endif

```

```

#ifndef spec_viewer_core_h
#define spec_viewer_core_h

#include <InterViews/scene.h>

class HBox;
class Message;
class MenuBar;
class PulldownMenu;
class MenuItem;
class TextEditor;
class ButtonState;

class spec_viewer_core : public MonoScene {
public:
    spec_viewer_core(const char*);
    virtual void _Saveas();
    virtual void _Print();
    virtual void _QuitView();
    virtual void _Find();
    virtual void view_body_action();
protected:
    Interactor* Interior();
protected:
    HBox* the_file_name;
    Message* default_message;
    MenuBar* menu_bar;
    PulldownMenu* file_menu;
    MenuItem* view_body;
    TextEditor* the_editor;
};

#endif

```

```

#ifndef spec_viewer.h
#define spec_viewer.h

#include "spec_viewer-core.h"

class spec_viewer : public spec_viewer_core {
private:
    char *component_name;
    char *body_file;
    char *the_string;
    TextBuffer *the_buffer;

public:

    spec_viewer(const char*component_name, char*spec_file, char *body_file);

    virtual void _Saveas();
    virtual void _Print();
    virtual void _QuitView();
    virtual void _Find();
    virtual void view_body_action();
};

#endif

```

```

#include <InterViews/menu.h>
#include "SB_main_menu.h"
#include <InterViews/button.h>
#include <InterViews/interactor.h>
#include <InterViews/2.6/_enter.h>
#include <InterViews/world.h>
#include "keyword_selector.h"
#include "component_selector.h"
#include "input_file_selector.h"
#include "error_viewer.h"
#include <stream.h>
#include <strstream.h>
#include <stdlib.h>
#include <string.h>
#include "query_file_dialog.h"

```

```

#define TEMP_ENVIRONMENT "TEMP"
#define CAPS_ENVIRONMENT "CAPS"
#define DEFAULT_TEMP "/tmp"
#define TL_PREFIX "tmptl"
#define ERROR_PREFIX "tmperror"
#define SB_PROGRAM "sb"

```

```

SB_main_menu::SB_main_menu(const char* name) : SB_main_menu_core(name) {}

```

```

void SB_main_menu::quit()
{
    World* w=GetWorld();
    w->quit();
}

```

```

void SB_main_menu::browse_by_keyword()
{
    World* w=GetWorld();
    keyword_selector* the_keyword_selector=new keyword_selector("the_key_word_selector");
    the_keyword_selector->Insert_keywords();
    //the_keyword_selector->SetName("Keyword Selection");
    w->InsertToplevel(the_keyword_selector,this);
}

```

```

void SB_main_menu::browse_by_type()
{
    World* w=GetWorld();
    component_selector* component_selector_by_type=new
        component_selector("component_selector_by_type");
    // create list of components
    int command_status;
    ostrstream command_buffer;
    ostrstream remove_buffer;
}

```

```

char *caps_dir = getenv(CAPS_ENVIRONMENT);
char *temp_dir = getenv(TEMP_ENVIRONMENT);
if (temp_dir == NULL)
{
    temp_dir = new char[strlen(DEFAULT_TEMP) + 1];
    strcpy(temp_dir, DEFAULT_TEMP);
};

command_buffer << SB_PROGRAM << " tl ada ";
char *list_file = tempnam(temp_dir, TL_PREFIX);
char *error_file = tempnam(temp_dir, ERROR_PREFIX);
command_buffer << list_file << " > " << error_file << ends;

remove_buffer << "rm ";
remove_buffer << error_file << ends;

char *rm_command = remove_buffer.str();

char *command = command_buffer.str();
command_status=system(command);
if (command_status==0)
{
    // no error occured so pass the tl to the component selector
    component_selector_by_type->Insert_components(list_file);
    w->InsertApplication(component_selector_by_type);
}
else
{
    // display error info
    cerr << "AN ERROR OCCURED WITH COMMAND " << command << "\n";
};

// remove temp files
system(rm_command);
delete command;
delete rm_command;
};

void SB_main_menu::browse_by_operator()
{
    World* w=GetWorld();
    component_selector* component_selector_by_type=new
        component_selector("component_selector_by_type");
    // create list of components
    int command_status;

```

```

ostrstream command_buffer;
ostrstream remove_buffer;

char *caps_dir = getenv(CAPS_ENVIRONMENT);
char *temp_dir = getenv(TEMP_ENVIRONMENT);
if (temp_dir == NULL)
{
    temp_dir = new char[strlen(DEFAULT_TEMP) + 1];
    strcpy(temp_dir, DEFAULT_TEMP);
};

command_buffer << SB_PROGRAM << " ol ada ";
char *list_file = tempnam(temp_dir, TL_PREFIX);
char *error_file = tempnam(temp_dir, ERROR_PREFIX);
command_buffer << list_file << " > " << error_file << ends;

remove_buffer << "rm ";
remove_buffer << error_file << ends;

char *rm_command = remove_buffer.str();

char *command = command_buffer.str();
command_status=system(command);
if (command_status==0)
{
    // no error occured so pass the tl to the component selector
    component_selector_by_type->Insert_components(list_file);
    w->InsertApplication(component_selector_by_type);
}
else
{
    // display error info
    cerr << "AN ERROR OCCURED WITH COMMAND " << command << "\n";
};

// remove temp files
system(rm_command);
delete command;
delete rm_command;
}

void SB_main_menu::query()
{
    World*w=GetWorld();
    query_file_dialog* the_query_dialog=new
        query_file_dialog("the_query_dialog");
    w->InsertTransient(the_query_dialog,this);
}

```

```

boolean status_flag=the_query_dialog→Accept();
if(status_flag)
{
    w→sync();
    int command_status;
    ostream command_buffer;
    ostream remove_buffer;

    char *caps_dir = getenv(CAPS_ENVIRONMENT);
    char *temp_dir = getenv(TEMP_ENVIRONMENT);
    if (temp_dir == NULL)
    {
        temp_dir = new char[strlen(DEFAULT_TEMP) + 1];
        strcpy(temp_dir, DEFAULT_TEMP);
    };

    command_buffer << SB_PROGRAM << " cq ada ";
    char *list_file = tempnam(temp_dir, TL_PREFIX);
    char *error_file = tempnam(temp_dir, ERROR_PREFIX);
    command_buffer << the_query_dialog→file_name() << " ";
    command_buffer << list_file << " > " << error_file << ends;

    remove_buffer << "rm ";
    remove_buffer << error_file << ends;

    char *rm_command = remove_buffer.str();

    char *command = command_buffer.str();
    command_status=system(command);
    if (command_status==0)
    {
        // no error occurred so pass the tl to the component selector

        component_selector* component_selector_by_query=new
        component_selector("component_selector_by_query");

        component_selector_by_query→Insert_components(list_file);
        w→InsertApplication(component_selector_by_query);

    }
    else
    {
        // display error info
        error_viewer *error_view=new
        error_viewer("Add Component",error_file);
        w→InsertApplication(error_view);

    };
}

```

```

        // remove temp files
        system(rm_command);
        delete command;

        delete rm_command;
    }:
    w→Remove(the_query_dialog);
    delete the_query_dialog;

}

void SB_main_menu::AddComponent() {
    World* w=GetWorld();
    input_file_selector* in_file=new input_file_selector("in_file");
    w→InsertApplication(in_file);
}

void SB_main_menu::UpdateComponent()
{
    World* w=GetWorld();
    input_file_selector* in_file=new input_file_selector("in_file");
    w→InsertApplication(in_file);
};

void SB_main_menu::Help()
{
    system("doc -title 'Softbase Help' softbase_help.doc&");
}

```

```

#include <InterViews/button.h>
#include <InterViews/box.h>
#include <InterViews/message.h>
#include <InterViews/menu.h>
#include <InterViews/texteditor.h>
#include <InterViews/textbuffer.h>
#include "body_viewer.h"
#include "spec_viewer.h"
#include "body_viewer.h"
#include "save_browser_dialog.h"
#include <InterViews/world.h>
#include <InterViews/2.6/_enter.h>
#include <strstream.h>
#include <stream.h>
#include <stdlib.h>
#include <string.h>

body_viewer::body_viewer(const char* name, char *body_file) :
    body_viewer_core(name)
{
    component_name=(char *)name;
    ostrstream view_name_buffer;
    view_name_buffer << component_name << "Implementation Body" << ends;
    the_file_name←Remove(default_message);
    delete default_message;

    Message* the_file_message=new Message("file_name",
                                           view_name_buffer.str(),
                                           Center);

    the_file_name←Insert(the_file_message);
    the_file_name←Change();

    ifstream body(body_file);
    if (body)
    {
        ostrstream construct_buffer;
        while(!body.eof())
        {
            char text=body.get();
            if(text≠EOF)
            {
                construct_buffer.put(text);
            }
        };

        the_string=construct_buffer.str();
        int the_length=strlen(the_string);
        the_buffer=new TextBuffer(the_string,the_length,the_length);
        the_editor←Edit(the_buffer);
    }
    else
    {

```

```

        cerr << "UNABLE TO OPEN BODY FILE TO VIEW FILENAME IS ";
        cerr << body_file << "\n";
    };

}

void body_viewer::_Saveas()
{
    ostringstream def_name_buffer;
    def_name_buffer << component_name << ".body.a" << ends;
    char *def_name=def_name_buffer.str();
    save_browser_dialog *save_browser=new save_browser_dialog(def_name);
    World *w=GetWorld();
    w->InsertTransient(save_browser,this);
    boolean save_flag=save_browser->Accept();
    if(save_flag)
    {

        ofstream output_file(save_browser->file_name());
        output_file << the_string;
        output_file.close();

    };

    w->Remove(save_browser);
    delete save_browser;

}

void body_viewer::_Print() {
    /* unimplemented */
}

void body_viewer::_Find() {
    /* unimplemented */
}

void body_viewer::_QuitView()
{
    // remove temp file and remove application from the world
    World *w=GetWorld();
    w->Remove(this);
    //delete this;

}

```

```

#include <InterViews/button.h>
#include <InterViews/strbrowser.h>
#include "component_selector.h"
#include <InterViews/2.6/_enter.h>
#include <stream.h>
#include <string.h>
#include <InterViews/world.h>
#include "psdl_viewer.h"
#include <strstream.h>

extern "C"
{
    int system(char *);
};

#define MAX_NAME_LENGTH 256;

component_selector::component_selector(const char* name) : component_selector_core(name) {}

void component_selector::selected()
{
    int state_value;
    the_browser_BS->GetValue(state_value);
    if(state_value!=0)
    {
        World* w=GetWorld();
        w->sync();
        int selected_index=the_browser->Selection(0);
        if(selected_index>=0)
        {
            char *selected_buffer=the_browser->
                String(selected_index);
            char *selected_name=new char[strlen(selected_buffer)+1];
            int i=0;
            while(selected_buffer[i]!=' ')
            {
                selected_name[i]=selected_buffer[i];
                i++;
            };
            selected_name[i]=NULL;

            psdl_viewer* the_psdl_view=new
                psdl_viewer(selected_name);
            w->InsertApplication(the_psdl_view);

        }
        else
        {
            w->RingBell(100);
        };
        the_browser_BS->SetValue(0);
    }
}

```

```

    };
}

void component_selector::cancel()
{
    World* w=GetWorld();
    w→Remove(this);
    delete this;
};

void component_selector::okay()
{
    int state_value;
    the_ok_BS→GetValue(state_value);
    if(state_value≠0)
    {
        the_browser_BS→SetValue(1);
        the_ok_BS→SetValue(0);
    };
}

void component_selector::Insert_components()
{
    the_browser→Append("comp 1");
    the_browser→Append("comp 2");
    the_browser→Append("comp 3");
    the_browser→Append("comp 4");
    the_browser→Append("comp 5");
    the_browser→Append("comp 6");
    the_browser→Append("comp 7");
    the_browser→Append("comp 8");
    the_browser→Append("comp 9");
    the_browser→Append("comp 0");
};

void component_selector::Insert_components(char *file_name)
{
    char the_component[256];

    ifstream infile(file_name);

    infile >> ws;
    while(!infile.eof())
    {
        infile.getline(the_component,256);
        the_browser→Append(the_component);
        infile >> ws;
    };
}

```

```
    ostream command;  
    command << "rm " << file_name << ends;  
    char *the_command=command.str();  
    system(the_command);  
};
```

```
#include <InterViews/button.h>  
#include "delete_warning.h"  
#include <InterViews/2.6/_enter.h>
```

```
delete_warning::delete_warning(const char* name) : delete_warning_core(name) {}
```

```

#include <InterViews/message.h>
#include <InterViews/strbrowser.h>
#include <InterViews/button.h>
#include "dependancy_selector.h"
#include <InterViews/world.h>
#include <InterViews/2.6/_enter.h>

dependancy_selector::dependancy_selector(const char* name) : dependancy_selector_core(name)
{}

void dependancy_selector::Insert_components()
{
    choice_browser→Append("comp 1");
    choice_browser→Append("comp 2");
    choice_browser→Append("comp 3");
    choice_browser→Append("comp 4");
    choice_browser→Append("comp 5");
    choice_browser→Append("comp 6");
    choice_browser→Append("comp 7");
    choice_browser→Append("comp 8");
    choice_browser→Append("comp 9");
    choice_browser→Append("comp 0");
};

void dependancy_selector::new_selection()
{
    int state_value;
    the_choice_BS→GetValue(state_value);
    if(state_value≠0)
    {
        // get the new selection and check it against the
        // selected list if it is not there than add it
        int selected_index=choice_browser→Selection(0);
        char *new_selection=choice_browser→String(selected_index);
        if(selected_browser→Index(new_selection)<0)
        {
            // not in the selected browser so append it
            selected_browser→Append(new_selection);
            the_choice_BS→SetValue(0);
        };
    };
}

void dependancy_selector::remove_selection()
{
    int state_value;
    the_selected_BS→GetValue(state_value);
    if(state_value≠0)
    {
        // remove the selected entry from the browser

```

```

        int selected_index=selected_browser→Selection(0);
        selected_browser→Remove(selected_index);
        the_selected_BS→SetValue(0);
    };

}

void dependancy_selector::cancel()
{
    World* w=GetWorld();
    w→Remove(this);
    delete this;
}

void dependancy_selector::okay() {
    World* w=GetWorld();
    w→Remove(this);
    delete this;
}

```

```

#include "error_viewer.h"
#include <InterViews/button.h>
#include <InterViews/box.h>
#include <InterViews/message.h>
#include <InterViews/menu.h>
#include <InterViews/texteditor.h>
#include <InterViews/textbuffer.h>
#include <InterViews/world.h>
#include <InterViews/2.6/_enter.h>
#include <strstream.h>
#include <stream.h>
#include <stdlib.h>
#include <string.h>

error_viewer::error_viewer(const char* name, char *error_file) :
    error_viewer_core(name)
{
    ostrstream view_name_buffer;
    view_name_buffer << "Error Messages" << ends;
    the_file_name→Remove(default_message);
    delete default_message;

    Message* the_file_message=new Message("error_file",
                                           view_name_buffer.str(),
                                           Center);

    the_file_name→Insert(the_file_message);
    the_file_name→Change();

    ifstream error(error_file);
    if (error)
    {
        ostrstream construct_buffer;
        while(!error.eof())
        {
            char text=error.get();
            if(text≠EOF)
            {
                construct_buffer.put(text);
            }
        };
        char *the_string=construct_buffer.str();
        int the_length=strlen(the_string);
        the_buffer=new TextBuffer(the_string,the_length,the_length);
        the_editor→Edit(the_buffer);
    }
    else
    {
        cerr << "UNABLE TO OPEN BODY FILE TO VIEW FILENAME IS ";
        cerr << error_file << "\n";
    }
};

```

```
}  
  
void error_viewer::ok_action()  
{  
    World *w=GetWorld();  
    w->Remove(this);  
    //delete this;  
};
```

```

#include <InterViews/button.h>
#include <InterViews/streditor.h>
#include <InterViews/filebrowser.h>
#include "input_file_selector.h"
#include "error_viewer.h"
#include <InterViews/2.6/_enter.h>
#include <stream.h>
#include <string.h>
#include <strstream.h>
#include <InterViews/world.h>
#include <sys/param.h>
#include <stdlib.h>
extern "C"
{
    extern char *getwd(char *);

};

#define TEMP_ENVIRONMENT "TEMP"
#define CAPS_ENVIRONMENT "CAPS"
#define DEFAULT_TEMP "/tmp"
#define KWL_PREFIX "tmpk1"
#define ERROR_PREFIX "tmperror"
#define KWQ_PREFIX "tmpc1"
#define SB_PROGRAM "sb"

input_file_selector::input_file_selector(const char* name) : input_file_selector_core(name) {

    char pathname[MAXPATHLEN];
    getwd(pathname);
    psdl_files→SetDirectory(pathname);
    spec_files→SetDirectory(pathname);
    body_files→SetDirectory(pathname);
    psdl_dir→Message(psdl_files→GetDirectory());
    ada_spec_dir→Message(psdl_dir→Text());
    ada_body_dir→Message(psdl_dir→Text());

}

void input_file_selector::psdl_selected()
{
    int state_value;
    psdl_BS→GetValue(state_value);
    if(state_value≠0)
    {
        int index=psdl_files→Selection(0);
        if(psdl_files→IsADirectory(psdl_files→Path(index)))
        {
            psdl_dir→Message(psdl_files→Path(index));
            ada_spec_dir→Message(psdl_files→Path(index));
            ada_body_dir→Message(psdl_files→Path(index));
            psdl_files→SetDirectory(psdl_dir→Text());
        }
    }
}

```

```

        spec_files→SetDirectory(psdl_dir→Text());
        body_files→SetDirectory(psdl_dir→Text());
        psdl_file_name→Message("");
        spec_file_name→Message("");
        body_file_name→Message("");
    }
    else
    {

        psdl_file_name→Message(psdl_files→Path(index));
        // update the other file names if the correct ones exist
    };
    psdl_BS→SetValue(0);
};
}

void input_file_selector::update_pdsl_dir() {
    int state_value;
    psdl_dir_BS→GetValue(state_value);
    if(state_value≠0)
    {
        // the user has entered a directory name of his own
        const char *temp_directory=psdl_files→Normalize(psdl_dir→Text());
        char *new_directory=new char[strlen(temp_directory)+1];
        strcpy(new_directory,temp_directory);
        if(psdl_files→IsADirectory(new_directory))
        {
            psdl_dir→Message(new_directory);
            ada_spec_dir→Message(new_directory);
            ada_body_dir→Message(new_directory);
            psdl_files→SetDirectory(new_directory);
            spec_files→SetDirectory(new_directory);
            body_files→SetDirectory(new_directory);
        };
        delete new_directory;
        psdl_dir_BS→SetValue(0);
    };
}

void input_file_selector::spec_selected()
{
    int state_value;
    spec_files_BS→GetValue(state_value);
    if(state_value≠0)
    {
        int index=spec_files→Selection(0);
        if(spec_files→IsADirectory(spec_files→Path(index)))
        {
            ada_spec_dir→Message(spec_files→Path(index));
            ada_body_dir→Message(spec_files→Path(index));
            spec_files→SetDirectory(ada_spec_dir→Text());
            body_files→SetDirectory(ada_spec_dir→Text());
        }
    }
}

```

```

    }
    else
    {

        spec_file_name→Message(spec_files→Path(index));
        // update the other file names if the correct ones exist
    };
    spec_files_BS→SetValue(0);
};
}

void input_file_selector::update_ada_spec_dir()
{
    int state_value;
    ada_spec_dir_BS→GetValue(state_value);
    if(state_value≠0)
    {
        // the user has entered a directory name of his own
        const char *temp_directory=spec_files→Normalize(ada_spec_dir→Text());
        char *new_directory=new char[strlen(temp_directory)+1];
        strcpy(new_directory,temp_directory);
        if(spec_files→IsADirectory(new_directory))
        {
            ada_spec_dir→Message(new_directory);
            ada_body_dir→Message(new_directory);
            spec_files→SetDirectory(new_directory);
            body_files→SetDirectory(new_directory);
        };
        delete new_directory;
        ada_spec_dir_BS→SetValue(0);
    };
}

void input_file_selector::body_selected() {
    int state_value;
    body_files_BS→GetValue(state_value);
    if(state_value≠0)
    {
        int index=body_files→Selection(0);
        if(body_files→IsADirectory(body_files→Path(index)))
        {
            ada_body_dir→Message(body_files→Path(index));
            body_files→SetDirectory(ada_body_dir→Text());
        }
        else
        {

            body_file_name→Message(body_files→Path(index));
        };
        body_files_BS→SetValue(0);
    }
}

```

```

    };
}

void input_file_selector::update_ada_body_dir()
{
    int state_value;
    ada_body_dir_BS→GetValue(state_value);
    if(state_value≠0)
    {
        // the user has entered a directory name of his own
        const char *temp_directory=body_files→Normalize(ada_body_dir→Text());
        char *new_directory=new char[strlen(temp_directory)+1];
        strcpy(new_directory,temp_directory);
        if(body_files→IsADirectory(new_directory))
        {
            ada_body_dir→Message(new_directory);
            body_files→SetDirectory(new_directory);
        };
        delete new_directory;
        ada_body_dir_BS→SetValue(0);
    };
}

```

```

void input_file_selector::cancel()
{
    World *w=GetWorld();
    w→Remove(this);
    delete this;
}

```

```

void input_file_selector::okay()
{
    int command_status;
    World *w=GetWorld();

    char *psdl_file=NULL;
    char *spec_file=NULL;
    char *body_file=NULL;
    ostream psdl_file_buffer;
    ostream spec_file_buffer;
    ostream body_file_buffer;
    // verify that all of the selected items are files
    if(psdl_files→Selections()>0 && body_files→Selections() > 0 &&
        spec_files→Selections() > 0)
    {
        // something has been selected in each browser
        // so see if they are all valid files

        int psdl_file_num=psdl_files→Selection(0);
        int spec_file_num=spec_files→Selection(0);
        int body_file_num=body_files→Selection(0);
        if(!psdl_files→IsADirectory(psdl_files→Path(psdl_file_num)))

```

```

{
    // not a directory so get the file_name
    psdl_file_buffer << psdl_files→Path(psdl_file_num);
    psdl_file_buffer << ends;
    psdl_file=psdl_file_buffer.str();
};
if(!spec_files→IsADirectory(spec_files→Path(spec_file_num)))
{
    // not a directory so get the file_name
    spec_file_buffer << spec_files→Path(spec_file_num);
    spec_file_buffer << ends;
    spec_file=spec_file_buffer.str();
};
if(!body_files→IsADirectory(body_files→Path(body_file_num)))
{
    // not a directory so get the file_name
    body_file_buffer << body_files→Path(body_file_num);
    body_file_buffer << ends;
    body_file=body_file_buffer.str();
};

if(spec_file≠NULL && psdl_file≠NULL && body_file≠NULL)
{
    // all were valid files so process the addition
    char *temp_dir=getenv(TEMP_ENVIRONMENT);

    if(temp_dir==NULL)
    {
        temp_dir=new char[strlen(DEFAULT_TEMP)+1];
        strcpy(temp_dir,DEFAULT_TEMP);
    };

    char *error_file=tempnam(temp_dir,"tmpa");
    ostream command_buffer;
    command_buffer << SB.PROGRAM << " ca ada ";
    command_buffer << psdl_file;
    command_buffer << " " << spec_file;
    command_buffer << " " << body_file;
    command_buffer << " > " << error_file << ends;

    ostream rm_buffer;

    rm_buffer << "rm " << error_file << ends;

    char *command=command_buffer.str();

    command_status=system(command);
    if(command_status≠0)
    {
        // display error info
        error_viewer *error_view=new

```

```

        error_viewer("Add Component",error_file);
        w->InsertApplication(error_view);
        char *rm=rm_buffer.str();
        system(rm);
    }
    else
    {
        char *rm=rm_buffer.str();

        system(rm);
    };

    w->Remove(this);
    delete this;

}
else
{
    w->RingBell(100);

};

}
else
{
    w->RingBell(100);

};

}

void input_file_selector::new_psd_file_name()
{}

void input_file_selector::new_spec_file_name()
{}

void input_file_selector::new_body_file_name()
{}

```

```

#include <InterViews/button.h>
#include <InterViews/message.h>
#include <InterViews/strbrowser.h>
#include "keyword_selector.h"
#include <InterViews/2.6/_enter.h>
#include <stream.h>
#include "component_selector.h";
#include <InterViews/world.h>
#include <strstream.h>
#include <stdlib.h>
#include <string.h>

#define TEMP_ENVIRONMENT "TEMP"
#define CAPS_ENVIRONMENT "CAPS"
#define DEFAULT_TEMP "/tmp"
#define KWL_PREFIX "tmpkl"
#define ERROR_PREFIX "tmperror"
#define KWQ_PREFIX "tmpql"
#define SB_PROGRAM "sb"

keyword_selector::keyword_selector(const char *name):keyword_selector_core(name)
{
}

void keyword_selector::
    okay()
{
    int command_status;
    // check buttonstate for a 0 value

    World *w = GetWorld();
    // check to ensure that at least 1 keyword was selected
    if (selected_browser->Count() > 0)
    {

        ostrstream command_buffer;
        ostrstream remove_buffer;
        char *caps_dir = getenv(CAPS_ENVIRONMENT);
        char *temp_dir = getenv(TEMP_ENVIRONMENT);
        if (temp_dir == NULL)
        {
            temp_dir = new char[strlen(DEFAULT_TEMP) + 1];
            strcpy(temp_dir, DEFAULT_TEMP);
        };

        char *list_file = tmpnam(temp_dir, KWL_PREFIX);
        ofstream output(list_file);
        if (output)

```

```

{
    // opened the kwl file fine
    int i;
    for (i = 0; i ≤ selected_browser→Count(); i++)
    {

        // output each kw selected to the file
        output << selected_browser→String(i) << "\n";
    };
    output.close();

    w→Remove(this);

    char *result_file = tempnam(temp_dir, KWQ_PREFIX);
    char *error_file = tempnam(temp_dir, ERROR_PREFIX);
    command_buffer << SB_PROGRAM << " kwq ada " << list_file << " ";
    command_buffer << result_file << " > " << error_file << ends;

    remove_buffer << "rm ";
    remove_buffer << list_file << " " << error_file;
    remove_buffer << ends;

    char *rm_command = remove_buffer.str();

    char *command = command_buffer.str();

    command_status=system(command);
    component_selector *component_selector_by_keyword;
    if (command_status≠0)
    {

        cerr << "AN ERROR OCCURED WITH COMMAND " << command << "\n";

    }
    else
    {
        // no error occured so create the component selector
        component_selector_by_keyword = new
        component_selector("component_selector_by_keyword");
        // pass the result stream to selector to process
        component_selector_by_keyword→
        Insert_components(result_file);
    };

    // remove temp files
    system(rm_command);
    delete command;

```

```

        delete rm_command;

        w→InsertApplication(component_selector_by_keyword);
        delete this;
    }
    else
    {
        cerr << "UNABLE TO OPEN THE kwl FILE\n";
        w→Remove(this);
    };
}
else
{
    w→RingBell(100);
    // reset the button state
};

};

void keyword_selector::
    cancel()
{
    World *w = GetWorld();
    w→Remove(this);
    delete this;
};

void keyword_selector::
    new_selection()
{
    int state_value;
    the_choice_BS→GetValue(state_value);
    if (state_value ≠ 0)
    {
        // get the new selection and check it against the
        // selected list if it is not there than add it
        int selected_index = choice_browser→Selection(0);
        char *new_selection = choice_browser→String(selected_index);
        if (selected_browser→Index(new_selection) < 0)
        {
            // not in the selected browser so append it
            selected_browser→Append(new_selection);
            the_choice_BS→SetValue(0);
        };
    };
}

void keyword_selector::
    remove_selection()

```

```

{
    int state_value;
    the_selected_BS→GetValue(state_value);
    if (state_value ≠ 0)
    {
        // remove the selected entry from the browser
        int selected_index = selected_browser→Selection(0);
        selected_browser→Remove(selected_index);
        the_selected_BS→SetValue(0);
    };
}

void keyword_selector::
    Insert_keywords()
{
    int command_status;
    ostream command_buffer;
    ostream remove_buffer;

    char *caps_dir = getenv(CAPS_ENVIRONMENT);
    char *temp_dir = getenv(TEMP_ENVIRONMENT);
    if (temp_dir == NULL)
    {
        temp_dir = new char[strlen(DEFAULT_TEMP) + 1];
        strcpy(temp_dir, DEFAULT_TEMP);
    };

    command_buffer << SB_PROGRAM << " kwl ada ";
    char *list_file = tempnam(temp_dir, KWL_PREFIX);
    char *error_file = tempnam(temp_dir, ERROR_PREFIX);
    command_buffer << list_file << " > " << error_file << ends;

    remove_buffer << "rm ";
    remove_buffer << list_file << " " << error_file << ends;

    char *rm_command = remove_buffer.str();

    char *command = command_buffer.str();
    command_status=system(command);
    if (command_status==0)
    {
        // no error occured so read in the kwl
        ifstream kwf(list_file);
        if (kwf)
        {
            char *next_keyword = new char[256];
            // file opened fine
            while (!kwf.eof())
            {
                kwf.getline(next_keyword,256);
                kwf >> ws;
            }
        }
    }
}

```

```

        choice_browser→Append(next_keyword);
    };

    }
    else
    {
        cerr << "UNABLE TO OPEN OUTPUT FILE FOR COMMAND ";
        cerr << command << "\n";
    };
}
else
{

    // display error info
    cerr << "AN ERROR OCCURED WITH COMMAND " << command << "\n";
};

// remove temp files
system(rm_command);
delete command;
delete rm_command;

};

```

```

#include <InterViews/button.h>
#include <InterViews/box.h>
#include <InterViews/message.h>
#include <InterViews/menu.h>
#include <InterViews/texteditor.h>
#include <InterViews/textbuffer.h>
#include <InterViews/world.h>
#include "psdl_viewer.h"
#include "spec_viewer.h"
#include "save_browser_dialog.h"
#include "delete_warning.h"
#include <string.h>
#include <InterViews/2.6/_enter.h>
#include <ostream.h>
#include <stream.h>
#include <stdlib.h>

```

```

#define TEMP_ENVIRONMENT "TEMP"
#define CAPS_ENVIRONMENT "CAPS"
#define DEFAULT_TEMP "/tmp"
#define PSDL_PREFIX "tmpvp"
#define SPEC_PREFIX "tmpvs"
#define BODY_PREFIX "tmpvb"
#define ERROR_PREFIX "tmperror"
#define SB_PROGRAM "sb"

```

```

psdl_viewer::psdl_viewer(const char* name) : psdl_viewer_core(name)
{
    component_name=(char *)name;
    ostream view_name_buffer;
    view_name_buffer << name << "PSDL Specification" << ends;
    the_file_name→Remove(default_message);
    delete default_message;

    Message* the_file_message=new Message("file_name",
                                           view_name_buffer.str(),
                                           Center);

    the_file_name→Insert(the_file_message);
    the_file_name→Change();
    // get the files from the softbase and open the psdl file

    int command_status;
    ostream command_buffer;
    ostream remove_buffer;

    char *caps_dir = getenv(CAPS_ENVIRONMENT);
    char *temp_dir = getenv(TEMP_ENVIRONMENT);
    if (temp_dir == NULL)
    {
        temp_dir = new char[strlen(DEFAULT_TEMP) + 1];
    }

```

```

    strcpy(temp_dir, DEFAULT_TEMP);
};

command_buffer << SB_PROGRAM << " cv ada " << name;

    psdl_file = tempnam(temp_dir, PSDL_PREFIX);
    spec_file = tempnam(temp_dir, SPEC_PREFIX);
body_file = tempnam(temp_dir, SPEC_PREFIX);

char *error_file = tempnam(temp_dir, ERROR_PREFIX);

command_buffer << " " << psdl_file << " " << spec_file;
command_buffer << " " << body_file;

command_buffer << " > " << error_file << ends;
remove_buffer << "rm " << error_file << ends;

char *command = command_buffer.str();
command_status=system(command);
if (command_status==0)
{
    // no error occurred so read in the psdl file
    ifstream psdl(psdl_file);
    if (psdl)
    {
        ostringstream construct_buffer;
        while(!psdl.eof())
        {
            char text=psdl.get();
            if(text!=EOF)
            {
                construct_buffer.put(text);
            }
        };
        the_string=construct_buffer.str();
        int the_length=strlen(the_string);
        the_buffer=new TextBuffer(the_string,the_length,the_length);
        the_editor->Edit(the_buffer);
    }
    else
    {
        cerr << "UNABLE TO OPEN PSDL FILE FOR COMMAND ";
        cerr << command << "\n";
    };
}
else
{
    // display error info
    cerr << "AN ERROR OCCURED WITH COMMAND " << command << "\n";
}

```

```

    };

    // remove temp files
    char *remove_command=remove_buffer.str();
    system(remove_command);
    delete remove_command;
    delete command;
}

void psdl_viewer::_Saveas()
{
    ostringstream def_name_buffer;
    def_name_buffer << component_name << ".psd1" << ends;
    char *def_name=def_name_buffer.str();
    save_browser_dialog *save_browser=new save_browser_dialog(def_name);
    World *w=GetWorld();
    w->InsertTransient(save_browser,this);
    boolean save_flag=save_browser->Accept();
    if(save_flag)
    {
        ofstream output_file(save_browser->file_name());
        output_file << the_string;
        output_file.close();
    };

    w->Remove(save_browser);
    delete save_browser;
}

void psdl_viewer::_Print() {
    /* unimplemented */
}

void psdl_viewer::_Delete()
{
    World* w=GetWorld();

    delete_warning *warning=new delete_warning("the_warning");
    w->InsertTransient(warning,this);
    if(warning->Accept())
    {
        // create list of components
        int command_status;
    }
}

```

```

ostrstream command_buffer;
ostrstream remove_buffer;

char *caps_dir = getenv(CAPS_ENVIRONMENT);
char *temp_dir = getenv(TEMP_ENVIRONMENT);
if (temp_dir == NULL)
{
    temp_dir = new char[strlen(DEFAULT_TEMP) + 1];
    strcpy(temp_dir, DEFAULT_TEMP);
};

command_buffer << SB_PROGRAM << " cd ada ";
command_buffer << component_name << " ";
char *error_file = tempnam(temp_dir, ERROR_PREFIX);
command_buffer << " > " << error_file << ends;

remove_buffer << "rm ";
remove_buffer << error_file << ends;

char *rm_command = remove_buffer.str();

char *command = command_buffer.str();

// put dialog here to ensure this is what you want

command_status=system(command);
if (command_status!=0)
{
    // display error info
    cerr << "AN ERROR OCCURED WITH COMMAND " << command << "\n";
};

// remove temp files
system(rm_command);
delete command;
delete rm_command;

};
w->Remove(warning);
delete warning;
}

void psdl_viewer::_QuitView()
{
    World *w=GetWorld();
    w->Remove(this);
    ostrstream remove_buffer;
    remove_buffer << "rm " << psdl_file << " " << spec_file;
    remove_buffer << " " << body_file << ends;

```

```

        char *remove=remove_buffer.str();
        system(remove);

        delete the_string;
        delete the_buffer;
        //delete this;
    }

    void psdl_viewer::_Find() {
        /* unimplemented */
    }

    void psdl_viewer::view_spec_action()
    {
        World *w=GetWorld();
        spec_viewer *new_view=new spec_viewer(component_name,
                                                spec_file,
                                                body_file);

        w->InsertApplication(new_view);
    }

    void psdl_viewer::Handle (Event& e)
    {
    };

```

```

#include <InterViews/button.h>
#include <InterViews/streditor.h>
#include <InterViews/filebrowser.h>
#include <InterViews/world.h>
#include "query_file_dialog.h"
#include <InterViews/2.6/_enter.h>
#include <stream.h>
#include <string.h>
#include <sys/param.h>
#include <stdlib.h>
#include <strstream.h>
extern "C"
{
    extern char *getwd(char *);
};

query_file_dialog::query_file_dialog(const char* name) : query_file_dialog_core(name)
{
    char pathname[MAXPATHLEN];
    getwd(pathname);
    the_file_browser->SetDirectory(pathname);
    directory_name->Message(the_file_browser->GetDirectory());
}

void query_file_dialog::file_name_action()
{
    World *w=GetWorld();

    // ensure file does not already exist
    int state_value;
    the_file_name_BS->GetValue(state_value);
    if(state_value!=0)
    {
        int the_index=the_file_browser->Index(the_file_name->Text());
        if(the_index >= 0)
        {
            w->RingBell(100);
        }
        else
        {
            // update the save browser button to 1 to indicate
            // success

            the_file_browser->Select(the_index);
            dialog_BS->SetValue(1);
        }
    }
};

```

```

        the_file_name_BS→SetValue(0);
    };

}

void query_file_dialog::directory_action()
{
    World *w=GetWorld();
    int state_value;
    directory_name_BS→GetValue(state_value);
    if(state_value≠0)
    {
        // the user has entered a directory name of his own
        const char
*temp_directory=the_file_browser→Normalize(directory_name→Text());
        char *new_directory=new char[strlen(temp_directory)+1];
        strcpy(new_directory,temp_directory);
        if(the_file_browser→IsADirectory(new_directory))
        {
            directory_name→Message(new_directory);
            the_file_browser→SetDirectory(new_directory);
        }
        else
        {
            w→RingBell(100);
        };
        delete new_directory;
        directory_name_BS→SetValue(0);
    };
}

void query_file_dialog::file_browser_action()
{
    int state_value;
    file_browser_BS→GetValue(state_value);
    if(state_value≠0)
    {
        int index=the_file_browser→Selection(0);
        if(the_file_browser→IsADirectory(the_file_browser→Path(index)))
        {
            directory_name→Message(the_file_browser→Path(index));
            the_file_browser→SetDirectory(directory_name→Text());
        }
        else
        {
            the_file_name→Message(the_file_browser→String(index));
            dialog_BS→SetValue(1);
        };
    };
}

```

```

        file_browser_BS→SetValue(0);
    };
}

void query_file_dialog::cancel_action()
{
    int state_value;
    cancel_BS→GetValue(state_value);
    if(state_value≠0)
    {
        // a value other than 1 indicates false
        dialog_BS→SetValue(2);
        cancel_BS→SetValue(0);
    };
};

void query_file_dialog::okay_action()
{
    World *w=GetWorld();
    int state_value;
    okay_BS→GetValue(state_value);
    if(state_value≠0)
    {
        int the_index=the_file_browser→Index(the_file_name→Text());
        if(the_index < 0)
        {
            // file already does not exists
            w→RingBell(100);
        }
        else
        {
            // update the save browser button to 1 to indicate
            // success
            dialog_BS→SetValue(1);
        };
        okay_BS→SetValue(0);
    };
};

const char *query_file_dialog::file_name()
{
    ostringstream full_file_name;
    full_file_name << the_file_browser→GetDirectory();
    full_file_name << the_file_name→Text();

    return full_file_name.str();
}

```

};

```

#include <InterViews/button.h>
#include <InterViews/streditor.h>
#include <InterViews/filebrowser.h>
#include <InterViews/world.h>
#include "save_browser_dialog.h"
#include <InterViews/2.6/_enter.h>
#include <stream.h>
#include <string.h>
#include <sys/param.h>
#include <stdlib.h>
#include <strstream.h>
extern "C"
{
    extern char *getwd(char *);
};

save_browser_dialog::save_browser_dialog(const char* name) : save_browser_dialog_core(name)
{
    default_name=(char *)name;
    char pathname[MAXPATHLEN];
    getwd(pathname);
    the_file_browser->SetDirectory(pathname);
    the_directory_name->Message(the_file_browser->GetDirectory());
    the_file_name->Message(default_name);
}

void save_browser_dialog::file_browser_action()
{
    int state_value;
    file_browser_BS->GetValue(state_value);
    if(state_value!=0)
    {
        int index=the_file_browser->Selection(0);
        if(the_file_browser->IsADirectory(the_file_browser->Path(index)))
        {
            the_directory_name->Message(the_file_browser->Path(index));
            the_file_browser->SetDirectory(the_directory_name->Text());
        };

        file_browser_BS->SetValue(0);
    };
}

void save_browser_dialog::update_file_name()
{
    World *w=GetWorld();

    // ensure file does not already exist

```

```

    int state_value;
    file_name_BS→GetValue(state_value);
    if(state_value≠0)
    {
        int the_index=the_file_browser→Index(the_file_name→Text());
        if(the_index ≥ 0)
        {
            // file already exists so do not accept it
            w→RingBell(100);
        }
        else
        {
            // update the save browser button to 1 to indicate
            // success
            dialog_BS→SetValue(1);

        }
        file_name_BS→SetValue(0);
    };
};

}

void save_browser_dialog::update_directory()
{
    World *w=GetWorld();
    int state_value;
    new_directory_BS→GetValue(state_value);
    if(state_value≠0)
    {
        // the user has entered a directory name of his own
        const char
        *temp_directory=the_file_browser→Normalize(the_directory_name→Text());
        char *new_directory=new char[strlen(temp_directory)+1];
        strcpy(new_directory,temp_directory);
        if(the_file_browser→IsADirectory(new_directory))
        {
            the_directory_name→Message(new_directory);
            the_file_browser→SetDirectory(new_directory);
        }
        else
        {
            w→RingBell(100);
        };
        delete new_directory;
        new_directory_BS→SetValue(0);
    };
};

}

void save_browser_dialog::cancel_action()
{

```

```

    int state_value;
    cancel_BS→GetValue(state_value);
    if(state_value≠0)
    {
        // a value other than 1 indicates false
        dialog_BS→SetValue(2);
        cancel_BS→SetValue(0);
    };
}

void save_browser_dialog::ok_action()
{
    World *w=GetWorld();
    int state_value;
    ok_BS→GetValue(state_value);
    if(state_value≠0)
    {
        int the_index=the_file_browser→Index(the_file_name→Text());
        if(the_index ≥ 0)
        {
            // file already exists so do not accept it
            w→RingBell(100);
        }
        else
        {
            // update the save browser button to 1 to indicate
            // success
            dialog_BS→SetValue(1);
        };
        ok_BS→SetValue(0);
    };
}

const char *save_browser_dialog::file_name()
{
    ostringstream full_file_name;
    full_file_name << the_file_browser→GetDirectory();
    full_file_name << the_file_name→Text();

    return full_file_name.str();
};

```

```

#include <InterViews/painter.h>
#include <InterViews/shape.h>
#include <InterViews/sensor.h>
#include <InterViews/world.h>
#include "SB_main_menu.h"
#include <InterViews/2.6/_enter.h>

static PropertyData properties[] = {
#include "softbase-props"
    { "*title", "CAPS SOFTBASE" },
    { nil }
};

static OptionDesc options[] = {
    { nil }
};

int main (int argc, char** argv) {
    World* w = new World("****", argc, argv, options, properties);
    SB_main_menu* the_main_menu = new SB_main_menu("the_main_menu");
    w->InsertApplication(the_main_menu);

    w->Run();
    delete w;
    return 0;
}

```

```

#include <InterViews/button.h>
#include <InterViews/box.h>
#include <InterViews/message.h>
#include <InterViews/menu.h>
#include <InterViews/texteditor.h>
#include <InterViews/textbuffer.h>
#include <InterViews/world.h>
#include "spec_viewer.h"
#include "psdl_viewer.h"
#include "body_viewer.h"
#include "save_browser_dialog.h"
#include <InterViews/2.6/_enter.h>
#include <strstream.h>
#include <stream.h>
#include <stdlib.h>
#include <string.h>

spec_viewer::spec_viewer(const char* name, char *spec_file, char *body_file_in) :
    spec_viewer_core(name)
{
    component_name=(char *)name;
    body_file=body_file_in;
    ostrstream view_name_buffer;
    view_name_buffer << component_name << "Implementation Specification" << ends;
    the_file_name→Remove(default_message);
    delete default_message;

    Message* the_file_message=new Message("file_name",
                                           view_name_buffer.str(),
                                           Center);

    the_file_name→Insert(the_file_message);
    the_file_name→Change();

    ifstream spec(spec_file);
    if (spec)
    {
        ostrstream construct_buffer;
        while(!spec.eof())
        {
            char text=spec.get();
            if(text≠EOF)
            {
                construct_buffer.put(text);
            }
        };
        the_string=construct_buffer.str();
        int the_length=strlen(the_string);
        the_buffer=new TextBuffer(the_string,the_length,the_length);
        the_editor→Edit(the_buffer);
    }
    else

```

```

        {
            cerr << "UNABLE TO OPEN SPEC FILE TO VIEW FILENAME IS ";
            cerr << spec_file << "\n";
        };
    }

void spec_viewer::Saveas()
{
    ostringstream def_name_buffer;
    def_name_buffer << component_name << ".spec.a" << ends;
    char *def_name=def_name_buffer.str();
    save_browser_dialog *save_browser=new save_browser_dialog(def_name);
    World *w=GetWorld();
    w->InsertTransient(save_browser,this);
    boolean save_flag=save_browser->Accept();
    if(save_flag)
    {
        ofstream output_file(save_browser->file_name());
        output_file << the_string;
        output_file.close();
    };

    w->Remove(save_browser);
    delete save_browser;
}

void spec_viewer::Print() {
    /* unimplemented */
}

void spec_viewer::QuitView()
{
    // remove temporary file and remove application from world
    World *w=GetWorld();
    w->Remove(this);
    //delete this;
}

void spec_viewer::Find() {
    /* unimplemented */
}

void spec_viewer::view_body_action()
{
    World *w=GetWorld();
    body_viewer *new_view=new body_viewer(component_name,

```

```
        body_file);  
w→InsertApplication(new_view);  
}
```

BIBLIOGRAPHY

1. Bayramoglu, S., *The Design and Implementation of an Expander for Hierarchical Real-Time Constraints of Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, September, 1991.
2. Biggerstaff, T. J. and Perlis, A. J., *Software Reusability Volume I Concepts and Models*, ACM Press, New York, NY, 1989.
3. Biggerstaff, T. J. and Perlis, A. J., *Software Reusability Volume II Applications and Experience*, ACM Press, New York, NY, 1989.
4. Booch, G., *Object Oriented Design With Applications*, The Benjamin/Cummings Publishing Company, Inc., 1991.
5. Dwyer, A. P. and Lewis, G. W., *The Development of a Design Database for the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, September 1991.
6. Elmasri, R. and Navathe, S., *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company, Inc., 1989.
7. Gough, K. J., *Syntax Analysis and Software Tools*, Addison-Wesley Publishing Company, 1988.
8. Levine, J., *An Efficient Heuristic Scheduler for Hard Real-Time Systems*, M.S. Thesis, Naval Postgraduate School, Monterey, CA, September, 1991.

INITIAL DISTRIBUTION LIST

- | | |
|--|---|
| 1. Defense Technical Information Center
Cameron Station
Alexandria, VA 22304-6145 | 2 |
| 2. Dudley Knox Library
Code 52
Naval Postgraduate School
Monterey, CA 93943 | 2 |
| 3. Computer Science Department
Code CS
Naval Postgraduate School
Monterey, CA 93943 | 2 |
| 4. Office of the Assistant Secretary of the Navy
Research Development and Acquisition
Department of the Navy
Attn: Mr. Gerald A. Cann
Washington, DC 20380-1000 | 1 |
| 5. Office of the Chief of Naval Operations
OP-094
Department of the Navy
Attn: VADM J. O. Tuttle, USN
Washington, DC 20301-3040 | 1 |
| 6. Director of Defense Information
Office of the Assistant Secretary of Defense
(Command, Control, Communications, & Intelligence)
Attn: Mr. Paul Strassmann
Washington, DC 20301-0208 | 1 |
| 7. Center for Naval Analysis
4401 Ford Avenue
Alexandria, VA 22302-0268 | 1 |

- | | |
|---|----|
| 8. Director of Research Administration
Attn: Prof. Howard
Code 08Hk
Naval Postgraduate School
Monterey, CA 93943 | 1 |
| 9. Chairman, Code CS
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100 | 1 |
| 10. Prof. Luqi, Code CSLq
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943 | 10 |
| 11. Chief of Naval Research
800 N. Quincy Street
Arlington, VA 22217 | 1 |
| 12. Director, Ada Joint Program Office
OUSDRE (R&AT)
Room 3E114, The Pentagon
Attn: Dr. John P. Solomond
Washington, DC 20301-0208 | 1 |
| 13. Carnegie Mellon University
Software Engineering Institute
Attn: Dr. Dan Berry
Pittsburgh, PA 15260 | 1 |
| 14. Office of Naval Technology (ONT)
Code 227
Attn: Dr. Elizabeth Wald
800 N. Quincy St.
Arlington, VA 22217-5000 | 1 |
| 15. Defense Advanced Research Projects Agency (DARPA)
Integrated Strategic Technology Office (ISTO)
Attn: Dr. B. Boehm
1400 Wilson Boulevard
Arlington, VA 22209-2308 | 1 |

16. Defense Advanced Research Projects Agency (DARPA) 1
ISTO
1400 Wilson Boulevard
Attn: LCol Eric Mattala
Arlington, VA 2209-2308
17. Defense Advanced Research Projects Agency (DARPA) 1
Director, Tactical Technology Office
1400 Wilson Boulevard
Arlington, VA 2209-2308
18. Attn: Dr. Charles Harland 1
Computer Science
Department of the Air Force
Bolling Air Force Base, DC 20332-6448
19. Chief of Naval Operations 1
Attn: Dr. R. M. Carroll (OP-01B2)
Washington, DC 20350
20. Dr. Amiram Yehudai 1
Tel Aviv University
School of Mathematical Sciences
Department of Computer Science
Tel Aviv, Israel 69978
21. Dr. Robert M. Balzer 1
USC-Information Sciences Institute
4676 Admiralty Way
Suite 1001
Marina del Ray, CA 90292-6695
22. Dr. Ted Lewis 1
OR State University
Computer Science Department
Corvallis, OR 97331
23. International Software Systems Inc. 1
12710 Research Boulevard, Suite 301
Attn: Dr. R. T. Yeh
Austin, TX 78759

- | | |
|---|---|
| 24. Kestrel Institute
Attn: Dr. C. Green
1801 Page Mill Road
Palo Alto, CA 94304 | 1 |
| 25. National Science Foundation
Division of Computer and Computation Research
Attn: K. C. Tai
Washington, DC 20550 | 1 |
| 26. Commander Space and Naval Warfare Systems Command
SPAWAR 3212
Department of the Navy
Attn: Cdr M. Romeo
Washington, DC 20363-5100 | 1 |
| 27. Naval Ocean Systems Center
Attn: Linwood Sutton, Code 423
San Diego, CA 92152-5000 | 1 |
| 28. Office of Naval Research
Computer Science Division, Code 1133
Attn: Dr. Gary Koob
800 N. Quincy Street
Arlington, VA 22217-5000 | 1 |
| 29. Commander, Naval Sea Systems Command (PMS-4123H)
Attn: William Wilder
Washington, DC 20380-1000 | 1 |
| 30. New Jersey Institute of Technology
Computer Science Department
Attn: Dr. Peter Ng
Newark, NJ 07102 | 1 |
| 31. Office of Naval Research
Computer Science Division, Code 1133
Attn: Dr. A. M. Van Tilborg
800 N. Quincy Street
Arlington, VA 22217-5000 | 1 |

32. Office of Naval Research 1
Computer Science Division, Code 1133
Attn: Dr. R. Wachter
800 N. Quincy Street
Arlington, VA 22217-5000
33. OR Graduate Center 1
Portland (Beaverton)
Attn: Dr. R. Kieburtz
Portland, OR 97005
34. Santa Clara University 1
Department of Electrical Engineering and
Computer Science
Attn: Dr. M. Ketabchi
Santa Clara, CA 95053
35. Software Group, MCC 1
9430 Research Boulevard
Attn: Dr. L. Belady
Austin, TX 78759
36. University of CA at Berkeley 1
Department of Electrical Engineering and
Computer Science
Computer Science Division
Attn: Dr. C.V. Ramamoorthy
Berkeley, CA 90024
37. University of CA at Irvine 1
Department of Computer and Information Science
Attn: Dr. Nancy Leveson
Irvine, CA 92717
38. Chief of Naval Operations 1
Attn: Dr. Earl Chavis (OP-16T)
Washington, DC 20350
39. Office of the Chief of Naval Operations 1
Attn: Dr. John Davis (OP-094H)
Washington, DC 20350-2000

40. University of Illinois 1
Department of Computer Science
Attn: Dr. Jane W. S. Liu
Urbana Champaign, IL 61801
41. University of MD 1
College of Business Management
Tydings Hall, Room 0137
Attn: Dr. Alan Hevner
College Park, MD 20742
42. University of MD 1
Computer Science Department
Attn: Dr. N. Roussapoulos
College Park, MD 20742
43. University of Massachusetts 1
Department of Computer and Information Science
Attn: Dr. John A. Stankovic
Amherst, MA 01003
44. University of Pittsburgh 1
Department of Computer Science
Attn: Dr. Alfs Berztiss
Pittsburgh, PA 15260
45. University of TX at Austin 1
Computer Science Department
Attn: Dr. Al Mok
Austin, TX 78712
46. Commander, Naval Surface Warfare Center, 1
Code U-33
Attn: Dr. Philip Hwang
10901 New Hampshire Avenue
Silver Spring, MD 20903-5000
47. Attn: George Sumiall 1
US Army Headquarters
CECOM
AMSEL-RD-SE-AST-SE
Fort Monmouth, NJ 07703-5000

48. Attn: Joel Trimble 1
1211 South Fern Street, C107
Arlington, VA 22202
49. United States Laboratory Command 1
Army Research Office
Attn: Dr. David Hislop
P. O. Box 12211
Research Triangle Park, NC 27709-2211
50. George Mason University 1
Computer Science Department
Attn: Dr. David Rine
Fairfax, VA 22030-4444
51. Hewlett Packard Research Laboratory 1
Mail Stop 321
1501 Page Mill Road
Attn: Dr. Martin Griss
Palo Alto, CA 94304
52. Carnegie Mellon University 1
SEI
Attn: Dr. Mario Barbacci
Pittsburgh, PA 15213
53. Persistent Data Systems 1
75 W. Chapel Ridge Road
Attn: Dr. John Nester
Pittsburgh, PA 15238
54. Sun Microsystems Inc. 1
MS MTV100-01
Silicon Valley Government District
1842 N. Shoreline Boulevard
Attn: Vice President c/o Don Chandler
Mountain View, CA 94043
55. Commandant of the Marine Corps 1
Ada Joint Program Representative
Code CCI
Attn: Capt Gerald Depasquale
Washington, DC 20301

56. Ontologic, Inc.

1

Three Burlington Woods
Attn: Mr. Gregory Harris
Burlington, MA 01803