NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A SYNTAX DIRECTED EDITOR FOR THE COMPUTER AIDED PROTOTYPING SYSTEM

by

Charles A. Mock

September 1997

Thesis Advisor: Luqi

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1997	Master's Thesis

4. TITLE AND SUBTITLE
A SYNTAX DIRECTED EDITOR FOR THE COMPUTER AIDED PROTOTYPING SYSTEM.

5. FUNDING NUMBERS

6. AUTHOR(S)
Mock, Charles A.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The Computer Aided Prototyping System (CAPS) is an integrated set of software engineering tools developed at the Naval Postgraduate School (NPS). It is designed to support rapid prototyping of real-time systems. CAPS consists of four major subcomponents; the graphics/text editor, the user interface, the software database system, and the execution support system. Reports from users of CAPS, particularly novices, indicated that the clumsy and unintuitive multi-windowed graphics/text editor present in the system hampered the use of the tool set. This thesis presents the substitution and integration of an efficient and user-friendly syntax directed editor into CAPS. The new syntax directed editor consists of a package of seven Ada95 parsers that recognize the elements of the Prototype System Description Language (PSDL) and an enhanced C\Motif based graphics editor. These modules combine the functionality of all the windows of the graphics/text editor into one window, using pop-up boxes and menus to guide the designer in providing the proper information. During integration, particular attention was paid to ensuring the proper manipulation of data was occurring between modules and the internal consistency was being maintained at the inter-language interfaces. The result is a faster, intuitive, and more efficient designer interface.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Syntax Directed Editor, Computer Aided Prototyping	177
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

A SYNTAX DIRECTED EDITOR FOR THE COMPUTER AIDED PROTOTYPING SYSTEM

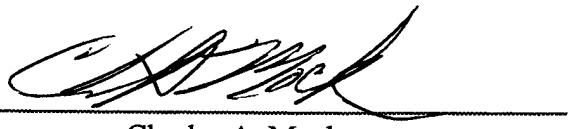
Charles A. Mock
Captain, United States Marine Corps
B.A., University of Florida, 1989

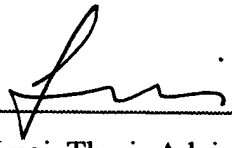
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

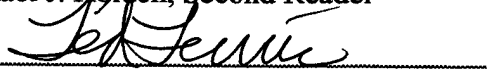
from the

NAVAL POSTGRADUATE SCHOOL
September 1997

Author: 
Charles A. Mock

Approved by: 
Luqi, Thesis Advisor


Michael J. Holden, Second Reader


Ted Lewis, Chairman
Department of Computer Science

ABSTRACT

The Computer Aided Prototyping System (CAPS) is an integrated set of software engineering tools developed at the Naval Postgraduate School (NPS). It is designed to support rapid prototyping of real-time systems. CAPS consists of four major subcomponents; the graphics/text editor, the user interface, the software database system, and the execution support system. Reports from users of CAPS, particularly novices, indicated that the clumsy and unintuitive multi-windowed graphics/text editor present in the system hampered the use of the tool set. This thesis presents the substitution and integration of an efficient and user-friendly syntax directed editor into CAPS. The new syntax directed editor consists of a package of seven Ada95 parsers that recognize the elements of the Prototype System Description Language (PSDL) and an enhanced C\Motif based graphics editor. These modules combine the functionality of all the windows of the graphics/text editor into one window, using pop-up boxes and menus to guide the designer in providing the proper information. During integration, particular attention was paid to ensuring the proper manipulation of data was occurring between modules and the internal consistency was being maintained at the inter-language interfaces. The result is a faster, intuitive, and more efficient designer interface.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

This research was possible due to the efforts of many people. Most notably is that of my thesis advisor, Professor Luqi and the members of the CAPS group. Professor Berzins' mentorship was truly instrumental in my understanding of the internal workings and interaction of the graphics editor and the syntax directed editor. The friendship and infallible cheerfulness of Professor Ying Jing made the long and arduous hours of coding bearable. I am also deeply indebted to Commander Mike Holden. Undoubtedly in the top 5% of instructors at this institute, he always places the student first. For his invaluable contributions to the organization and content of this paper, at a high cost to himself, I will always be grateful.

I would also like to thank the students and faculty members at the Naval Postgraduate School. I have found the educational experience to be a positive one and this is only possible with the outstanding faculty and peers with which this institution is blessed.

Finally, I am grateful to my wife Juli for her patience and understanding throughout this research. Without her support, none of this would have been possible.

I. INTRODUCTION

A. GOALS

The goals of this thesis are to produce an intuitive and efficient syntax directed graphical editor and document its integration into the Naval Postgraduate School's Computer Aided Prototyping System (CAPS).

CAPS operates using the Prototype System Design Language (PSDL), a design language created to present software designers with a tool for creating rapid prototypes. A syntax directed editor (SDE) for PSDL is a tool for editing PSDL programs, called prototypes. The editor checks each element of the PSDL language entered into the prototype by the designer for errors. This is done using a set of parsers designed to analyze text for specific words or strings of letters, known as tokens. If the tokens are correct for the language, the designer may proceed with the design, if not, the designer is alerted to the error and directed to the position of the mistake. In this way, an SDE can assist the designer in creating prototypes in a quick and effective manner.

CAPS is an integrated set of tools designed to allow designers to rapidly produce a prototype matching user specified requirements. Rapid prototyping allows the designer to clarify the needs and desires of the user early in the software development cycle. This eliminates costly errors that normally are not discovered until later, when significant effort is required for correction. The intent in inserting an updated designer interface is to compress the time necessary to create a working prototype by making CAPS easier to use. A successful integration of an enhanced syntax directed editor would make CAPS

more accessible to novices and allow experienced users to minimize time lost correcting errors during the prototyping cycle. This, in turn, will speed up the software development cycle.

B. BACKGROUND AND MOTIVATION

There is a problem with the creation of software today. It is costing the nation billions of dollars in failed projects and wasted efforts. Often the problem stems from a faulty design process. One solution to this problem, CAPS, is presently hampered by a poorly designed interface. Correcting this defect is one step toward solving the greater issue.

The well-know truism, "time is money", is especially relevant in the design of software. Certainly, the longer a project takes the higher the cost for labor and materials. The truth of this is quite obvious. The desire to reduce costs and meet deadlines invariably results in constricted timelines for identifying requirements, outlining specifications, and creating a solid design. The rush to implementation allows a product to be swiftly pushed out the door, but a cost must then be paid in user satisfaction and maintenance.

Time is money. As much as eighty percent of the cost involved in software production is maintenance [Ref. 4]. Yet that is not the least of the problem. In 1995, IEEE reported that for the Department of Defense (DOD) $42 billion in software programs were canceled due to overruns in time and budget or they simply did not perform as required [Ref. 4]. The waste of billions of dollars is not confined to the DOD.

Other prominent debacles include the automated baggage system at the Denver airport and the IRS integrated data network. While expensive, these systems are cheap compared to the seven billion dollar loss involved in the destruction of the Ariane rocket [Ref. 14] and the death of 225 passengers in the 6 August 1997 crash of a Boeing 747 on Guam [Ref. 16], both attributed to software errors.

Few of the problems with modern software systems involve poor coding. A NASA study performed during the Galileo project found that a full 98% of all software errors are traceable to incorrect requirements, specifications, or design [Ref. 4]. The advanced state of modern integrated development suites such as Aonix ObjectAda 7.1, Microsoft Visual Studio '97 and Borland C++ 5.0, combined with rigorous testing, have nearly eliminated implementation errors. Unfortunately, few tools exist for properly creating a design that faithfully represents the user's desire. CAPS is one of them.

Designed for rapid prototyping of real time systems, CAPS is aimed at reducing software design errors by presenting the user with successively more accurate representations of the final product. In order to do this effectively, CAPS itself must be fast and easy to use. That requires an easily learned, intuitive interface. That is the rationale for this thesis.

C. RESEARCH OBJECTIVES

The objectives for this thesis include the following specific objectives:

- Create parsers that correctly interpret the seven main elements of the Prototype System Design Language (PSDL).

- Automate coding of large segments of the project using existing UNIX environment tools and software engineering procedures

- Integrate the new parsers into a previously designed graphical editor.

- Integrate the combined graphical editor/parser package into the main flow of the CAPS environment.

- Document the flow of control and the design of data structures into a maintenance handbook to improve maintenance of this new enhanced version of CAPS.

D. ORGANIZATION

Chapter II provides a general overview of the workings of CAPS, PSDL and the need for and uses of rapid prototyping. Chapter III examines the processes and structures created during the analysis of the proposed evolution of CAPS for requirements, the specification of those requirements, and the creation of a design for the modifications to the CAPS code, with emphasis on the requirements for the appropriate interfaces, data manipulation, and improvements in storage management. Chapter IV presents the implementation phase of the project as well as testing and evaluation of the new version. In Chapter V the results are presented, with discussion on the utility of the improved interface and maintainability of CAPS. Chapter VI summarizes the thesis.

II. BACKGROUND

A. INTRODUCTION

The creation of a modern graphical editor for CAPS is a project that has been envisioned since 1991. By 1997 it became imperative that this goal become a reality for two major reasons.

The first is the state of software development. There is a software development crisis looming over the information age [Ref 8]. The rapid increase in the speed of computer hardware and the use of computer technology has not been matched by a corresponding improvement in software design. Software design has not only failed to keep pace, but has increasingly been the reason for late projects, exceeding budgets, and often complete failure of projects. This amounts to what was coined in 1968 as 'The Software Crisis'. This crisis has as its root cause the problem of complexity (brought about by sheer length of programs) combined with a poor control over how each line of code affects the overall system [Ref. 17]. These problems can often be corrected by using an alternative software development model. In many cases, rapid prototyping is a better method for designing software than current methodologies and CAPS is a powerful tool that could formalize the creation of software, if it achieves acceptance on a wide scale. Acceptance of CAPS on a wide scale requires a modern user-friendly interface.

The second reason is the participation of the Software Engineering Group in the DOD's Technology Transfer Program. The Department of Defense sponsors billions of dollars in advanced research yearly. In order to enable the various research laboratories

within the DOD to reap the maximum benefit from this effort, the Office of Technology Transition (OTT) was established. OTT serves as a clearinghouse for coordinating and facilitating the transition of such technologies and technological advancements within the DOD to other military organizations and the private sector. CAPS has been distributed throughout the DOD as a result of this program, and is in use by a large number of organizations. A number of these supporters have requested that improvements to the interface and in storage management be included with an enhanced version. In order for the enhanced version to be available for pending projects, its creation was necessary as soon as possible.

1. The Computer Revolution And The Software Crisis

The Information Age is here and is expanding at an extraordinary rate. The burgeoning use of computers and information systems in nearly every aspect of daily life is undeniable. Microprocessors and software programs inhabit our automobiles, videocassette recorders, coffee makers, and calculators. Fifty-one million Americans regularly access the Internet [Ref. 1] and it is the stated goal of the United States Government to place every classroom in the nation online. Riding the wave of computing power created by the realization of Moore's Law [Ref. 15], the use and scope of automated systems are increasing at a geometric rate.

Unfortunately, the same cannot be said for the state of software development. While microprocessor speed continues to double every 18 months, the creation of software remains labor intensive. Success is heavily dependent upon the skill and

experience of the individuals involved in production. Improvement is linear, if at all. Errors are common. The costs for labor are high, with starting salaries over $50,000 common for recent four-year computer science graduates [Ref 13]. Complicating this equation is the continuing decline in the number of skilled professionals capable of producing high quality software. Over the previous decade, the number of degrees awarded for computer science in the U.S. has dropped an astounding 42% [Ref. 13].

2. The Rapid Prototyping Solution

With demand for faster, better, bigger, and more complex software accelerated by the explosion of information technology and automated systems, the cost of increasingly scarce software expertise has risen in tandem. It has become incumbent upon software manufacture's to maximize the productivity of software designers, engineers, and programmers in order to remain competitive and meet demand. This is where the utility of integrated rapid-prototyping environments such as CAPS becomes important. By enhancing the effectiveness in identifying and specifying requirements, CAPS decreases errors, rework, and reduces the overall time to complete a software project. However, for CAPS to be seriously considered as a viable alternative solution it must be easily understood and used. It is the intent of this thesis to make CAPS a more attractive alternative to other software development systems by creating an intuitive, user friendly interface.

B. SOFTWARE DEVELOPMENT SYSTEMS

1. The Waterfall Model

Problems exist with the predominant software development methodology, the Waterfall Model (Figure 1.). The Waterfall Model is a linear plan for the production of

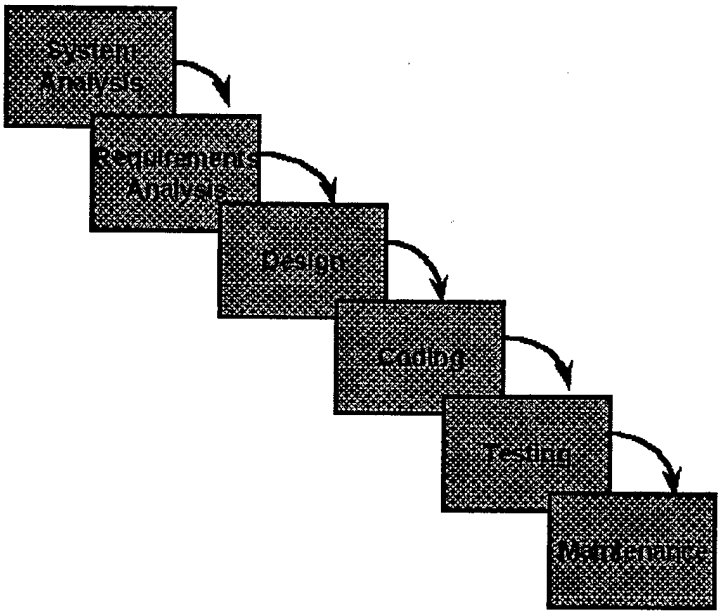


Figure 1. Waterfall Methodology for the software life cycle.

software. The user and the designer analyze the users needs and produce a set of requirements that are used to create the design for the software system. The program is implemented, tested, and then returned to the user for validation.

Unfortunately, most errors occurring in software are requirements based. This stems from the imprecise communication that often occurs between the end user, who understands the problem domain but not software design, and the software designer, who understands software development but usually possesses little knowledge of the problem domain. The user, who is the ultimate arbiter of what is correct in the software, is unable to validate a system effectively until a working version is produced. At this point, a large portion of the development time and budget has been expended, leaving little time or

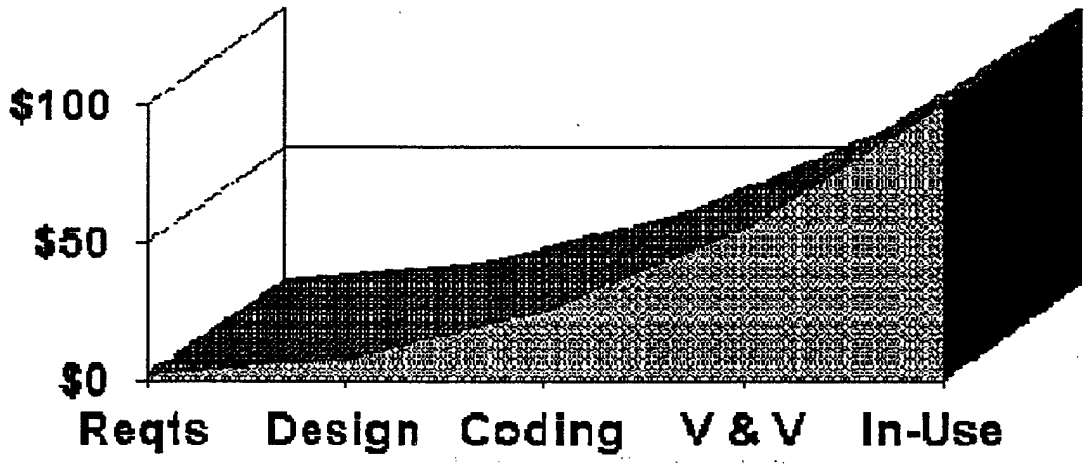


Figure 2. Relative cost for error correction over the software life cycle. [Ref. 4]

money for corrective action. Furthermore, an invalid requirement at this stage has affected large portions of the implementation code, resulting in a 100 fold increase in the effort necessary to correct the mistake over detecting the same error before implementation had taken place (Figure 2).

2. Rapid Prototyping

One workable answer to the problem is rapid prototyping. As illustrated by Figure 3, rapid prototyping does not rely on validation of a finished product. As in the Waterfall Model, the users initial goals are analyzed and a set of requirements created. These are used to create a partially functional prototype that can be demonstrated to the user. The user then assists in identifying faults in the design of the system very early on,

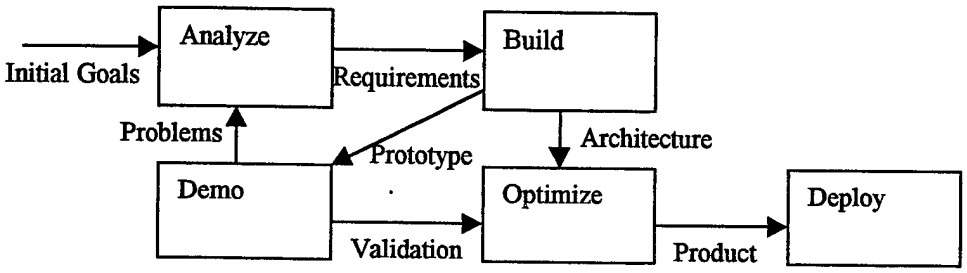


Figure 3. Rapid Prototyping Software Life Cycle.

before valuable time and money are poured into propagating those faults through the entire system.

The payoff for using rapid prototyping is evident. Requirements are often changed repeatedly by users who are initially unsure of the functionality required, leading to wasted effort and frustration. Rapid prototyping obtains user feedback and solidifies requirements early, providing a clear path for future progress and reducing friction between user and designer.

In traditional software development, specifications evolved from the user requirements are presented in written form or a formal specification language. Neither of these methods are suitable to users with little or no knowledge of software design.

Misunderstandings result. Rapid prototyping allows the user to perform validation on executable specifications. For example, if a user requires the system to initiate the firing sequence of a rocket motor when a certain air speed is reached, a prototype of the system, complete with simulations of the missile hardware, can be completed and demonstrated to the user long before a physical prototype could be delivered and at a fraction of the cost. The user could then evaluate the prototype's performance based on whether the rocket's motor was signaled to ignite at the appropriate air speed without actually having to launch the missile. This interaction between the user and actual working prototype results in a clearer understanding by both the user and designer of what the user truly desires.

The risk of failure in software development is considerable. Colonel Chadwick, the commander of the Marine Corps Tactical Systems Support Activity stated that software projects with a budget in excess of $100 million had a 100% failure rate [Ref. 18]. Unfortunately, this is not an isolated phenomenon. In 1995, DOD wide, only 16% of software programs were completed on time and on budget. Furthermore, an astounding 31% were cancelled entirely (Figure. 4). Rapid prototyping can help resolve this problem. Large and complex systems can be modeled quickly, concentrating on the design of the software architecture while leaving the details for later implementation. This allows the user and designer to perform risk assessments and feasibility studies using the prototype. This is preferable to waiting for delivery of the beta version of the total system, when time and budget are nearly exhausted and changes are much more expensive.

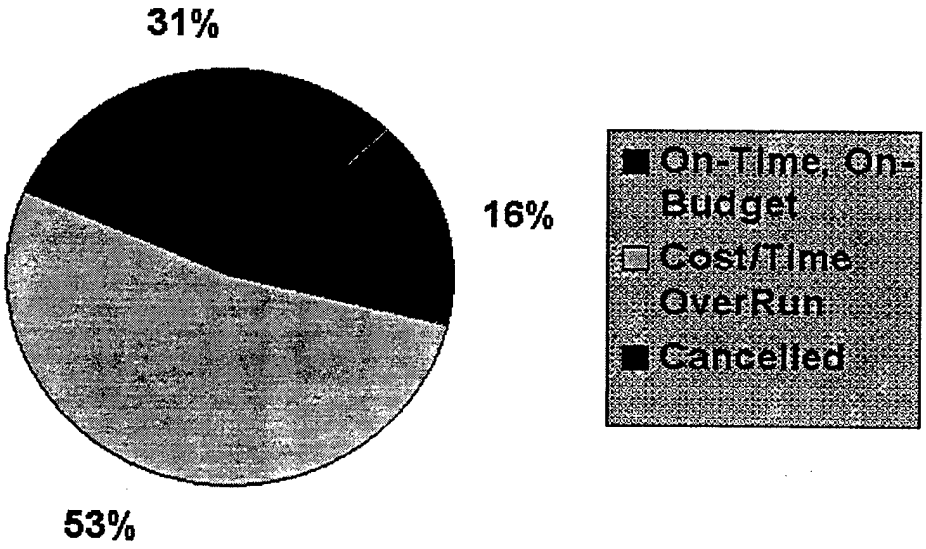


Figure 4. DOD software delivery statistics for 1995 [Ref. 4].

While the advantages of prototyping are many, the prototyping process must be fast in order to be effective. The designer has limited time in which to create the final product and the sooner the prototype receives final validation from the user, the sooner it can be forwarded to the programming team for optimized implementation. This implies automation. Automation can generate code faster and more accurately than any human programmer as well as provide organization to help with decision support. Database software can automate searching through a system to find code that meets specific design requirements. Allowing software to perform tedious and time consuming tasks can be used to speed up nearly every aspect of the project.

3. The Computer Aided Prototyping System (CAPS)

CAPS supplies the automation that rapid prototyping requires. Hosting an array of tools to assist in producing a prototype in minimal time, CAPS is a completely integrated software design environment. The major areas of automation support provided by CAPS are designer interface, software databases, execution support, and an evolution control system. Combined, these tools allow the designer to perform all the necessary functions required by the validation cycle of the rapid prototyping model (Figure 5).

The designer interface contains the syntax directed editors and consists of:

- Editing tools for the Prototype System Design Language (PSDL). Consisting of the Graphical Editor and the PSDL Editor, this is the portion of the CAPS system concerned with creating and modifying prototype

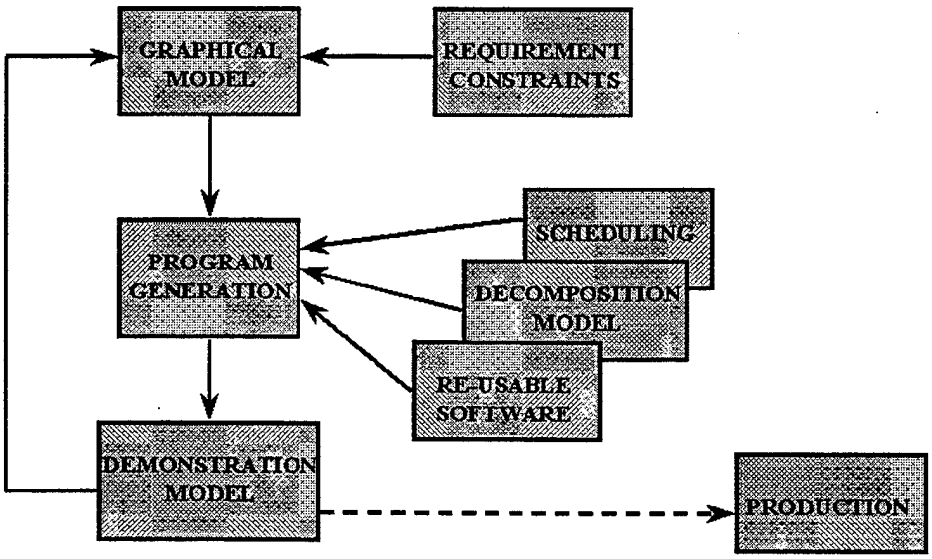


Figure 5. CAPS prototype creation flow diagram.

designs. It is also the major subject of this thesis.

- An Ada programming language editor. CAPS currently uses the Verdix Ada83 editor for creating and modifying Ada modules for use in the CAPS system.

- A requirements editor. The requirements editor allows for creating, updating, and tracing the accomplishment of user requirements over the life of the prototype.

- An interface editor. Currently the interface editor is The Transportable Applications Environment (TAE) Project , a state-of-the-art user interface development and management system. The interface editor allows the designer to rapidly create a graphic user interface from which to display and input data into the prototype.

These tools are included to assist the designer in rapidly entering information pertinent to the creation of the prototype. Speed is important. For the selection of editors, a premium was placed on propagation of appropriate constraints, preventing syntax errors, and providing for robust error diagnostics. Propagation of constraints is a necessity to ensure that the final product performs within limits initially set by the user. Prevention of errors and robust diagnostics serves a dual purpose. In the early stages of prototype design, the detection and elimination of errors is much less costly than correcting the same error later in the development cycle. The reduction in the number of

errors in the design also serves to reduce the time required to create a prototype, allowing a working prototype to be demonstrated to the user quicker.

The Database consists of two sections:

- The Design database. The design database provides for the storage of prototype development data. This allows a designer to search the database for previous prototypes that may have components that fit functionality in the present design. Allowing reuse of prototype components not only speeds up the process, but reduces errors as completed prototypes have been tested repeatedly during the rapid prototyping cycle.
- The Software database. Just as the design database allows reuse of existing prototypes, the software database lets the designer search for reusable Ada and PSDL components. Additionally, since PSDL specifications are formalized, they can be used as the index for a database query that returns a component that is a very accurate match to what the designer needs.

Execution support provides compilers and other tools necessary to convert the PSDL code and graphs into a working real-time prototype. Execution support allows the designer to make an executable prototype a reality in four easy steps.

These are:

- The Translator - translates PSDL language to Ada code.
- The Scheduler - creates schedules for execution of a real-time prototype with corresponding Ada code.

- The Compiler - compiles Ada code into executable form, presently Sun Ada version 1.1.
- The Execution Shell- opens a shell in which to run a prototype.

4. Prototype System Design Language (PSDL).

PSDL is the design language that supports CAPS. Standard programming languages are optimized to support efficient execution of a program. They consist of complex algorithms and data structures for creating small, fast executables. They are quite detailed and require a well-trained programmer in order to be used efficiently. Design languages are quite different. Execution is not the prime motivation. The objective is the creation of an efficient and easily understood design. Because of this, design languages are more expressive and simpler to use. Furthermore, where programming languages provide for comments as an adjunct to the main functionality, design languages have the recording of goals and justifications as a prime objective.

PSDL achieves these goals. Based on a simple grammar and a graph, PSDL provides for easy understanding of the language while yielding a powerful ability to create prototypes.

The most basic building blocks of PSDL are operators and streams. They represent the two things that can be done with data: manipulation or relocation. Operators are functions (or state machines) and streams are data flows.

Operators have three manifestations. Ordinary operators, represented by circles in

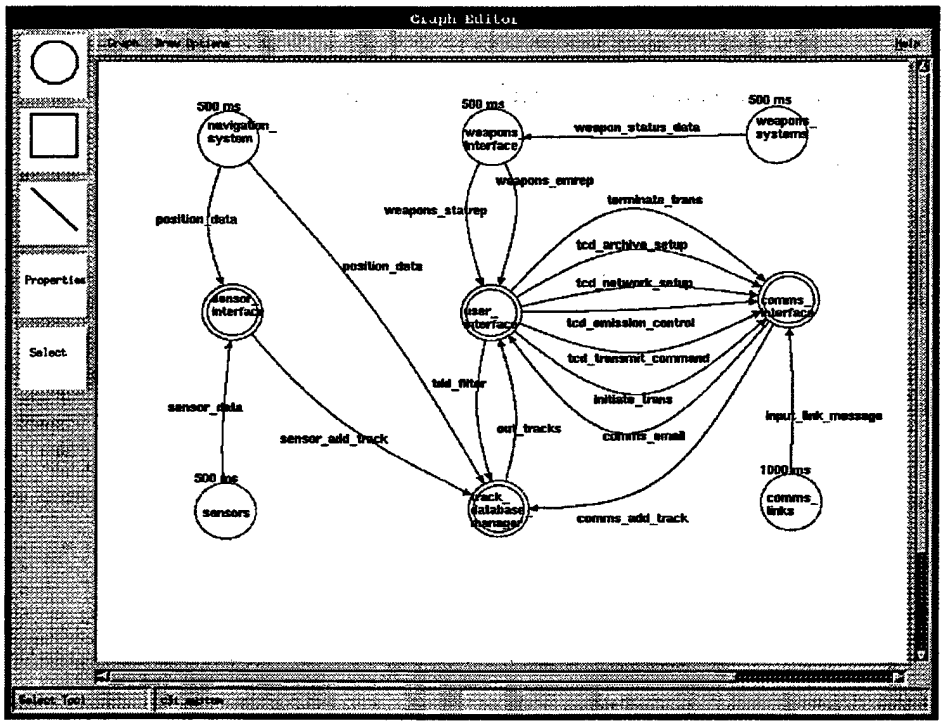


Figure 6. Graphical Interface for CAPS.

the example of the graphical editor shown (Figure 6) perform operations on the data from the streams. Composite operators, designated by a circle with a double ring, represent a sub-graph. Sub-graphs are standard PSDL graphs, and are represented by composite operators for the sake of clarity. A graph would rapidly become cluttered and confusing without the ability to compress functional areas into composite operators. Decomposition of functionality is easily achieved using this construct, as composite operators can contain other composite operators, allowing decomposition to continue until only simple atomic functions remain. Terminators represent sources of input external to the design. Terminators are simulations of external systems, are not required for the prototype, and

will not be in the delivered software. The third kind of operator is the type operators used to represent the functionality of abstract data types defined by the user.

A stream is a communications link connecting two or more operators and is represented in the GE by a line connecting operators. Operators are producers and consumers. A stream requires at least one operator, producer or consumer. There are two types of streams available in PSDL, dataflow streams and sampled streams. The dataflow stream works on the first in, first out principle. Data values are never lost or replicated. Sampled streams model continuous data input. Only the most recent data value is used by the consumer with all others being lost.

Triggers are control constructs that act as decision points on the firing of operators as a result of receiving data through a stream. Triggers can be set to fire in the event that some or all of a set of data is present. Execution guards are the constructs within the operator in which triggers operate. If no execution guard is present within an operator, the default causes the operator to always execute. Execution guards and triggers can be thought of as implementing an if-then-else programming control structure.

Operators can also be fired through the use of a timer. Timers are software stopwatches that are declared within a composite operator and hold a time expression that, while running, is continuously updated to record the passage of real time. Timers have three control constraints that affect the value held by the timer. They are START, STOP, and RESET. Start obviously causes the timer to run. Stop freezes the current value. Reset returns the current value to zero.

Like triggers, timers, and execution guards, other output guards exist to assist the user in controlling the flow of a program. Output guards allow conditional transmission of data to an output stream. Output guards are assigned by stream and are normally used to filter output to the remainder of the program. This allows a multiply threaded stream to pass data selectively rather than broadcasting to all.

PSDL is a real time prototyping language. As such, timing constraints are a necessary part of the language. The most important of PSDL's timing constraints is the maximum execution time (MET). The MET is the maximum amount of time that the operator is allowed to complete its activities. The addition of a MET constraint to an operator defines that operator as time critical and thus subject to timing constraints. Two types of time critical operators, periodic and sporadic, are used within PSDL.

Timers trigger periodic operators. A value respresenting the period of the operator is set to a time value. When the timer reaches that value, the operator is allowed to fire. It is not necessarily required to fire immediately. A second value, finished within (FW), is provided to allow flexibility in scheduling. After the operator is triggered by the period, the program can delay execution of the operator as long as the operator is completely executed prior to the arrival of the FW time.

Sporadic operators are triggered by the arrival of data. These operators are executed whenever data arrives subject to the minimum calling period (MCP) of the operator. The MCP is a time value. If a sporadic operator executes, it cannot be executed again until the time indicated by the MCP has elapsed, regardless of the receipt of data. This allows the designer to sample data at intervals, and prevent responses to

every byte of data sent to the operator. Sporadic operators also require a maximum response time (MRT). The MRT is the maximum amount of time that can elapse between the arrival of data and the completion of the operation. This constraint provides the designer with a mechanism to ensure that critical work is completed within a known amount of time.

Operators also contain data elements that assist human users in understanding the purpose of that component. The description element allows the designer to place a natural language description of what the operator does within the structure of the PSDL code. This acts much like a comment would in a programming language. Keywords are also employed by PSDL. The keyword component allows the designer to place natural language identifiers within the prototype in order to assist future designers in evaluating the component for reuse. Keywords are used by the database as indices for queries.

Composite operators contain one other construct of interest, the graph description. PSDL provides for a component that describes the design graph of a decomposed composite operator to the graphical editor. The graphical editor translates this component in order to display the design within the editor window. A complex structure, the graph description contains coordinates, colors, fonts, and all visual components of child operators and streams.

C. THE DESIGNER INTERFACE

The graphical editor and the syntax directed editor are two representations of the same PSDL code. The graphical editor displays a truncated version of the code based on

the objects comprising a PSDL graphic. The syntax directed editor displays the hidden information that gives the prototype depth. When CAPS was originally designed, the use of graphical interfaces was a rarity and they were often poorly designed. Designers also prefered to directly access the PSDL code if they wished. These decisions are no longer valid. Graphical tools are much better and abstraction in software design is realized to be essential for truly understanding a system. There is no inherent reason for this separation of PSDL code into two separate interfaces.

1. The Graphical Editor (GE).

The graphical editor is the portion of the designer interface with which the designer creates the graphical representation of the prototype (Figure 6). The GE is simple in design and execution, allowing novice operators to begin creating real-time prototypes using the basic stream and operator objects with little formal training. With the use of the Quickstart manual for CAPS, a novice can create a simple prototype within minutes using only the graphical interface. Although a powerful concept and tool, the CAPS graphical interface is unable to deliver the necessary functionality required by more complex prototypes. For this, CAPS provides a syntax directed PSDL editor that works in conjunction with the graphical interface to produce a working design.

2. Syntax Directed Editor.

The CAPS syntax directed editor (SDE) displays the text version of the PSDL prototype – the part that the translator converts to programming language code (Figure 7). The SDE allows the designer to perform detailed, advanced PSDL programming. It is the principal interface for the designer, allowing the creation, editing and viewing of CAPS designs. PSDL language code generated by creating a design is modified within the SDE. The SDE also provides access to a wide variety of tools, such as the ability to

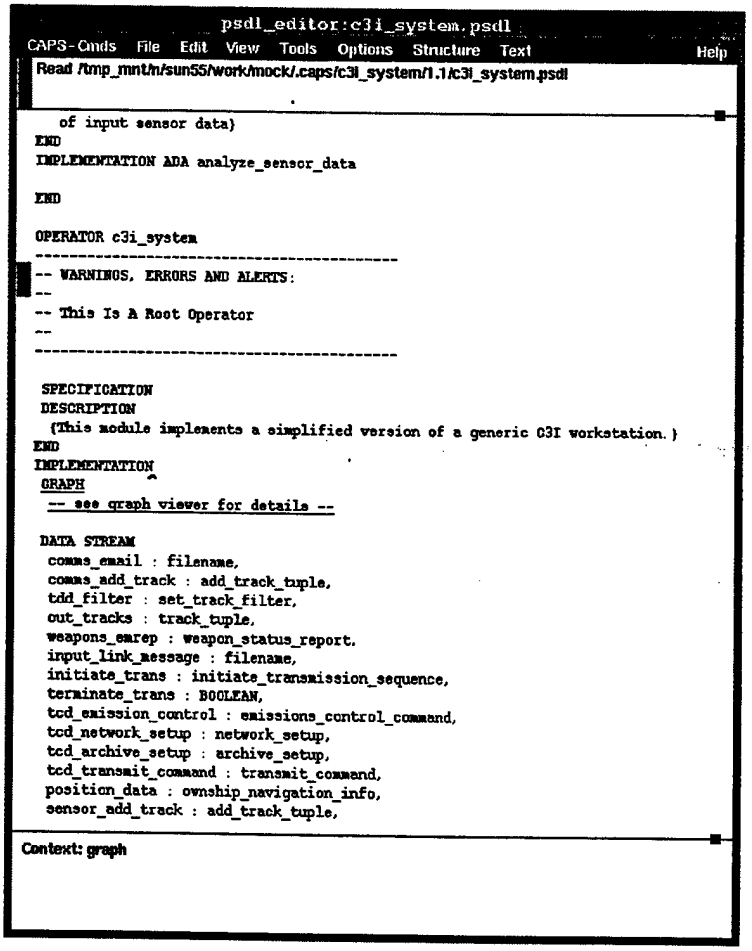


Figure 7. PSDL Editor.

query the software database, that assist the designer in creating a prototype.

The separation of these two editors is also the major deficiency in the design of CAPS release 1.0. As early as 1990 users of CAPS were underscoring the need for a syntax directed editor combined with the graphical editor for generating PSDL code [Ref. 2]. The necessity of writing PSDL language code in the SDE brings with it the entire training overhead associated with programming languages. Furthermore, since the modification of the design in the GE almost always requires a corresponding change in the SDE, the division of the two editors into separate windows has no practical purpose to justify the cumbersome environment. Merging these two windows and their underlying functionality is the major objective of this project.

3. PSDL parsing.

A syntax directed editor assists the designer in creating a prototype by automatically checking the PSDL language code for appropriate structure. For the purposes of this project, PSDL has seven major constructs that may be input into the graphical editor. Each of these constructs must be checked for syntax errors. Needless to say, the descriptions of the seven parsing elements are truncated. The full specification of the meanings of each can be found in Appendix D.

The seven constructs are:

- Expressions: Expressions represent a wide range of tokens within PSDL. An expression can be a string, an integer literal, Boolean expression, a type

operator identifier, time value, data stream, timer or a combination of the above. Example: X > 2.

- Initial expressions: Initial expressions are exactly the same as an expression except references to time values, timers, and data streams are excluded. Example: Y = 3.
- Output guards: A set of logical statements initiated by the OUTPUT token, output guard statements consist of a list of identifiers followed by IF, then an expression and ending with a requirements trace. For example:

  OUTPUT temperature IF x<2 req1

  In this example temperature is a data stream. When the stream is triggered, if x is less than two, then requirement #1 is fulfilled.
- Timer operations: Timer operations start with a timer_op token. This consists of one of three values - Start, Stop, and Reset. The timer_op token is followed by the identifier (name) of the timer performing the operation. This is then followed by the IF token, an expression, and a requirements trace. This works in the same manner as the output guard, with the exception that the expression is evaluated when the timer performs the action represented by the timer op token. Example:

  START TIMER

  timer1
- Operator specifications: A sequence of attributes and requirements traces comprise the body of this structure, starting with the OPERATOR token and

an identifier, followed by the SPECIFICATION token and concluding with an END token. An attribute is simply a characteristic of an operator, such as an input, output, state or an exception. Example:

```
OPERATOR Producer_1
SPECIFICATION
    GENERIC
      G1 : FLOAT
    OUTPUT
      DA : Missing__Info
    MAXIMUM EXECUTION TIME 0 MS
END
```

- Type specifications: This element begins with TYPE token and an identifier followed by a SPECIFICATION token and ends with an END token, in the same manner as the operator specification except the interior is more complex. This component describes the attributes of a user-defined type. Generic declarations are parsed first, followed by a set of operator specifications as described above. A final list of functions available to the type completes the structure. Example:

```
TYPE STACK
SPECIFICATION
    GENERIC
      types : private

    type_2 : public

    OPERATOR PUSH
    SPECIFICATION
        INPUT
          I : INTEGER
        INPUT
          S : STACK
        OUTPUT
          S : STACK
    END

    OPERATOR POP
    SPECIFICATION
```

```
        INPUT
          S : STACK
        OUTPUT
          I : INTEGER
        OUTPUT
          S : STACK
    END

    OPERATOR Empty
    SPECIFICATION
        OUTPUT
          dummy : STACK
    END
    KEYWORDS
      stack, adt
    DESCRIPTION
      {This is a generic stack adt}
    AXIOMS
      {push (s,x) = s::x}
END
```

- Exception guards: Exception guards begin with the EXCEPTIONS token followed by an identifier. The IF token is next followed by an expression and a requirements trace. With the exception of substituting the EXCEPTIONS token for the OUTPUT token, the exception guard and output guards are identical.

4. Previous Work.

The grammar necessary for creation of the seven parsers that were created in this thesis was already a part of the original CAPS code created by Professor Luqi . This code was combined in the Aflex and Ayacc files for parsing PSDL code as a whole language. The work necessary for converting these files into useful material for this project consisted of identifying and separating the syntax elements of a particular PSDL component from the overall grammar. This greatly simplified this portion of the project.

Originally, a commercial graphical editor had been selected for use with the new version of CAPS. A powerful tool, it also provided a very intuitive user interface. Unfortunately, licensing restrictions connected to this product would have seriously hampered the free use and distribution of the completed system. As an alternative, a graphical editor created as a class project by NPS graduate students, under the guidance of Professor Mantak Shing , was selected. Although not as robust and extensive as the commercial version, enhancements added by Ken Moeller, a graduate, have made it a viable and cost effective replacement for the commercial graphical editor.

D. SUMMARY

Rapid prototyping is a viable solution that is available now that can alleviate many of the problems presently experienced by the software industry. CAPS is an excellent system with which to create prototypes. An effort to improve the efficiency with which a software prototype designer can use CAPS by simplifying and enhancing the interface is valuable to the Naval Postgraduate School, the DOD, and the software development community as a whole. An improved interface allows prototypes to be developed more accurately and in less time. The reduction in time to create a prototype version creates a corresponding reduction in time to produce a user-validated design. An improved interface reduces errors; again saving time spent correcting implemented code.

The interface is essential in creating an effective CAPS environment. An effective CAPS environment is important in extending the use of rapid prototyping. Extending the use of rapid prototyping will reduce the amount of late, poor, or simply

unusable software being produced. This is a small step on a road that can save the DOD and the nation billions of dollars.

III. ARCHITECTURE OF THE PSDL EDITOR

The Software Development Method is normally divided into five distinct phases: Requirements Analysis, Functional Specification, Architectural Design, Implementation, and Evolution and Repair [Ref. 5]. This chapter will record the first three phases as they occurred during the project.

A. REQUIREMENTS ANALYSIS

Requirements analysis begins with the receipt of an initial problem statement from the customer. The CAPS team possessed an advantage in this respect, being the customer and the designer. The initial problem statement was quickly developed as:

The purpose of the proposed software system is to provide a more efficient and user friendly interface for the CAPS environment.

The goals of requirements analysis are to define the purpose of the proposed system and to determine the constraints on development [Ref. 5]. To achieve this, the initial problem statement is reviewed in the context of the environment. Specific requirements and constraints for the combined graphical editor are then developed.

Three specific requirements for the system were developed:

- Create seven PSDL parsers to be integrated into an existing graphical interface that will allow syntax directed editing of PSDL components from within the graphical editor. The editors must check one of the previously denoted PSDL constructs for accuracy, and return an error location in the event that a syntax error is discovered.

- Retain all the functionality of the present SDE and graphical interface. It is necessary to retain all functionality in order to ensure that prototypes currently developed under CAPS 1.1 are usable in the new version being developed.
- Remove the memory leak present in the current version. The failure to reclaim memory following the destruction of temporary data structures causes the previous version of CAPS to slowly drain resources from the overall system. Over time, if CAPS continues to run, all memory resources are exhausted and the system fails. The new system will rigorously recycle memory freed when data structures are no longer necessary and are destroyed.

Constraints on development must also be addressed during requirements analysis. Constraints are limits which the designer is required to respect in the construction of the proposed system. There are many ways in which constraints can be documented. For the CAPS project, constraints were divided into five categories: Resources, Performance, Environment, Form, and Method.

1. Resources.

Time to complete the project was limited. The Software Engineering Group at NPS participates in the DOD technology transfer program. As such, a number of organizations were dependent upon the next release of CAPS to accomplish their own objectives. Although the original delivery date was set for 17 July 1997, a more realistic assessment rescheduled the delivery of the product on 12 August 1997.

Manpower was also limited. The CAPS team consisted of seven members. Furthermore, only four of the seven were available for full time work on the project. Given that the project resulted in the modification of 65,000 lines of code, having only seven people provided a limited pool of man-hours from which to draw.

Budget considerations were minor in comparision to a comparable software effort within the private sector. Over the course of the project 3024 man hours were expended or about one and a half man-years. At current labor costs for software programmers/designers this would have represented an investment of approximately $100,000. Other budget costs were inconsequential.

2. Performance.

Constraints on performance were limited to insistence that memory storage management be rigidly controlled to correct the memory leak in the original release.

3. Environment.

The targeted system was to use SunOS 4.1.3 operating on a Sparc 10 workstation The system will have Motif X.11 available. No other environmental constraints were imposed.

4. Form.

The system would be implemented in Ada95 to assist in maintainability, and the Graphical Editor (GE) developed and enhanced by NPS graduate students would be incorporated into the design.

5. Methods.

There were no constraints assigned in regard to methods.

B. SPECIFICATION

The goal of the functional specification stage is to define precisely the external interfaces of the proposed system [Ref. 5]. The process of integrating the graphical editor (GE) into CAPS and integrating a PSDL syntax directed editor into the GE creates an unusual situation where interfaces that normally would be considered internal are external for our purposes. Three interfaces need to be considered (Figure 8).

The first interface for specification is between the CAPS prototyping menu and the PSDL editor. The action taking place consists of passing the name of the PSDL file to the PSDL editor. Passing of data occurs in only one direction. One argument is

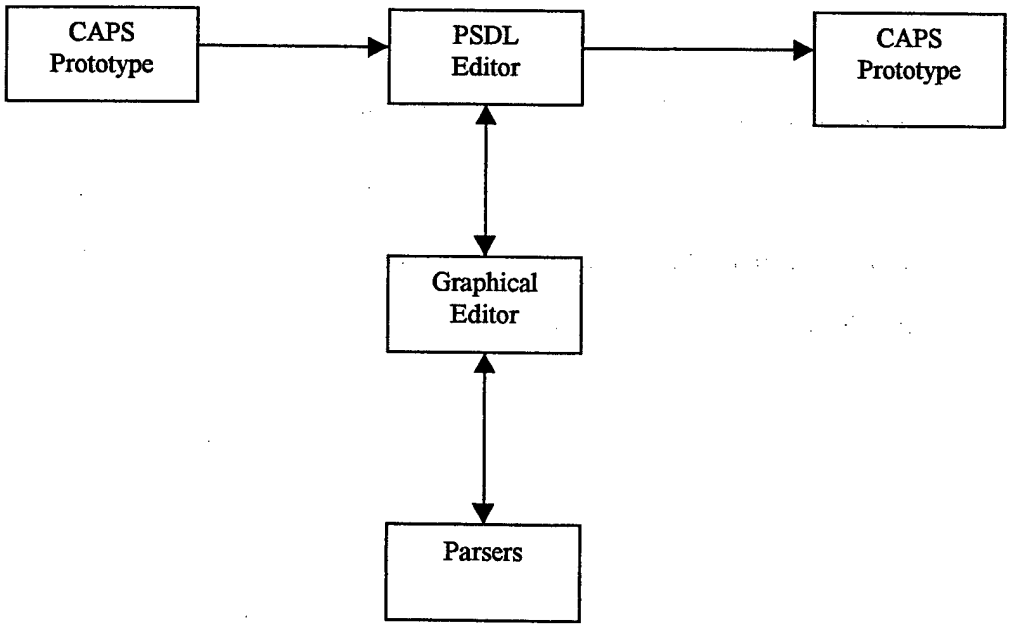


Figure 8. PSDL Editor Architecture.

required. It must be a string with a length of at least six characters, the last five of which match the string ".psdl".

The second interface is also between the PSDL editor and the CAPS Prototype, except in the opposite direction, and requiring different arguments. This is not a direct call but rather an implicit obligation on the part of the PSDL Editor to supply useable PSDL code for translation. As such, the specification for this interface required that the PSDL Editor save a valid PSDL prototype to the file called when the editor was initialized. To accomplish this interface, a filename must be supplied that is exactly the same six character string as in the first interface. It must also have a valid PSDL *prototype*, an abstract data type defined in the editor , to save to the file.

The third interface is between the PSDL Editor and the GE. This interface passes data in both directions, with the editor feeding the GE the graph description of the prototype and the GE returning that description along with the users choice for the next action to be performed and any error messages that may be necessary. The transfer of information in this interface is complicated by the difference in languages between the two modules. Fortunately, data types exist within the PSDL Editor that fit the needs of the interface. The graph can be passed using an in/out parameter *Graph_Desc_Node*, the action will use an out *Action* type, and the error messages will also be an out parameter of the type *error_msg*. The *Action* and *error_msg* types are simply enumerations and are readily converted to structures of the C programming language. The *Graph_Desc_Node* is a specially constructed data type consisting of C compatible structures designed for passing graph data.

The final interface joins the GE and the parsers that make up the SDE. Again, a cross language exchange of data is necessary and again, data flows in both directions. The data that needs to be exchanged consists of a character string that requires parsing for correctness, a Boolean variable designating whether the string passes the parsing test, and integers designating the line, column, and length of the first error discovered. As the C to Ada library of functions and available data types is not as extensive as the reverse, the interface used C pointers., which can be readily converted in Ada95, providing access to each of the five data elements being passed.

C. DESIGN

Design encompasses the decomposition of the system into software modules.

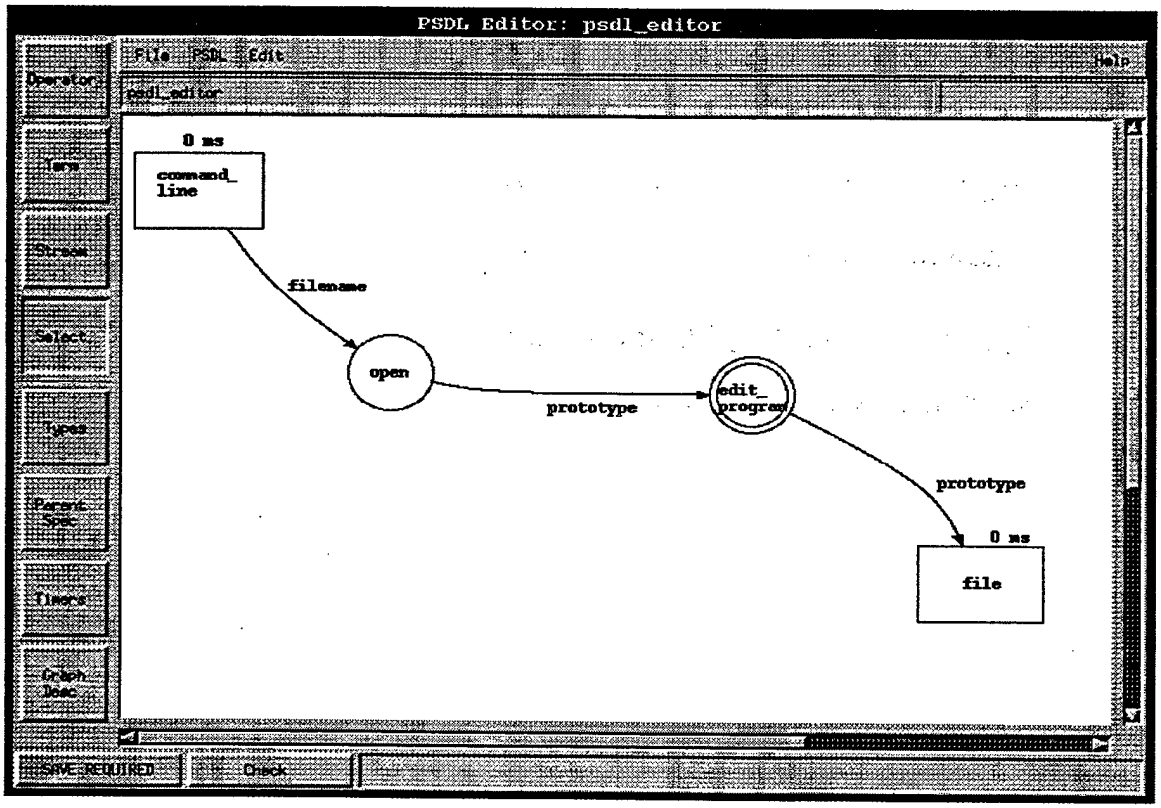


Figure 9. Functional Decomposition of the PSDL Editor

Many of the modules were already in place and only required modification, which was extensive in some cases. However, a description of the improved PSDL editor is in order. As an aid to understanding, Figures 9 - 13 show the PSDL editor being used to model and prototype the PSDL editor.

At the top most level, the PSDL editor is expected to perform two functions (Figure 9). Upon initiation the program receives a PSDL filename as an argument from the command line of the operating system. The *Open* function takes that filename and searches the default prototype directory in search of the named file. If it exists, it is opened and the prototype name contained therein is passed to the next function,

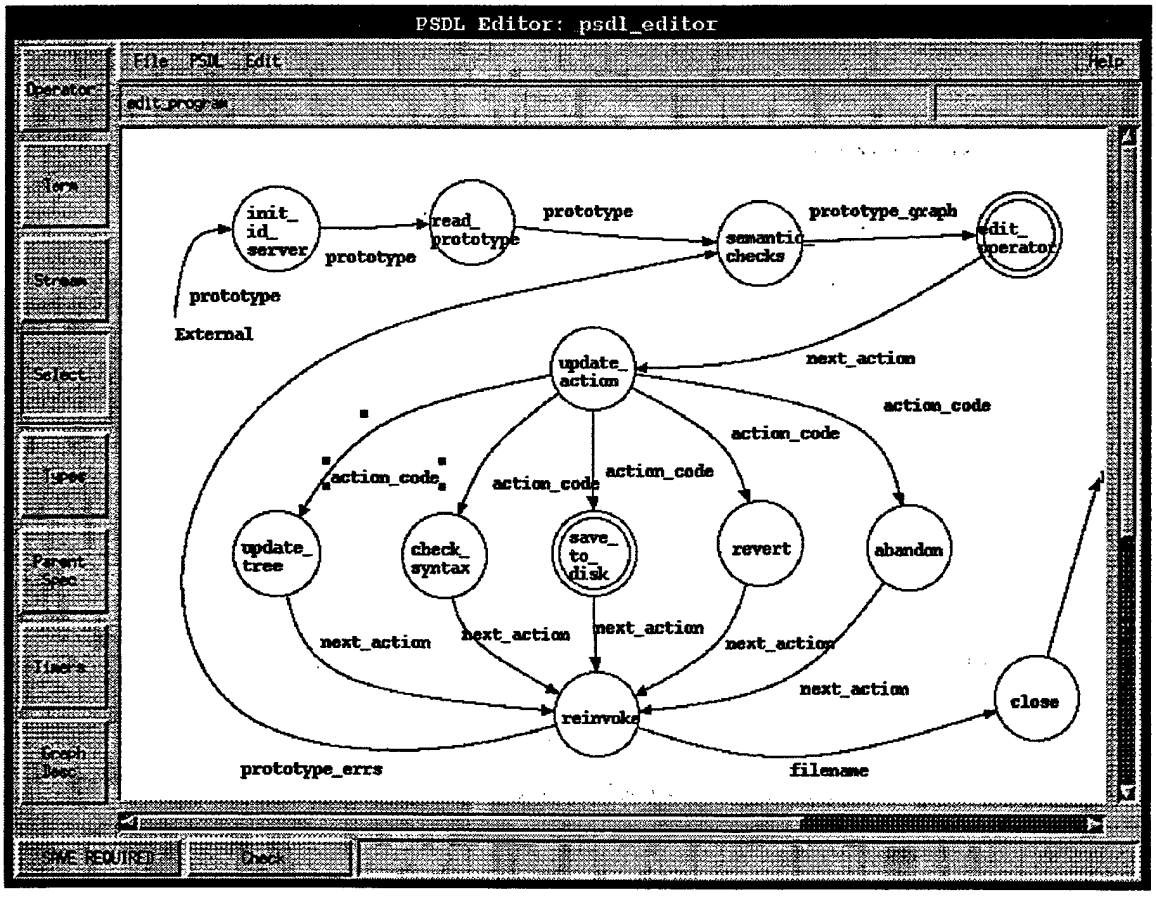


Figure 10. Functional Decomposition for Edit_Program

edit_program. If the file is not found, a new file is created using the given filename and an empty prototype is forwarded.

Edit_program receives the prototype name, manipulates its contents and then returns the modified program to file. Edit_program, because of complexity of the tasks assigned, must be decomposed further (Figure 10).

When initialized *Edit_program* begins by assigning the prototype an unique identification number. This allows the system to recognize the prototype uniquely even

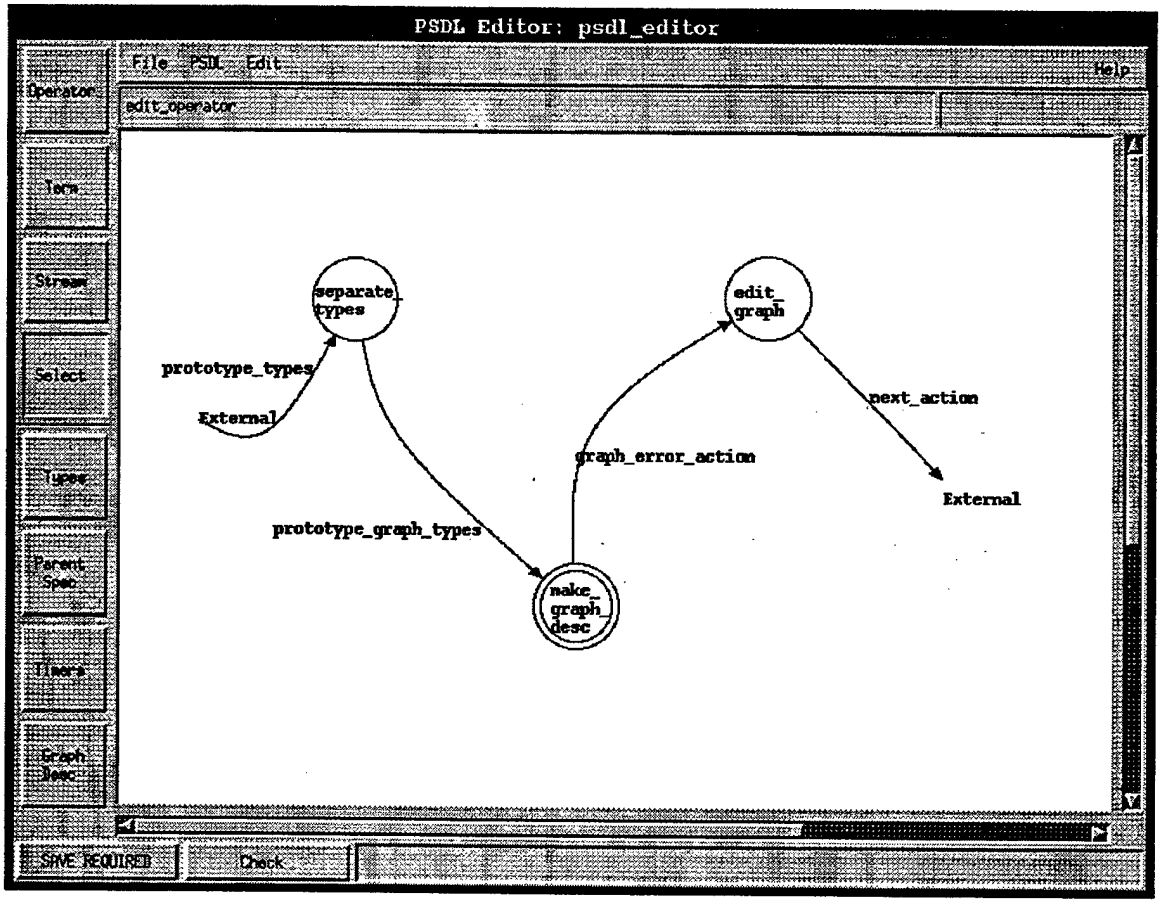


Figure 11. Functional Decomposition of Edit_Operator

if the user eventually creates other components using the same name. After assigning the identity, *read_prototype* reads the entire PSDL program from the file into a skeleton prototype data structure. The prototype is then subjected to semantic checks to ensure that the program is in a useable format and not corrupted in any way. The verified prototype is then ready to be edited by the GE. This is performed within the substructure of *edit_operator*, which will be detailed below. Upon completion, *edit_operator* will have received the modified prototype and an action code indicating the user preference for further use of the system. *Update_action* tests the action returned and performs the necessary action, such as *save_to_file* or *revert*. *Reinvoke* is then initiated, testing whether further action is required. If true, the program loops back to *semantic_check* and performs the cycle again. If false, the *Close* function closes the opened file and returns control to the operating system.

Edit_operator performs only three functions, but they are vital to the interface with the GE (Figure 11). First, user defined types and operators are separated within the operator list. This is necessary because types and operators display difference characteristics and must be manipulated in different manners in order to present the GE with a useable graph. The separation results in two lists of operators and types. These lists and a blank graph data stucture are passed to *make_graph_desc*. Make_graph_desc takes the PSDL data presented and manipulated it to form a graph description. The graph description is a data structure that represents all the information contained in the prototype, converted to use C language conventions. This allows the edit_graph inter-language function call to import the data for use within the GE. Upon completion of user

action, the GE returns the graph description to *edit_operator* along with the user's next requested action.

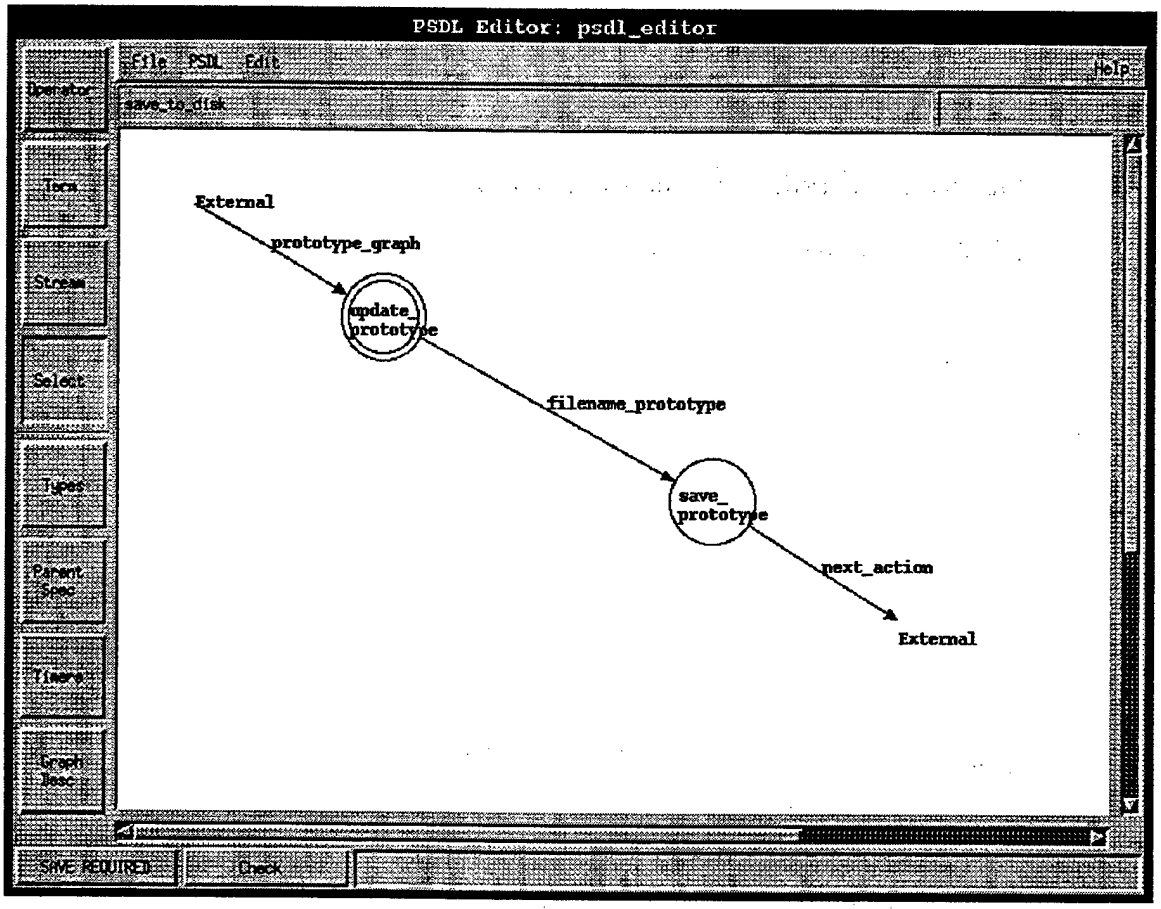


Figure 12. Functional Decomposition of Save_To_Disk

Save_to_disk, the most typical of users actions, also needs to be reviewed. It consists of two functions, *update_prototype* and *save_prototype* (Figure 12). Both functions do what their names suggest, updating the PSDL prototype (as opposed to the C oriented graph description) and saving the prototype to the file. Of the two, *update_prototype* deserves closer inspection.

Update_prototype is the function that unwraps the changes performed within the GE and converts them into PSDL code that is useable by the remainder of CAPS. It

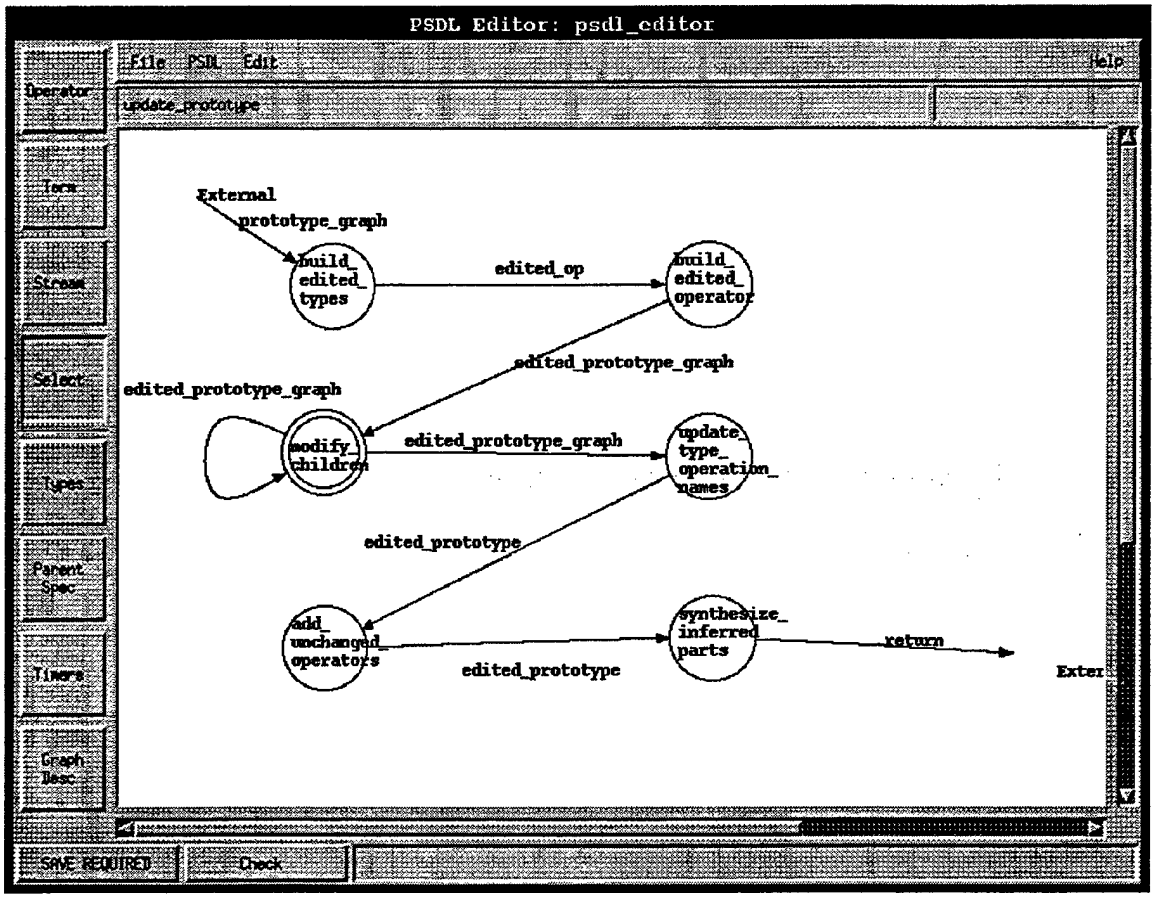


Figure 13. Functional Decomposition of Update_Prototype

consists of six subfunctions (Figure 13). The first function is *build_edited_types*. The second is *build_edited_operators*. These two functions review the graph description returned by *edit_graph* and determine which types and operators have been changed. The PSDL structure is then modified to reflect the changes. Upon completion of these updates, the children of modified composite operators must be updated, as well as any grandchildren, etc. *Modify_children* performs this task using a recursive structure that checks for changes rippling through to each operator from the root down. After the children are checked for changes, the type operations are updated. At this stage, all

modifications performed within the GE will have been implemented in PSDL structure. Unfortunately, the GE will not provide data on operators that were unchanged. This requires that a function be created to check the original prototype against the one being built and add any unmodified operators back into the prototype. The final action in the reconversion of the graph description is *synthesize_inferred_parts*. This function evaluates changes made by the GE that would effect other portions of the PSDL code indirectly, such as the setting of terminator flags, type definitions, and exception declarations.

A final note on the design of the seven PSDL parsers is in order. The parsers were designed to accept a string for an argument, and return an error message and error location. All are C structures. This allows each individual parser to be called directely from the appropriate place in the graphical editor.

D. SUMMARY

This chapter detailed the first three of the five phases (Requirements Analysis, Functional Specification, Architectural Design, Implementation, and Evolution and Repair) of the software development methodology for the enhanced PSDL Editor. The next chapter deals with its implementation, and the fifth phase, Evolution and Repair, will be discussed in Chapter VI.

IV. IMPLEMENTATION

Implementation is the fourth phase of the software development cycle. In this chapter we will cover the aspects of implemention as they applied to the enhanced PSDL Editor project. The tools used, the approach to attacking the problem, and the process by which the project was completed will be covered. Repair and Evolution (R&E) is the final phase of the software development cycle but is properly outside the scope of this paper. Testing and evaluation are normally considered as part of this phase and were a very important part of this project and will be covered in the place of R&E.

A. TOOLS AND ENVIRONMENT.

Tools can make or break a software development effort. The CAPS group attempted to assemble a comprehensive suite of tools that would meet our requirements at a minimal expense.

1. Introduction.

When a project expands to larger than a few hundred lines of code; extensive hand coding cannot be done. Hand coding introduces errors and becomes an unacceptable risk to the robustness of the final product. The proper line of attack is to use pre-existing tools in order to automate large portions of the implementation process.

2. Environment.

The environment for the project was standard for work in the software engineering lab at the Naval Postgraduate School. The project code was created on a Sun Micro Systems Sparc 10 workstation networked within the software engineering laboratory subnet, which in turn is linked to the computer science department network. Keeping the project confined to a subnet allowed work to continue at all times, even when the larger department net was offline for backups.

The operating system installed on the test system was SunOS version 4.1.3. Chosen for proven reliability and known compatibility with the present CAPS system, version 4.1.3 is the dominant operating system within the NPS computer science department, ensuring widespread compatibility of newer CAPS versions with the department at large. One account was used for the entire project with each member of the team having access.

3. Languages.

Programming languages for the project were chosen in deference to the constraints applied to the project by previous work.

Ada95 was selected as the main language for working within the main SDE modules. The original version of CAPS had been implemented using Ada83 in accordance with the DOD Ada mandate being enforced at the time. Ada95 is a logical choice as a successor for that reason alone. However, Ada95 has advantages that would have resulted in its selection regardless. Like CAPS, Ada95 is designed for software

engineering. Ada95 includes all of Ada83's discipline enhancing aspects, such as strong typing and information hiding, which made Ada83 the language of choice for large and complex projects [Ref 9]. In addition, Ada95 is now object oriented, adding flexibility to robustness. Using Ada reduces the cost of initial development of software and significantly reduces the risks involved in program development. Maintenance costs are reduced. Ada is especially useful for projects that involve a large number of personnel. Although this project used only seven team members, the nature of work at NPS has shown that working groups expand and contract. Ada's readability supports easy understanding of code without recourse to extensive commenting. This is an essential feature for long term projects, and CAPS, now operational for nine years, is a long-term project.

GNAT (GNU/NYU Ada Translator) Ada95 version 3.09 became the main compiler. A perfectly serviceable compiler, GNAT was the first compiler to implement the Ada95 design [Ref. 7] . It is certified. It was also documented as supporting the Unbounded_Strings abstract datatype, which contained the key functionality for enhancing storage management, a key requirement. Finally, cost is always a consideration, and GNAT provided the features needed by the project at no cost.

The choice to use the C programming language was also determined by the requirements of the project. The existing graphical editor to be integrated into the CAPS program was created using C. Although regrettable from the standardization and maintenance points of view, the original creators of the graphical editor did have good and sufficient reason for choosing C in that the X.11 Motif function calls necessary for

building a windowing environment were all in that language. Fortunately, experience in the C language is widespread and a number of team members were quite proficient in its use.

The C compiler chosen for the project was "gcc", the gnu C compiler, which is included as a standard part of SunOS. The reasons for its acceptance were the familiarity of the compiler to the team members and, as with the GNAT Ada95 compiler, it was available at no cost.

"Awk", though called a programming language by it's creators (and from whose initials it gets its name), is really a pattern matching language [Ref. 6]. Awk was included among the languages needed because early in the design phase, it became apparent that considerable amounts of hand reworking of code segments could be avoided if certain recognizable patterns could be replaced automatically. Awk performs tasks like this using very few lines of code, making it an excellent addition to the arsenal of tools available.

Make is not in itself a programming language, but is a tool that uses Unix shell commands to create powerful scripts for building executable code. Make was chosen because of this ability and the lack of any viable alternative.

4. Editor

The main text editor employed by the project team was vi, the standard visual line editor provided with the operating system at no cost. If employed, another text editor would have required installation on the project system, eating into project time and

administrative resources. Knowledge of vi's command set was widespread within the team, even if the knowledge was shallow in some cases. In addition, vi's command set is easily combined with Unix shell commands to create extremely powerful searching and replacing routines. This capability in itself was enough to guarantee inclusion in the project tool set.

5. Miscellaneous Tools

"Sed", the Unix stream editor, acts in much the same way as Awk, but possesses a simpler grammar. It was chosen in order to perform quick substitutions of text. As part of the Unix tool set, it also possesses the advantages of availability and cost.

"Gen" is a code manipulation tool for creating loops within the Ada structure that would normally be considered illegal by the Ada Reference Manual. For example, consider the following code segment:

```
For child_vertex: op_id in op_id_set_pkg.scan(vertices(g)) loop
```

Normally this would not be allowed because there is no way for the compiler to ascertain the number of child_vertex that will be discovered using the scan function. Gen solves this problem by unrolling the loop, generating new Ada code in the process. The advantage of using this tool over hand coding the affected areas was significant

"Gnatchop" is a text search tool for by the the GNAT compiler that scans a file for Ada package bodies and specifications. Gnatchop then divides the file into corresponding Ada files. Names are assigned according to the convention that the file is named the same as the package contained with ".adb" appended for the package body and

".ads" for the specification. Gnatchop was recognized as necessary to automate processes. Gen, Aflex, and Ayacc were all developed for Ada83 and produce files that require splitting before compilation. Gnatchop not only helped us to produce Ada95 packages, but allowed the use of sophisticated makefile scripts to automate the process.

6. Version Control Tool

The Revision Control System (RCS) was chosen as the archiving system for the project. RCS tracks changes in files from one version to the next and saves only the changes made. This delivers a substantial reduction in the amount of magnetic storage necessary to backup a multiple version library for a large project. In addition, RCS allows adding comments and descriptions to archived versions for easy recognition in the event retrieval is necessary. Beyond the standard need for a backup system for the code being produced, the use of RCS by the implementation team was necessary for project control. SunOS 4.1.3, while an excellent operating system, has no mechanisms to prevent simultaneous editing of a file by two or more programmers.

The implementation team used a process of mirrored public and private directories for holding the latest working version and the current version being modified. The latest working version was to be exported to the public directories following a full archive with RCS. The private directories would hold the code being created and modified. This process possessed two important advantages. Testing could be conducted on a stable version of the system in parallel with the creation of the next version, and in

the event that changes to the code created compiler errors that could not be corrected immediately, a working version of the system was still available for demonstrations.

7. Lexical Analyzers

Aflex and Ayacc are the Ada versions of the popular C tools, lex and yacc. Aflex is a lexical analyzer/generator that accepts a file containing the definitions of lexical element to be recognized and returns a file with a ".a" suffix that contains a lexical scanner recognizing tokens corresponding to those elements. Ayacc accepts a set of tokens, such as those created by Aflex, and a set of rules provided by the designer. Together, the tokens and rules make a grammar. Ayacc generates Ada code for a parser to recognize that grammar. Both tools have been used for years and are well known. For a project such as creating parsers for a syntax directed editor, ignoring the benefits provided by these tools would have cost the team days of hand coding.

8. Debuggers

Two debuggers were employed for this project. Both debuggers were from GNAT and were designed for the C and Ada languages. The reasons for the choice of these debuggers were cost, compatibility, and availability. Availability was the key factor in the choice of the Ada debugger. It is one of the more disheartening aspects of working with Ada95 at the current time that an effective debugger, public domain or commercial, is extremely difficult to find for the SunOS environment.

B. APPROACH

The CAPS team considered a number of factors when planning how to approach the implementation of the project. Automation was of prime importance to the group. Limited time in which to complete the effort and a shortage of available man-hours required that the maximum value of automated assistance be achieved.

The experience of senior members of the team in creating sophisticated makefiles was primary importance to achieving maximum automaton. A quick look at some of the code within the PSDL Editor Makefile will serve to demonstrate this. Eight of the major packages are created using Gen, one of them being modified by Sed in the process. A short script is available to run a test case directly following a compilation. Two scripts were created to allow for easy export of private versions to public directories, and for entering a current version into the RCS.

The other process used to enhance the effectiveness of the group was the early division of the four main designers into teams of two. The newer members of the group were then able to learn the system from experienced personnel while providing a second set of eyes to spot syntax and logic errors. The investment in time up front undoubtedly paid off in the end in reduced errors, team spirit, and knowledge learned.

C. PROCESS

The implementation process was performed at multiple sites over a period stretching from 9 June to 11 August 1997. While the majority of the effort was performed within the CAPS Laboratory and the Software Engineering Laboratory at

NPS, one team member was located in Lancaster, CA. Furthermore, at various times, other members were working from as far away as San Diego, CA and Brazil. This dispersion over time and distance complicated the problems of project management, especially in the initial phases and made the use of the RCS indispensable to success.

The parser creation project was the first part of the implementation phase undertaken. It was an elementary undertaking that involved the analyzing of an existing Aflex grammar for PSDL and removing the components that were not applicable to the parser at hand. This part of the project was very automated, taking advantage of the properties Awk, Sed, Aflex, and Ayacc to achieve success.

Testing of the parsers outside of the GE proved them to be accurate. Later testing within the GE found them to be less so. A default property within the GE caused streams not assigned a specific type to be listed as "?" in the graph description. The "?" had no meaning to the parsers and locating the error became a difficult process since the offending symbol was propagated to the graph description from within a window that is not normally parsed for errors.

The second phase of implementation involved the creation of the PSDL editor interface to the GE. This was a project of considerable detail. Implementation was simply a time consuming task occasionally punctuated with challenging problems. The most serious of these problems involved the GNAT compiler, version 3.09, and the attempt to improve storage management through use of Ada.Unbounded_Strings, which was new in Ada95. Repeated system crashes ultimately forced concentration on discovering what appeared to be a particularly difficult fault in the code. Extensive

debugging and testing found no code error, but rather an instability in the Unbounded_Strings package. This forced the abandonment of large segments of code created to make use of that functionality.

Software engineering princples were enforced during the implementation process and a number of poor programming practices performed during the original implementation of CAPS were corrected. There was a strong reliance on large omnibus utility packages in the original code. This clear violation of the principles of object oriented programming and information hiding was removed in all cases but one. This was not a trivial project. The break up of the GE utitilities package alone created 13 new packages. The one package not broken down is the PSDL concrete types package. This massive file would have required far more man-hours than was available to reduce to its component pieces. It is therefore slated for replacement during the next evolution of the system.

D. TESTING AND EVALUATION

Testing occurred concurrently with integration. This was done to achieve maximum benefit from working in parallel. Frequently during a project, programmers lose productive hours waiting for another team to complete testing of a required section of code. Our concurrent approach avoided this loss. It also ensured that the implementers received accurate and timely feedback on the viability of code modules, essential in preventing the propagation of errors throughout the code structure. Testing was conducted in cycles. When a version was exported to the public directories, testing was

performed and the results returned to the programming team. While the testing was being performed on a version, the implementers were correcting problems identified during the previous test cycle. During the course of integrating the SDE into CAPS, twelve testing cycles were completed, uncovering 75 errors that were then corrected. The improved version of CAPS was not released until three error-free testing cycles had been completed -- testing cycles without an error that could result in an abnormal termination of the program. While continued testing with real users was necessary to thoroughly test the design, the system had to be robust enough to allow work to proceed.

V. RESULTS

A. GRAPHICAL INTERFACE

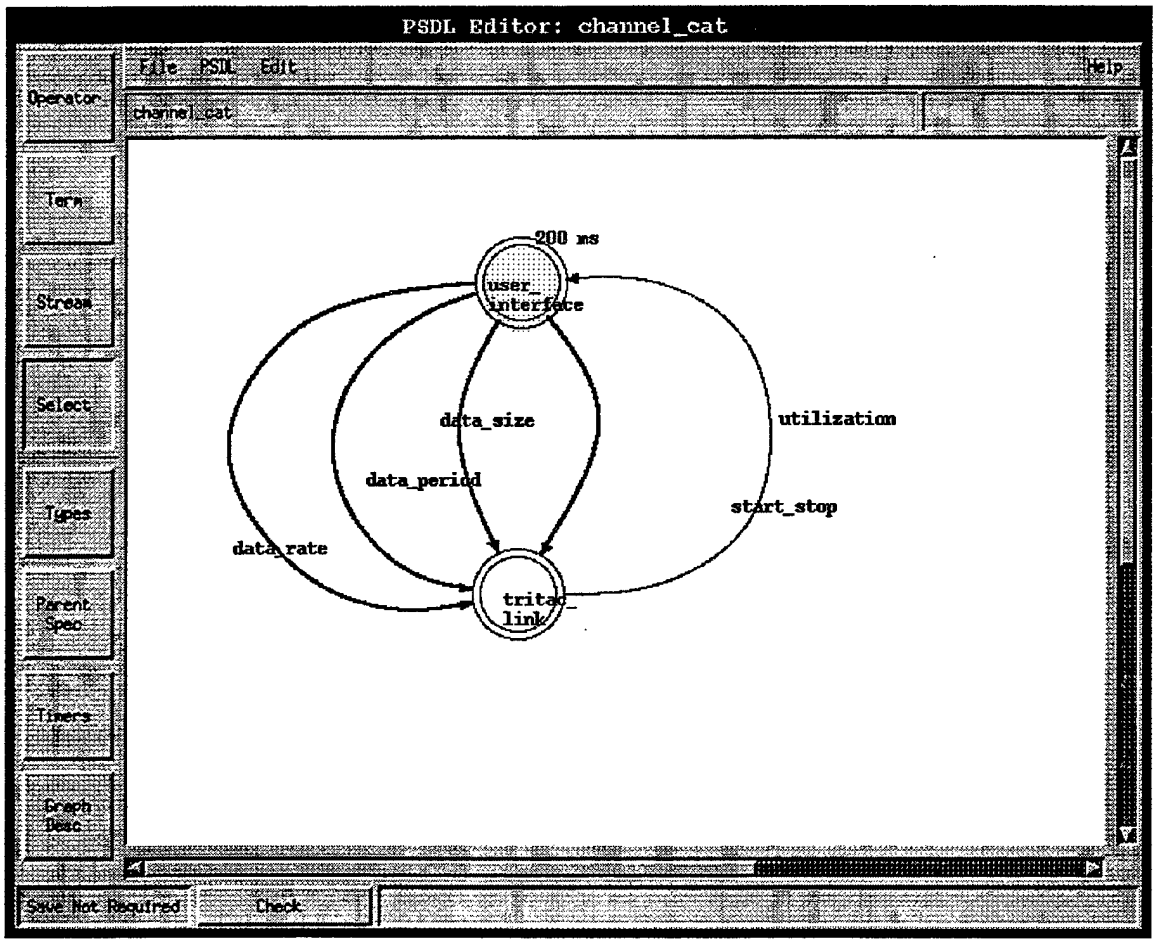


Figure 14. Improved Graphical Interface.

The primary objective of this project was the merging of the SDE and the Graphical Editor into one efficient module. That has been accomplished (Figure 14) and, without entering into detail appropriate to a user manual, a short display of the new features is in order. The user now can access the entire functionality for software prototype design through a single window. Although concentrating the functionality in

one window is a major benefit, the true value of the new GE becomes apparent when the prototype designer requires information. All the most frequently used information groups are placed at the designer's fingertips. This is the true benefit as it allows the designer to maintain focus on the task at hand.

Buttons for operators, streams and terminators still exist and perform the same functions. The properties and select buttons have been removed. Simply clicking on that object using the left mouse button does selection of a graph component. Accessing the

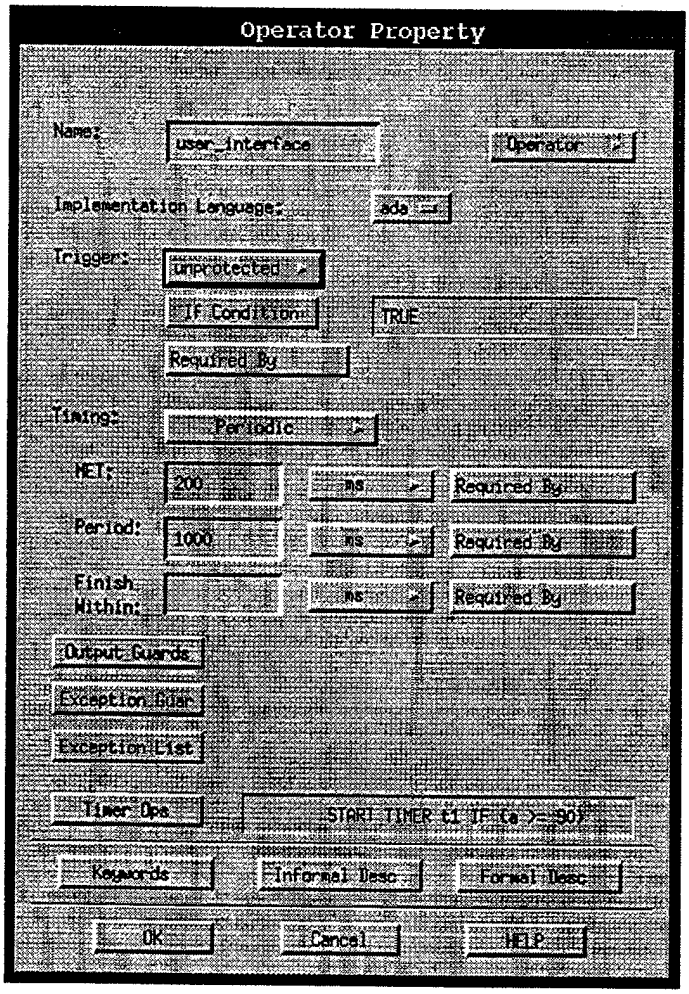


Figure 15. Operator Property Box.

properties of an operator or stream is now performed by clicking on that object using the right mouse button. This brings up the operator property box (Figure 15) or the stream property box (Figure 16). The old editor allowed only the modification of the Maximum Execution Time and the period from the properties box. Now available for modification are requirements, guards, triggers, implementation languages, descriptions and keywords. The more extensive variety of fields reflects not an increase in complexity of the editor, but a decrease in the requirement for hand coded PSDL components. Previously, the designer would be required to enter these attributes directly into the PSDL code in the PSDL SDE, with a corresponding loss of focus on the flow of the design and a significant increase in the possibility of creating an error. The new GE does not increase the functionality of the CAPS system in this case, but it makes the functionality that always existed much more accessible. Attributes that were invisible to the novice before are now readily available.

Added to the tool bar on the left of the improved GE are three new buttons;,

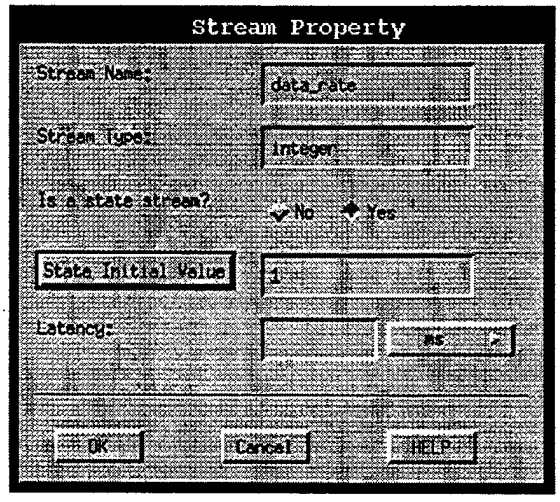


Figure 16. Stream Property Box.

PARENT SPEC, TIMERS and GRAPH DESC.

The first allows access to the timers tool box (Figure 17). The timers tool box enables the designer to view and modify timers from within the GE at the touch of a button. The tool box itself contains a suite of options for operating on timers. Prior to

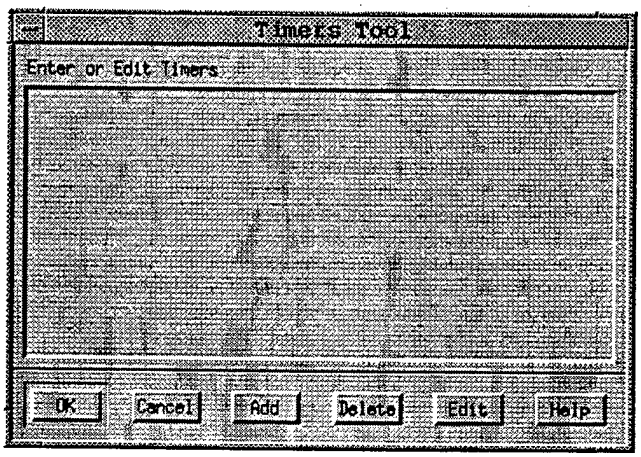


Figure 17. Timers Tool Box.

this modification, timers were created and edited within the PSDL editor, with a corresponding loss in focus from the task at hand.

The parent specification box (Figure 18) displays and allows editing of information pertaining to a parent composite operator. When working on a decomposition graph, it is frequently necessary to access the data of the parent operator in order to accurately set up the child graph. This used to entail extensive searching for the appropriate information within the PSDL code. Now it is available, and modifiable, directly from the interface.

The graph description button on the new GE's toolbar provides data on the PSDL graph. The PSDL graph contains a wealth of information pertaining to the operation of

the prototype. Previously, the designer was required to continually check the PSDL editor in order to gather the necessary information about the PSDL graph Time is saved and attention to the project at hand is retained.

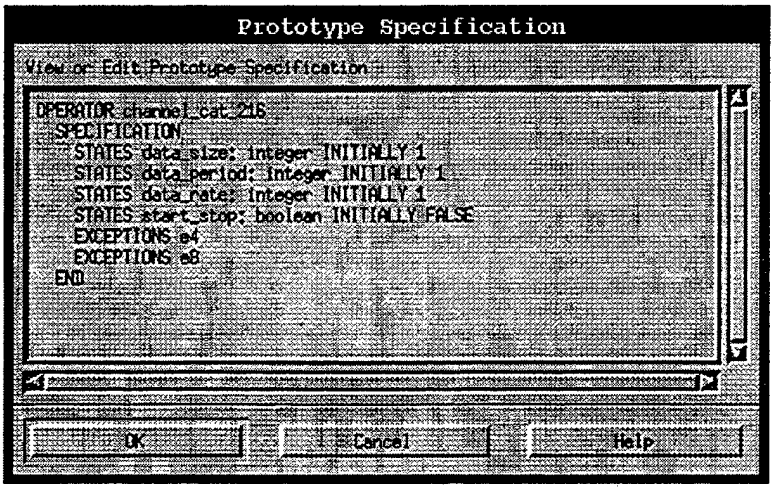


Figure 18. Parent Specification Box.

Lastly, the GE provides further two functions to assist the designer. At the bottom of the editor window (Figure 14.) is an information window and a button labeled "CHECK". The information window simply informs the designer in the event that the graph has changed since the last save. If it has, then the designer receives a gentle reminder to save the prototype. The "CHECK" button is an error checker. If an error is created that the parsers detect, the label will change to "ERROR MSGS". This is to alert the users that an mistake has been made in the PSDL code. This can occur without the syntax parsers alerting the designer when editing data directly on the graph. The user can then bring up the list of errors and correct them directly from the graph. If the button is depressed while in "CHECK" mode, the SDE will scan the PSDL program for errors and

return the result. In effect, the check button is a miniature compiler for PSDL that generates a report of correctness.

B. TECHNOLOGY TRANSFER

The Software Engineering Group at the Naval Postgraduate School is a full participant in the DOD Technology Transfer Program. CAPS version 1.1 has been in use by various DOD agencies since its creation in 1988. In accordance with this, on 12 August 1997, the CAPS version 1.2 with the enhanced syntax directed graphical editor was presented to members of the Army Research Office, the Army Research Lab and the Naval Research and Development activity for further testing and improvement. When follow up testing is completed, the new version will be disseminated to the various agencies within the DOD that support the rapid prototyping paradigm and use or support CAPS.

Transfer of CAPS technology within the DOD has resulted in the successful application of rapid prototyping and the CAPS environment to numerous projects. Currently the Navy's SmartShip and the Distributed Computing Networks programs are evaluating the use of CAPS in support of DOD research.

VI. SUMMARY AND RECOMMENDATIONS

A. SUMMARY

The integration of an improved SDE into the CAPS environment will decrease the overall time necessary to produce a software prototype and can substantially reduce the number of errors per thousand line of code. With proper use, a similar decrease can be achieved in both the time and development costs associated with software development through the use of CAPS in the early stages of a project. Four of the five original objectives were completed successfuly.

1. Creation of parsers that correctly interpret the seven main elements of PSDL.

The first objective completed, the creation of parsers for PSDL, was the simplest portion of the project. Extensive code reuse contributed to the accomplishment of this goal. Always a prime tenet of sound software engineering, the reuse of code in this case significantly reduced the time necessary to create the parsers, by reusing over 3000 lines of validated and tested code. The parsers themselves work perfectly and the error-locating mechanism properly presents the position and size of any faulty code the parsers discover.

2. **Use existing UNIX environment tools and software engineering procedures to automate coding of large segments of the project.**

The use of Awk, Aflex, Ayacc, and gen significantly reduced the amount of hand coding necessary to create the parsers needed. However, it was the creative use of the Unix make utility that allowed large portions of code creation to be automated.

3. **Integrate the new parsers into a previously designed graphical editor.**

Integration of the parsers into the Graphical Editor was completed and required relatively little effort.

4. **Integrate the combined graphical editor/parser package into the main flow of the CAPS environment.**

This goal required the most extensive work with the system. The multiple levels at which data was transferred to and from the graphical editor as well as the language conversion issues tended to make straightforward solutions impractical. Success was due to hard work and rigorous testing.

5. **Document flow control and design of data structures into a maintenance handbook to improve the maintenance of the enhanced version.**

Although an admirable goal and one that would have greatly enhanced the ability of future prototype designers to understand the nuances of the code for the SDE, time

constraints forced the postponement of this objective. The creation of a maintenance handbook will be left for future work.

B. LESSONS LEARNED

Planning ahead is a key to success in software development and this project was not an exception. Time not spent properly specifying requirements and outlining the design of the new system will be spent correcting errors in the testing and evaluation phase of development.

Proper use of Unix Makefiles to automate portions of a program was an enormous time saver and resulted in more accurate code. Effective use of makefiles can only be accomplished with thorough knowledge of the tools available and how to link them together, in effect creating an assembly line for the creation of code, much like that used in manufacturing. The power of an integrated tool set was demonstrated during this project. Even though only linked by shell commands and the make tool the capabilities were immense. This concept truly requires more emphasis in computer science curricula.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

CAPS is a versatile and powerful designing environment that can be used by the DOD now. However, the system could benefit from research in a number of areas. The DOD is moving toward Microsoft WindowsNT™ as the standard for information systems. It is essential that a version of CAPS that operates under WindowsNT is

produced soon. The graphical editor in CAPS would benefit from a new type of component, a composite stream, that would decrease the complexity of large designs. The graphical editor would also benefit from an auto-redraw function to assist the user in removing clutter. Implementing a CAPS prototype in CAPS would greatly improve the ability of the system to grow and evolve over time. Lastly, the PSDL parsers function admirably on PSDL components, but there is no mechanism in place to enforce strong typing within streams. Examining each of these areas in detail:

- Windows NT™ was first introduced to the network computer community in 1993. In 1995, NT was the operating system of choice on 36% of the network computers sold. By 1997 that percentage had increased to almost half [Ref 12]. Also in 1997, the DOD, and the U.S. Navy in particular through the Information Technology Initiative for the 21st Century (IT-21), opted for the WindowsNT operating system as the standard toward which the military information system will march. The time when we must count on a Unix compatible system being available to supporting agencies for the installation of CAPS is coming to an end. Therefore, it is critical that a IntelPC/Windows compatible version of CAPS be developed. Although this is a very large task and planning for the eventuality cannot begin too soon, the increasing usage of Microsoft Foundation Classes and their Ada95 equivalents will enable significant use of preexisting software components for the creation of a new graphical interface.

- As a prototype increases in complexity, the number of data streams linking operators with the graph also increases. While operators can be aggregated into a composite operator and decomposed in a separate graph, there is no such option to use

with streams. Indeed, composite operators often require a large number of data streams to feed the sub-functions imbedded in the lower levels. The proliferation of streams makes the graph unwieldy to create and manipulate, and in no way enhances the designer's overall knowledge of the design. The creation of a composite stream seems to be the obvious solution. Incorporating all streams between two operators, a composite stream would be subject to decomposition like an operator and would contain it's own graph containing the two terminating operators and all the streams linking them. This addition to CAPS would vastly enhance the functionality of the system especially at the higher levels of abstraction of the prototype design.

- In the absence of a composite stream, redrawing a prototype containing numerous streams is an arduous and time-consuming task. Users of CAPS are sometimes forced to create designs in a separate drawing tool in order to organize the components prior to drawing them within the graphical editor simply to avoid the time required to redraw streams. An automatic redraw tool that can create a neat and understandable output with a minimum of crossed streams would be of inestimable value to the system.

- CAPS was designed and is still evolving using the outmoded Waterfall methodology. There is also an unfortunate shortage of design information on the CAPS system. The combination of these two factors makes understanding the workings of the environment extraordinarily difficult for a designer viewing the code for the first time. Because of this state of affairs, evolutions to CAPS are necessarily major events requiring a substantial commitment of time and money. The creation of a CAPS prototype by reverse engineering the CAPS system would greatly relieve this problem. Although a

substantial project, the ability of the CAPS system to decompose the problem at successively finer levels of detail would allow the project to be spread over several theses. Alternately, it could be done as an ongoing class project in an advanced software-engineering course. This would have the added advantage of teaching new students the utility of the tool while improving it. Upon completion, CAPS would benefit from all the evolutionary advantages that accrue to projects using rapid prototyping.

- The parsers installed in the Graphical Editor perform quite well in detecting errors with PSDL components. However, they do not check for type conformity within operators or streams. For example, if the designer decides that a stream type will be an integer, the system still allows the designer to enter 1.2, a float, as the initial state variable. This will undoubtedly result in problems when create problems during compilation of generated Ada code. Enforcing strong typing for data types within the PSDL structure should be a minor fix and will have a significant impact upon the robustness of the system.

- The parsers installed in the Graphical Editor check the designer's input and, if an error is present, show the point where the error occurs. This is a functional and adequate system. However, a much more robust method of ensuring that the designer provides the necessary information in the correct format is by using buttons to display templates for the various PSDL constructs. The designer can be prompted to choose from different variations of a component. This would result in a minimal amount of hand entered data, with a corresponding drop in the number of hand entered errors.

APPENDIX A: SELECTED SOURCE CODE (PSDL TYPE)

```
--Makefile for PSDL_TYPE


INCLUDE_FLAGS = \
  -I./GENERIC_TYPES \
  -I./INSTANTIATIONS

GEN_ADA = \
  psdl_io.adb \
  old_psdl_io.adb \
  psdl_program_pkg.adb \
  pre_expander_pkg.adb \
  psdl_component_pkg.adb \
  psdl_concrete_type_pkg.adb \
  psdl_graph_pkg.adb \
  parser.adb

SPARSERS = \
  exception_guard_io.adb \
  exception_io.adb \
  expression_io.adb \
  op_id_io.adb \
  output_guard_io.adb \
  parser.adb \
  timer_op_guard_io.adb \
  type_name_io.adb

EXP_YACC = \
  expression_io_goto.ads \
  expression_io_shift_reduce.ads \
  expression_io_tokens.ads

TYPE_NAME_YACC = \
  type_name_io_goto.ads \
  type_name_io_shift_reduce.ads \
  type_name_io_tokens.ads

OP_ID_YACC = \
  op_id_io_goto.ads \
  op_id_io_shift_reduce.ads \
  op_id_io_tokens.ads

OUTPUT_GUARD_YACC = \
  output_guard_io_goto.ads \
  output_guard_io_shift_reduce.ads \
  output_guard_io_tokens.ads
```

```
EXCEPTION_GUARD_YACC = \
  exception_guard_io_goto.ads \
  exception_guard_io_shift_reduce.ads \
  exception_guard_io_tokens.ads

EXCEPTION_YACC = \
  exception_io_goto.ads \
  exception_io_shift_reduce.ads \
  exception_io_tokens.ads

TIMER_OP_GUARD_YACC = \
  timer_op_guard_io_goto.ads \
  timer_op_guard_io_shift_reduce.ads \
  timer_op_guard_io_tokens.ads

AYACC = \
  parser_goto.ads \
  parser_shift_reduce.ads \
  parser_tokens.ads \
  parser.g

AFLEX = \
  parser_lex_dfa.ads \
  parser_lex_dfa.adb \
  parser_lex_io.ads \
  parser_lex_io.adb \
  parser_lex.ads \
  parser_lex.adb

all: $(GEN_ADA) $(SPARSERS) $(AFLEX)
      (cd I* ; make gen)
      (cd GE* ; make gen)
      gnatmake -g -c $(INCLUDE_FLAGS) psdl_io.adb

gen: $(GEN_ADA)
      (cd I* ; make gen)
      (cd GE* ; make gen)

parsers: $(SPARSERS) $(AFLEX)

psdl_io.adb: psdl_io.g
      gen < psdl_io.g > psdl_io.adb

pre_expander_pkg.adb: pre_expander_pkg.g
      gen < pre_expander_pkg.g > pre_expander_pkg.adb

old_psdl_io.adb: old_psdl_io.g
      gen < old_psdl_io.g > old_psdl_io.adb

psdl_program_pkg.adb: psdl_program_pkg.g
      gen < psdl_program_pkg.g > psdl_program_pkg.adb
```

```
psdl_component_pkg.adb: psdl_component_pkg.g
	gen < psdl_component_pkg.g > psdl_component_pkg.adb

psdl_concrete_type_pkg.adb: psdl_concrete_type_pkg.g
	gen < psdl_concrete_type_pkg.g > psdl_concrete_type_pkg.adb

psdl_graph_pkg.adb: psdl_graph_pkg.g
	gen < psdl_graph_pkg.g > psdl_graph_pkg.adb

$(AFLEX): parser_lex.l
	aflex -s parser_lex.l
	gnatchop -w yylex.adb
	/bin/rm yylex.*

$(AYACC): parser.y
	ayacc parser.y debug =\> off verbose =\> on
	mv yyparse.adb parser.g
	/bin/rm yyparse.*

$(EXP_YACC) expression_io.g: expression_io.y
	ayacc expression_io.y debug =\> off verbose =\> on
	mv yyparse.adb expression_io.g
	/bin/rm yyparse.*

$(TYPE_NAME_YACC) type_name_io.g: type_name_io.y
	ayacc type_name_io.y debug =\> off verbose =\> on
	mv yyparse.adb type_name_io.g
	/bin/rm yyparse.*

$(OP_ID_YACC) op_id_io.g: op_id_io.y
	ayacc op_id_io.y debug =\> off verbose =\> on
	mv yyparse.adb op_id_io.g
	/bin/rm yyparse.*

$(OUTPUT_GUARD_YACC) output_guard_io.g: output_guard_io.y
	ayacc output_guard_io.y debug =\> off verbose =\> on
	mv yyparse.adb output_guard_io.g
	/bin/rm yyparse.*

$(EXCEPTION_GUARD_YACC) exception_guard_io.g: exception_guard_io.y
	ayacc exception_guard_io.y debug =\> off verbose =\> on
	mv yyparse.adb exception_guard_io.g
	/bin/rm yyparse.*

$(EXCEPTION_YACC) exception_io.g: exception_io.y
	ayacc exception_io.y debug =\> off verbose =\> on
	mv yyparse.adb exception_io.g
	/bin/rm yyparse.*

$(TIMER_OP_GUARD_YACC) timer_op_guard_io.g: timer_op_guard_io.y
	ayacc timer_op_guard_io.y debug =\> off verbose =\> on
	mv yyparse.adb timer_op_guard_io.g
	/bin/rm yyparse.*
```

```
expression_io.adb: expression_io.g $(AYACC) $(AFLEX)
	gen < expression_io.g > expression_io.a
	gnatchop -w expression_io.a
	@sed -f expression_io.sed expression_io.adb > tmpfile
	@mv tmpfile expression_io.adb
	/bin/rm expression_io.a

type_name_io.adb: type_name_io.g $(AYACC) $(AFLEX)
	gen < type_name_io.g > type_name_io.a
	gnatchop -w type_name_io.a
	@sed -f type_name_io.sed type_name_io.adb > tmpfile
	@mv tmpfile type_name_io.adb
	/bin/rm type_name_io.a

op_id_io.adb: op_id_io.g $(AYACC) $(AFLEX)
	gen < op_id_io.g > op_id_io.a
	gnatchop -w op_id_io.a
	@sed -f op_id_io.sed op_id_io.adb > tmpfile
	@mv tmpfile op_id_io.adb
	/bin/rm op_id_io.a

output_guard_io.adb: output_guard_io.g $(AYACC) $(AFLEX)
	gen < output_guard_io.g > output_guard_io.a
	gnatchop -w output_guard_io.a
	@sed -f output_guard_io.sed output_guard_io.adb > tmpfile
	@mv tmpfile output_guard_io.adb
	/bin/rm output_guard_io.a

exception_guard_io.adb: exception_guard_io.g $(AYACC) $(AFLEX)
	gen < exception_guard_io.g > exception_guard_io.a
	gnatchop -w exception_guard_io.a
	@sed -f exception_guard_io.sed exception_guard_io.adb > tmpfile
	@mv tmpfile exception_guard_io.adb
	/bin/rm exception_guard_io.a

exception_io.adb: exception_io.g $(AYACC) $(AFLEX)
	gen < exception_io.g > exception_io.a
	gnatchop -w exception_io.a
	@sed -f exception_io.sed exception_io.adb > tmpfile
	@mv tmpfile exception_io.adb
	/bin/rm exception_io.a

timer_op_guard_io.adb: timer_op_guard_io.g $(AYACC) $(AFLEX)
	gen < timer_op_guard_io.g > timer_op_guard_io.a
	gnatchop -w timer_op_guard_io.a
	@sed -f timer_op_guard_io.sed timer_op_guard_io.adb > tmpfile
	@mv tmpfile timer_op_guard_io.adb
	/bin/rm timer_op_guard_io.a

parser.adb: parser.g $(AYACC) $(AFLEX)
	gen < parser.g > parser.a
	gnatchop -w parser.a
	/bin/rm parser.a
```

```
# currently not used, old my_pkg
# add to GEN_ADA if used
psdl_ops_pkg.adb:
	gen < psdl_ops_pkg.g > psdl_ops_pkg.adb

ci: gen parsers
	ci_files -tRCS/desc *.[Cgly] *.ad[sb] Makefile
	sleep 1
	touch *.adb
	touch *.ali *.o [IG]*/*.ali [IG]*/*.o
	(cd GENERIC_TYPES ; make ci)
	(cd INSTANTIATIONS ; make ci)

test: all
	gnatmake -g -IINSTANTIATIONS -IGENERIC_TYPES test.adb
	./test > test.out

test2: all
	gnatmake -g -IINSTANTIATIONS -IGENERIC_TYPES test2.adb
	./test2 > test2.out

pre_expander: all
	gnatmake -g -IINSTANTIATIONS -IGENERIC_TYPES pre_expander.adb
	./pre_expander < autopilot.psdl > pre_expander.out

test3: all $(SPARSERS)
	gnatmake -g -IINSTANTIATIONS -IGENERIC_TYPES test3.adb
	./test3 > test3.out

test_text: all
	gnatmake -g -IINSTANTIATIONS -IGENERIC_TYPES test_text.adb
	./test_text > test_text.out
```

APPENDIX B: SELECTED SOURCE CODE (PARSERS)

```
#
# Makefile for PSDL Parsers:
#      . Expression
#      . Initial Expression
#      . Output Guard
#      . Timer_op Guard
#      . Exception Guard
#      . Exception List
#      . Type Spec
#      . Operator Spec
#      . Opid
#
# CAPS Lab.
# Jun/1997
#

SOURCES = parse_expression.y parse_init_expression.y
parse_output_guard.y parse_timer_op_guard.y parse_exception_guard.y
parse_exception_list.y parse_type_spec.y parse_oper_spec.y parse_opid.y
parse_lex.l parser_pkg.adb parser_pkg.ads

OBJ  = yyparse.ads parse_tokens.ads

COBJ = test.o test_exp.o test_init_exp.o test_output_guard.o
test_timer_op_guard.o test_exception_guard.o test_exception_list.o
test_type_spec.o test_oper_spec.o test_opid.o

OBJ1 = parse_expression_goto.ads parse_expression_shift_reduce.ads
OBJ2 = parse_init_expression_goto.ads
parse_init_expression_shift_reduce.ads
OBJ3 = parse_output_guard_goto.ads parse_output_guard_shift_reduce.ads
OBJ4 = parse_timer_op_guard_goto.ads
parse_timer_op_guard_shift_reduce.ads
OBJ5 = parse_exception_guard_goto.ads
parse_exception_guard_shift_reduce.ads
OBJ6 = parse_exception_list_goto.ads
parse_exception_list_shift_reduce.ads
OBJ7 = parse_type_spec_goto.ads parse_type_spec_shift_reduce.ads
OBJ8 = parse_oper_spec_goto.ads parse_oper_spec_shift_reduce.ads
OBJ9 = parse_opid_goto.ads parse_opid_shift_reduce.ads

SCRIPTS= Makefile parse.awk parse_io.awk

PARSER= yyparse.adb yylex.adb

parsers: parser_pkg.ali

test: parser_pkg.ali ${COBJ}
      gnatbind -n parser_pkg.ali
```

```
	gnatlink -o test parser_pkg.ali ${COBJ}

parser_pkg.ali: parser_pkg.adb ${PARSER} ${SCRIPTS} ${SOURCES} \
                ${OBJ1} ${OBJ2} ${OBJ3} ${OBJ4} ${OBJ5} ${OBJ6} ${OBJ7}
${OBJ8} ${OBJ9} ${OBJ}
	gnatmake parser_pkg.adb

$(OBJ1): parse_expression.y
	ayacc parse_expression.y debug =\> off verbose =\> on
	gnatchop -w yyparse.adb

$(OBJ2): parse_init_expression.y
	ayacc parse_init_expression.y debug =\> off verbose =\> on
	gnatchop -w yyparse.adb

$(OBJ3): parse_output_guard.y
	ayacc parse_output_guard.y debug =\> off verbose =\> on
	gnatchop -w yyparse.adb

$(OBJ4): parse_timer_op_guard.y
	ayacc parse_timer_op_guard.y debug =\> off verbose =\> on
	gnatchop -w yyparse.adb

$(OBJ5): parse_exception_guard.y
	ayacc parse_exception_guard.y debug =\> off verbose =\> on
	gnatchop -w yyparse.adb

$(OBJ6): parse_exception_list.y
	ayacc parse_exception_list.y debug =\> off verbose =\> on
	gnatchop -w yyparse.adb

$(OBJ7): parse_type_spec.y
	ayacc parse_type_spec.y debug =\> off verbose =\> on
	gnatchop -w yyparse.adb

$(OBJ8): parse_oper_spec.y
	ayacc parse_oper_spec.y debug =\> off verbose =\> on
	gnatchop -w yyparse.adb

$(OBJ9): parse_opid.y
	ayacc parse_opid.y debug =\> off verbose =\> on
	gnatchop -w yyparse.adb

	@sed -f expression.sed parser_pkg-parse_expression.adb > tmpfile
	@mv tmpfile parser_pkg-parse_expression.adb
	@sed -f init_expression.sed parser_pkg-parse_init_expression.adb
> tmpfile
	@mv tmpfile parser_pkg-parse_init_expression.adb
	@sed -f output_guard.sed parser_pkg-parse_output_guard.adb >
tmpfile
	@mv tmpfile parser_pkg-parse_output_guard.adb
	@sed -f timer_op_guard.sed parser_pkg-parse_timer_op_guard.adb >
tmpfile
	@mv tmpfile parser_pkg-parse_timer_op_guard.adb
```

```
	@sed -f exception_guard.sed parser_pkg-parse_exception_guard.adb
> tmpfile
	@mv tmpfile parser_pkg-parse_exception_guard.adb
	@sed -f exception_list.sed parser_pkg-parse_exception_list.adb >
tmpfile
	@mv tmpfile parser_pkg-parse_exception_list.adb
	@sed -f type_spec.sed parser_pkg-parse_type_spec.adb > tmpfile
	@mv tmpfile parser_pkg-parse_type_spec.adb
	@sed -f oper_spec.sed parser_pkg-parse_oper_spec.adb > tmpfile
	@mv tmpfile parser_pkg-parse_oper_spec.adb
	@sed -f opid.sed parser_pkg-parse_opid.adb > tmpfile
	@mv tmpfile parser_pkg-parse_opid.adb
	@yes | rm tmpfile

.IGNORE:

ci:
	ci_files -tRCS/desc ${SOURCES} ${SCRIPTS} ${OBJ}

yylex.ads yylex.adb: parse_lex.l parse.awk parse_io.awk
	aflex -s parse_lex.l
	nawk -f parse.awk yylex.adb > yylex.patched
	nawk -f parse_io.awk parse_lex_io.adb > temp
	mv temp parse_lex_io.adb
	gnatchop -w yylex.patched
```

```
--*********************
-- Alex file for Parser
--*********************

-------------------------------------------------------------------------
-- This file is the Aflex input file for PSDL grammar,
-- and defines the lexical tokens for the ayacc psdl parser.
-------------------------------------------------------------------------

-- Definitions of lexical classes

Digit   [0-9]
Int     {Digit}+
Letter      [a-zA-Z_]
Alpha   ({Letter}|{Digit})
Blank [ \t\n]
-- Text is anything but '{' and '}'
Text  [^{}]
-- StrLit is anything but '"' and '\' OR a '\' followed by '"' or a '\'
StrLit      [^"\\]|[\\]["\\]
Quote ["]

%%

axioms|AXIOMS                                   { return (AXIOMS_TOKEN); }
by{Blank}+all|BY{Blank}+ALL                     { return (BY_ALL_TOKEN);       }
by{Blank}+some|BY{Blank}+SOME                   { return (BY_SOME_TOKEN);      }
control|CONTROL                                 { return (CONTROL_TOKEN); }
constraints|CONSTRAINTS                         { return (CONSTRAINTS_TOKEN); }
data|DATA                                       { return (DATA_TOKEN); }
stream|STREAM                                   { return (STREAM_TOKEN); }
description|DESCRIPTION                         { return (DESCRIPTION_TOKEN); }
edge|EDGE                                       { return (EDGE_TOKEN); }
end|END                                         { return (END_TOKEN); }
exceptions|EXCEPTIONS                           { return (EXCEPTIONS_TOKEN); }
exception|EXCEPTION                             { return (EXCEPTION_TOKEN); }
finish|FINISH                                   { return (FINISH_TOKEN); }
within|WITHIN                                   { return (WITHIN_TOKEN); }
generic|GENERIC                                 { return (GENERIC_TOKEN); }
graph|GRAPH                                     { return (GRAPH_TOKEN); }
hours|HOURS                                     { return (HOURS_TOKEN); }
if|IF                                           { return (IF_TOKEN); }
implementation|IMPLEMENTATION                   { return (IMPLEMENTATION_TOKEN);
}ly|INITIALLY                                   { return (INITIALLY_TOKEN); }
iNPUT                                           { return (INPUT_TOKEN); }
ks|KEYWORDS                                     { return (KEYWORDS_TOKEN); }
m|MAXIMUM                                       { return (MAXIMUM_TOKEN); }
eon|EXECUTION                                   { return (EXECUTION_TOKEN); }
tME                                             { return (TIME_TOKEN); }
rse|RESPONSE                                    { return (RESPONSE_TOKEN); }
microsec|MICROSEC|microseconds|MICROSECONDS { return (MICROSEC_TOKEN);}
```

```
minimum|MINIMUM                                  { return (MINIMUM_TOKEN); }
calling{Blank}+period|CALLING{Blank}+PERIOD
                                          { return (CALL_PERIOD_TOKEN); }
min|MIN|minutes|MINUTES                     { return (MIN_TOKEN); }
ms|MS|milliseconds|MILLISECONDS             { return (MS_TOKEN); }
operator|OPERATOR                           { return (OPERATOR_TOKEN); }
output|OUTPUT                               { return (OUTPUT_TOKEN); }
period|PERIOD                               { return (PERIOD_TOKEN); }
property|PROPERTY                           { return (PROPERTY_TOKEN); }
required{Blank}+by|REQUIRED{Blank}+BY       { return (REQ_BY_TOKEN); }
reset{Blank}+timer|RESET{Blank}+TIMER       { return (RESET_TOKEN); }
sec|SEC|seconds|SECONDS                     { return (SEC_TOKEN); }
specification|SPECIFICATION           { return (SPECIFICATION_TOKEN); }
start{Blank}+timer|START{Blank}+TIMER       { return (START_TOKEN); }
states|STATES                               { return (STATES_TOKEN); }
stop{Blank}+timer|STOP{Blank}+TIMER         { return (STOP_TOKEN);}
timer|TIMER                                 { return (TIMER_TOKEN); }
triggered|TRIGGERED                        { return (TRIGGERED_TOKEN); }
type|TYPE                                   { return (TYPE_TOKEN); }
vertex|VERTEX                               { return (VERTEX_TOKEN); }

"?"                                         { return ('?'); }
"{"                                         { return ('{'); }
"}"                                         { return ('}'); }
"!"|"@"|"#"|"$"|"%"|"^"|"~"|"`"            { return (ILLEGAL_TOKEN); }
"("                                         { return ('('); }
")"                                         { return (')'); }
"["                                         { return ('['); }
"]"                                         { return (']'); }
":"                                         { return (':'); }
","                                         { return (','); }
"."                                         { return ('.'); }
"|"                                         { return ('|'); }
"->"                                        { return (ARROW); }

"and"|"AND"          { set_token_value((token_category => psdl_id_cat,
                                psdl_id_value => convert("AND") ));
                       return (AND_TOKEN); }
"or"|"OR"            { set_token_value((token_category => psdl_id_cat,
                                 psdl_id_value => convert("OR") ));
                       return (OR_TOKEN); }
"xor"|"XOR"          { set_token_value((token_category => psdl_id_cat,
                                psdl_id_value => convert("XOR") ));
                       return (XOR_TOKEN); }
">="                 { set_token_value((token_category => psdl_id_cat,
                                psdl_id_value => convert(">=") ));
                       return (GREATER_THAN_OR_EQUAL); }
"<="                 { set_token_value((token_category => psdl_id_cat,
                                 psdl_id_value => convert("<=") ));
                       return (LESS_THAN_OR_EQUAL); }
"/="|"~="            { set_token_value((token_category => psdl_id_cat,
                                 psdl_id_value => convert("/=") ));
                       return (INEQUALITY); }
"="                  { set_token_value((token_category => psdl_id_cat,
```

```
                                    psdl_id_value => convert("=") ));
                    return ('='); }
"+"                 { set_token_value((token_category => psdl_id_cat,
                                    psdl_id_value => convert("+") ));
                    return ('+'); }
"-"                 { set_token_value((token_category => psdl_id_cat,
                                    psdl_id_value => convert("-") ));
                    return ('-'); }
"*"                 { set_token_value((token_category => psdl_id_cat,
                                    psdl_id_value => convert("*") ));
                    return ('*'); }
"/"                 { set_token_value((token_category => psdl_id_cat,
                                    psdl_id_value => convert("/") ));
                    return ('/'); }
"&"                 { set_token_value((token_category => psdl_id_cat,
                                  psdl_id_value => convert("AND") ));
                    return ('&'); }
">"                 { set_token_value((token_category => psdl_id_cat,
                                    psdl_id_value => convert(">") ));
                    return ('>'); }
"<"                 { set_token_value((token_category => psdl_id_cat,
                                    psdl_id_value => convert("<") ));
                    return ('<'); }
"mod"|"MOD"         { set_token_value((token_category => psdl_id_cat,
                                  psdl_id_value => convert("MOD") ));
                    return (MOD_TOKEN); }
"rem"|"REM"         { set_token_value((token_category => psdl_id_cat,
                                  psdl_id_value => convert("REM") ));
                    return (REM_TOKEN); }
"**"                { set_token_value((token_category => psdl_id_cat,
                                  psdl_id_value => convert("**") ));
                    return (EXP_TOKEN); }
"abs"|"ABS"         { set_token_value((token_category => psdl_id_cat,
                                  psdl_id_value => convert("ABS") ));
                    return (ABS_TOKEN); }
"not"|"NOT"         { set_token_value((token_category => psdl_id_cat,
                                  psdl_id_value => convert("NOT") ));
                    return (NOT_TOKEN); }
true|TRUE           { set_token_value((token_category => psdl_id_cat,
                                 psdl_id_value => convert("TRUE") ));
                    return (TRUE); }
false|FALSE         { set_token_value((token_category => psdl_id_cat,
                                psdl_id_value => convert("FALSE") ));
                    return (FALSE); }


{Letter}{Alpha}*      { set_token_value((token_category => psdl_id_cat,
                                 psdl_id_value => convert(yytext) ));
                        return (IDENTIFIER); }

{Quote}{StrLit}*{Quote}  { set_token_value((token_category => text_cat,
                                  text_value => convert(yytext) ));
                           return (STRING_LITERAL); }
```

```
{Int}                   { set_token_value((token_category => integer_cat,
                           integer_value => string_to_integer(yytext) ));
                              return (INTEGER_LITERAL); }

{Int}"."{Int}             { set_token_value((token_category => text_cat,
                                    text_value => convert(yytext) ));
                              return (REAL_LITERAL); }

"{"{Text}*"}"             { set_token_value((token_category => text_cat,
                                    text_value => convert(yytext) ));
                              return (TEXT_TOKEN); }

[\n]                        { increment_line_number; }

[ \t]                       { null; }  -- ignore spaces and tabs


%%

  with parser_tokens;          use parser_tokens;
  with psdl_concrete_type_pkg;       use psdl_concrete_type_pkg;
  with psdl_id_pkg; use psdl_id_pkg;
  with  text_io;
package parser_lex is
  -- Global variables used by the lexical analyser
  line_number: positive := 1;
  num_errors: natural := 0;

  -- Initialization procedure to allow multiple calls from the parser
  procedure initialize_yylex;

  -- Line numbers to be used for error messages during parsing
  function current_line return positive;

  -- Lexical analyzer function generated by aflex
  function yylex return token;

  -- Procedure that increments line_numbers
  procedure increment_line_number;

end parser_lex;


  with text_io;          use text_io;
package body parser_lex is
  -- Externally visible subprograms
  procedure initialize_yylex is
    begin
      line_number := 1;        -- reset line_number to 1
      yy_init := true;         -- tell yylex to reinitialize itself
    end initialize_yylex;

  function current_line return positive is
```

```
    begin
      return(line_number);
    end current_line;

  procedure increment_line_number is
    begin
      line_number := line_number + 1;
    end increment_line_number;

  -- Functions and subprograms used in actions associated with tokens
  package nat_io is new integer_io(natural);

  function string_to_integer (digit_string: string) return natural is
    digit, value : natural := 0;
  begin
    for i in 1 ..  digit_string'length loop
      case digit_string(i) is
        when '0' => digit := 0;
        when '1' => digit := 1;
        when '2' => digit := 2;
        when '3' => digit := 3;
        when '4' => digit := 4;
        when '5' => digit := 5;
        when '6' => digit := 6;
        when '7' => digit := 7;
        when '8' => digit := 8;
        when '9' => digit := 9;
        when others => return value;
      end case;
      value := (10 * value) + digit;
    end loop;
    return value;
  end string_to_integer;


  procedure set_token_value(value: yystype) is
    -- Set the value of a token, like "$$ := value" in ayacc.
  begin
    yylval := value;
  end set_token_value;

  -- The generated yylex function goes here.

##

end parser_lex;
```

```
--**********************
-- Ayacc file for Parser
--**********************

-- $Header: /work/sde/PSDL_TYPE.NEW/RCS/parser.y,
-- v 1.19 1997/08/12 21:35:47 sde Exp sde $
-------------------------------------------------------------------------
--
-- This file is the ayacc source file for PSDL parser.
-------------------------------------------------------------------------
--

-- token declarations section

%token '?' '{' '}' ILLEGAL_TOKEN '(' ')' ','   '[' ']' ':'  '.' '|'
%token ARROW
%token TRUE FALSE
%token AXIOMS_TOKEN
%token BY_ALL_TOKEN REQ_BY_TOKEN BY_SOME_TOKEN
%token CALL_PERIOD_TOKEN CONTROL_TOKEN
%token CONSTRAINTS_TOKEN
%token DATA_TOKEN DESCRIPTION_TOKEN
%token EDGE_TOKEN END_TOKEN EXCEPTIONS_TOKEN
%token EXCEPTION_TOKEN EXECUTION_TOKEN
%token FINISH_TOKEN
%token GENERIC_TOKEN GRAPH_TOKEN
%token HOURS_TOKEN
%token IF_TOKEN IMPLEMENTATION_TOKEN
%token INITIALLY_TOKEN INPUT_TOKEN
%token KEYWORDS_TOKEN
%token MAXIMUM_TOKEN MINIMUM_TOKEN
%token MICROSEC_TOKEN
%token MIN_TOKEN MS_TOKEN  MOD_TOKEN
%token NOT_TOKEN
%token OPERATOR_TOKEN OR_TOKEN OUTPUT_TOKEN
%token PERIOD_TOKEN PROPERTY_TOKEN
%token RESET_TOKEN RESPONSE_TOKEN
%token SEC_TOKEN SPECIFICATION_TOKEN
%token START_TOKEN STATES_TOKEN STOP_TOKEN
%token STREAM_TOKEN
%token TIME_TOKEN
%token TIMER_TOKEN TRIGGERED_TOKEN TYPE_TOKEN
%token VERTEX_TOKEN
%token WITHIN_TOKEN
%token IDENTIFIER
%token INTEGER_LITERAL REAL_LITERAL
%token STRING_LITERAL
%token TEXT_TOKEN

-- operator precedences
-- left means group and evaluate from the left
%left AND_TOKEN OR_TOKEN XOR_TOKEN LOGICAL_OPERATOR
%left '<' '>' '=' GREATER_THAN_OR_EQUAL LESS_THAN_OR_EQUAL INEQUALITY
RELATIONAL_OPERATOR
```

```
%left '+' '-' '&' BINARY_ADDING_OPERATOR
%left UNARY_ADDING_OPERATOR
%left '*'  '/'  MOD_TOKEN REM_TOKEN MULTIPLYING_OPERATOR
%left EXP_TOKEN ABS_TOKEN NOT_TOKEN HIGHEST_PRECEDENCE_OPERATOR

%start start_symbol
-- this is an artificial start symbol, for initialization


-- declaration of the value type for the parser stack.

%with psdl_concrete_type_pkg, expression_pkg, psdl_id_pkg;
%use psdl_concrete_type_pkg, expression_pkg, psdl_id_pkg;

{
    type token_category_type is (integer_cat,
                                 text_cat,
                                 psdl_id_cat,
                                 psdl_id_sequence_cat,
                                 op_id_cat,
                                 operator_name_cat,
                                 opt_arg_cat,
                                 type_name_cat,
                            type_decl_cat,
                                 timer_op_id_cat,
                                 expression_cat,
                                 expression_seq_cat,
                                 property_map_cat,
                                 no_value_cat);

    type yystype (token_category: token_category_type := no_value_cat)
is
      record
         case token_category is
           -- lexical token attributes:
           when integer_cat =>
             integer_value: integer;
           when text_cat =>
             text_value: text;
           -- grammar psdl_id attributes:
           when psdl_id_cat =>
             psdl_id_value: psdl_id;
           when psdl_id_sequence_cat =>
             psdl_id_sequence_value: psdl_id_sequence;
           when op_id_cat =>
             op_id_value: op_id;
           when operator_name_cat =>
             type_name_part, op_name_part: psdl_id;
           when opt_arg_cat =>
             input_value, output_value: psdl_id_sequence;
           when type_name_cat =>
             type_name_value: type_name;
           when type_decl_cat =>
             type_decl_value: type_declaration;
```

```
          when timer_op_id_cat =>
              timer_op_id_value: timer_op_id;
            when expression_cat =>
              expression_value: expression;
            when expression_seq_cat =>
              expression_seq_value: expression_sequence;
            when property_map_cat =>
              property_map_value: init_map;
            when no_value_cat => null;
          end case;
        end record;
}

%%

start_symbol
        :   { the_program := empty_psdl_program; }
          psdl
        ;

psdl    : psdl component
            { if member(name(the_component), the_program)
              then yyerror("Component redefined: "
                           & convert(name(the_component)));
              else bind(name(the_component),
                        the_component, the_program);
              end if; }
        |
        ;

component
        : data_type
        | operator
        ;

data_type
        : TYPE_TOKEN IDENTIFIER
            { the_operation_map := empty_operation_map;
              is_specification := true; }
          type_spec
            { is_specification := false; }
          type_impl
            { -- Construct the psdl type using global variables.
              build_psdl_type($2.psdl_id_value,
                              the_ada_name,
                              the_imp_lang,
                              the_model,
                              the_data_structure,
                              the_operation_map,
                              the_type_gen_par,
                              the_keywords,
                              the_description,
                              the_axioms,
                              is_atomic_type,
```

```
                                the_component); }
        ;

type_spec
        :
          SPECIFICATION_TOKEN optional_generic_param optional_type_decl
          op_spec_list functionality END_TOKEN
        ;

optional_generic_param
        : GENERIC_TOKEN
            { the_type_decl := empty_type_declaration; }
          list_of_type_decl
            { the_type_gen_par := the_type_decl; }
        |   { the_type_gen_par := empty_type_declaration; }
        ;

optional_type_decl
        :   { the_type_decl := empty_type_declaration; }
          list_of_type_decl
            { the_model := the_type_decl; }
        |   { the_model := empty_type_declaration; }
        ;

op_spec_list
        : op_spec_list OPERATOR_TOKEN IDENTIFIER operator_spec
            { build_psdl_operator($3.psdl_id_value,
                                  to_ada_id($3.psdl_id_value),
                                  the_imp_lang,
                                  the_gen_par,
                                  the_gen_par_rb,
                                  the_keywords,
                                  the_description,
                                  the_axioms,
                                  the_input,
                                  the_output,
                                  the_state,
                                  the_initial_expression_map,
                                  the_exceptions,
                                  the_specified_met,
                                  the_input_rb,
                                  the_output_rb,
                                  the_state_rb,
                                  the_exception_rb,
                                  the_spec_met_rb,
                                  is_atomic => true,
                                  the_opr => the_operator);

              bind_operation ($3.psdl_id_value,
                              the_operator,
                              the_operation_map); }
        |
        ;
```

```
operator
        : OPERATOR_TOKEN IDENTIFIER
            { is_specification := true; }
          operator_spec
            { is_specification := false; }
          operator_impl
            { -- construct the psdl operator using the global variables
             build_psdl_operator($2.psdl_id_value,
                                 the_ada_name,
                                 the_imp_lang,
                                 the_gen_par,
                                 the_gen_par_rb,
                                 the_keywords,
                                 the_description,
                                 the_axioms,
                                 the_input,
                                 the_output,
                                 the_state,
                                 the_initial_expression_map,
                                 the_exceptions,
                                 the_specified_met,
                                 the_input_rb,
                                 the_output_rb,
                                 the_state_rb,
                                 the_exception_rb,
                                 the_spec_met_rb,
                                 the_graph,
                                 the_streams,
                                 the_timers,
                                 the_trigger_map,
                                 the_exec_guard,
                                 the_out_guard,
                                 the_excep_trigger,
                                 the_timer_op,
                                 the_per,
                                 the_fw,
                                 the_mcp,
                                 the_mrt,
                                 the_impl_desc,
                                 the_eg_rb,
                                 the_per_rb,
                                 the_fw_rb,
                                 the_mcp_rb,
                                 the_mrt_rb,
                                 the_o_rb,
                                 the_e_rb,
                                 the_reset_rb,
                                 the_start_rb,
                                 the_stop_rb,
                                 is_atomic_operator,
                                 the_component); }
        ;
```

```
operator_spec
        : SPECIFICATION_TOKEN
            { -- Initialize the variables used to
              -- build an operator spec.
              the_gen_par := empty_type_declaration;
              the_gen_par_rb := empty;
              the_input_rb := empty;
              the_output_rb := empty;
              the_state_rb := empty;
              the_exception_rb := empty;
              the_spec_met_rb := empty;
              the_input := empty_type_declaration;
              the_output := empty_type_declaration;
              the_state := empty_type_declaration;
              expression_sequence_pkg.empty(the_init_exp_seq);
              the_initial_expression_map := empty_init_map;
            the_exceptions := empty;
              the_specified_met := undefined_time; }
          interface
            { bind_initial_state(the_state, the_init_exp_seq,
                                 the_initial_expression_map); }
          functionality END_TOKEN
        ;

interface
        : interface attribute reqmts_trace
            { bind_spec_rb(the_attribute_type,
                           $2.psdl_id_sequence_value,
                           $3.psdl_id_sequence_value); }
        |
        ;

attribute
        : GENERIC_TOKEN
            { the_type_decl := the_gen_par; }
          list_of_type_decl
            { $$ := $3;
              the_gen_par := the_type_decl;
              the_attribute_type := gen_par; }
        | INPUT_TOKEN
            { the_type_decl := the_input; }
          list_of_type_decl
            { $$ := $3;
              the_input := the_type_decl;
              the_attribute_type := input; }
        | OUTPUT_TOKEN
            { the_type_decl := the_output; }
          list_of_type_decl
            { $$ := $3;
              the_output := the_type_decl;
              the_attribute_type := output; }
        | STATES_TOKEN
            { the_states_token_line := current_line;
            -- For error messages.
```

```
            the_states_token := convert(yytext);
            -- For error messages.
            the_type_decl := the_state; }
          list_of_type_decl
            { the_state := the_type_decl; }
          INITIALLY_TOKEN initial_expression_list
            { $$ := $3;
              expression_sequence_pkg.append(the_init_exp_seq,
                                             $6.expression_seq_value,
                                             the_init_exp_seq);
              the_attribute_type := state; }
        | EXCEPTIONS_TOKEN id_list
            { $$ := $2;
              psdl_id_set_pkg.union(the_exceptions,
                                    to_set($2.psdl_id_sequence_value),
                                    the_exceptions );
              the_attribute_type := exc; }
        | MAXIMUM_TOKEN EXECUTION_TOKEN TIME_TOKEN time
            { $$ := ( token_category => psdl_id_sequence_cat,
                      psdl_id_sequence_value => empty );
              if the_specified_met = undefined_time or
                 $4.integer_value < the_specified_met
              then the_specified_met := $4.integer_value;
              end if;
              the_attribute_type := met; }
              -- Time is converted into millisec .
        ;

  -- Initialization of the_type_decl is
  -- done by the callers of this rule.
list_of_type_decl
        : list_of_type_decl ',' type_decl
            { $$ := ( token_category => psdl_id_sequence_cat,
                      psdl_id_sequence_value =>
                   psdl_id_sequence_pkg.append($1.psdl_id_sequence_value,
                      $3.psdl_id_sequence_value )); }
        | type_decl
            { $$ := $1; }
        ;

type_decl
        : id_list ':' type_name
            { $$ := $1;
              bind_type_declaration($1.psdl_id_sequence_value,
                                    $3.type_name_value,
                             the_type_decl); }
        ;

type_name
        : IDENTIFIER
            { -- Save the previous value of the_type_decl.
            -- Needed because the list_of_type_decl below
            -- might contain nested type declarations.
            $$ := (token_category => type_decl_cat,
```

```
                    type_decl_value => the_type_decl);
                the_type_decl := empty_type_declaration; }
          '[' list_of_type_decl ']'
            { the_type_name :=
                create(name => $1.psdl_id_value,
                       formals => $4.psdl_id_sequence_value,
                       gen_par => the_type_decl);
            -- Now restore the previous value saved above.
            the_type_decl := $2.type_decl_value;
              $$ := (token_category => type_name_cat,
                     type_name_value => the_type_name); }
        | IDENTIFIER
            { the_type_name :=
                create(name => $1.psdl_id_value,
                       formals => psdl_id_sequence_pkg.empty,
                       gen_par => empty_type_declaration);
              $$ := (token_category =>  type_name_cat,
                     type_name_value =>  the_type_name); }
        ;

id_list
        : id_list ',' IDENTIFIER
            { $$ := ( token_category => psdl_id_sequence_cat,
                      psdl_id_sequence_value =>
                        add($3.psdl_id_value,
                            $1.psdl_id_sequence_value )); }
        | IDENTIFIER
            { $$ := ( token_category => psdl_id_sequence_cat,
                        psdl_id_sequence_value => add($1.psdl_id_value,
                        empty) ); }
        ;

reqmts_trace
        : REQ_BY_TOKEN id_list
          { $$ := $2; }
        |
          { $$ := ( token_category => psdl_id_sequence_cat,
                    psdl_id_sequence_value => empty ); }
        ;

functionality
        : keywords informal_desc formal_desc
        ;

keywords
        : KEYWORDS_TOKEN id_list
            { the_keywords := to_set($2.psdl_id_sequence_value); }
        |   { the_keywords := empty; }
        ;

informal_desc
        : DESCRIPTION_TOKEN TEXT_TOKEN
            { if is_specification then
                the_description := remove_braces($2.text_value);
```

```
            else the_impl_desc := remove_braces($2.text_value);
            end if; }
        |
            { if is_specification then
                the_description := empty;
            else the_impl_desc := empty;
            end if; }
        ;

formal_desc
        : axioms_TOKEN TEXT_TOKEN
            { the_axioms:= remove_braces($2.text_value); }
        |
            { the_axioms:= empty; }
        ;

type_impl
        : IMPLEMENTATION_TOKEN IDENTIFIER IDENTIFIER END_TOKEN
            { is_atomic_type := true;
              the_imp_lang := $2.psdl_id_value;
              the_ada_name := to_ada_id($3.psdl_id_value); }
        | IMPLEMENTATION_TOKEN type_name op_impl_list END_TOKEN
            { is_atomic_type := false;
              the_data_structure := $2.type_name_value; }
        ;

op_impl_list
        : op_impl_list OPERATOR_TOKEN IDENTIFIER operator_impl
            { -- add implementation part to the operator in the
operation map
              add_op_impl_to_op_map($3.psdl_id_value,
                                    the_ada_name,
                                    is_atomic_operator,
                                    the_operation_map,
                                    the_graph,
                                    the_streams,
                                    the_timers,
                                    the_trigger_map,
                                    the_exec_guard,
                                    the_out_guard,
                                    the_excep_trigger,
                                    the_timer_op,
                                    the_per,
                                    the_fw,
                                    the_mcp,
                                    the_mrt,
                                    the_impl_desc ); }
        |
        ;

operator_impl
        : IMPLEMENTATION_TOKEN IDENTIFIER IDENTIFIER END_TOKEN
            { is_atomic_operator := true;
              the_imp_lang := $2.psdl_id_value;
```

```
            the_ada_name := to_ada_id($3.psdl_id_value); }
        | IMPLEMENTATION_TOKEN psdl_impl END_TOKEN
            { is_atomic_operator := false; }
        ;

psdl_impl
        : data_flow_diagram streams timers control_constraints
informal_desc
        ;

data_flow_diagram
        : { the_graph := empty_psdl_graph; }
          GRAPH_TOKEN vertex_list edge_list
        ;

                -- Time is the maximum execution time.
vertex_list
        : vertex_list VERTEX_TOKEN op_id optional_time graph_properties
            { the_graph := psdl_graph_pkg.add_vertex($3.op_id_value,
                                                     the_graph,
                                                     $4.integer_value,

$5.property_map_value); }
        |
        ;

                -- Time is the latency.
edge_list
        : edge_list EDGE_TOKEN IDENTIFIER
          optional_time op_id ARROW op_id graph_properties
            { the_graph := psdl_graph_pkg.add_edge($5.op_id_value,
                                                   $7.op_id_value,
                                                   $3.psdl_id_value,
                                                   the_graph,
                                                   $4.integer_value,

$8.property_map_value); }
        |
        ;

graph_properties
        : graph_properties PROPERTY_TOKEN IDENTIFIER '=' expression
            { bind($3.psdl_id_value, $5.expression_value,
             $1.property_map_value);
              $$ := ( token_category => property_map_cat,
                      property_map_value => $1.property_map_value ); }
        |
            { $$ := ( token_category => property_map_cat,
                      property_map_value => empty_init_map ); }
        ;

op_id
        : operator_name opt_arg
            { $$ := ( token_category => op_id_cat,
```

```
                    op_id_value =>
                    ( type_name => $1.type_name_part,
                      operation_name => $1.op_name_part,
                      inputs => $2.input_value,
                      outputs => $2.output_value )); }
        ;

operator_name
        : IDENTIFIER '.' IDENTIFIER
            { $$ := ( token_category => operator_name_cat,
                      type_name_part => $1.psdl_id_value,
                      op_name_part => $3.psdl_id_value ); }
        | IDENTIFIER
            { $$ := ( token_category => operator_name_cat,
                      type_name_part => empty,
                      op_name_part => $1.psdl_id_value ); }
        ;

opt_arg
        : '(' optional_id_list '|' optional_id_list ')'
            { $$ := ( token_category => opt_arg_cat,
                      input_value => $2.psdl_id_sequence_value,
                      output_value => $4.psdl_id_sequence_value ); }
        |   { $$ := ( token_category => opt_arg_cat,
                      input_value => empty,
                      output_value => empty ); }
        ;

optional_id_list
        : id_list { $$ := $1; }
        | { $$ := ( token_category => psdl_id_sequence_cat,
                    psdl_id_sequence_value => empty ); }
        ;

optional_time
        : ':' time
            { $$ := (token_category => integer_cat,
                     integer_value => $2.integer_value); }
        |   { $$:= (token_category => integer_cat,
                    integer_value => undefined_time); }
        ;

streams
        : DATA_TOKEN STREAM_TOKEN
            { the_type_decl := empty_type_declaration; }
          list_of_type_decl
            { the_streams := the_type_decl; }
        |
            { the_streams := empty_type_declaration; }
        ;


        ------------------------------------------
        -- The order of id's is not important, so
```

```
        -- we use psdl_id_set as the data structure
        -- to store the timers.
        ------------------------------------------
timers
        : TIMER_TOKEN id_list
            { the_timers := to_set($2.psdl_id_sequence_value); }
        |   { the_timers := empty; }
        ;

control_constraints
        : CONTROL_TOKEN CONSTRAINTS_TOKEN
            { the_trigger_map := empty_trigger_map;
              the_per := empty_timing_map;
              the_fw := empty_timing_map;
              the_mcp := empty_timing_map;
              the_mrt := empty_timing_map;
              the_exec_guard := empty_exec_guard_map;
              the_out_guard := empty_out_guard_map;
              the_excep_trigger := empty_excep_trigger_map;
              the_timer_op := empty_timer_op_map;
              the_eg_rb := empty;
              the_per_rb := empty;
              the_fw_rb := empty;
              the_mcp_rb := empty;
              the_mrt_rb := empty;
              the_o_srbm := empty;
              the_e_srbm := empty;
              the_reset_srbm := empty;
              the_start_srbm := empty;
              the_stop_srbm := empty;
              the_o_rb := empty;
              the_e_rb := empty;
              the_reset_rb := empty;
              the_start_rb := empty;
              the_stop_rb := empty; }
          constraints
        ;

constraints
        : constraints OPERATOR_TOKEN op_id
            { the_operator_id := $3.op_id_value;
              the_timer_op_set := timer_op_set_pkg.empty; }
          opt_trigger opt_period opt_finish_within
          opt_mcp opt_mrt constraint_options
            { bind(the_operator_id, the_o_srbm, the_o_rb);
              bind(the_operator_id, the_e_srbm, the_e_rb);
              bind(the_operator_id, the_reset_srbm, the_reset_rb);
              bind(the_operator_id, the_start_srbm, the_start_rb);
              bind(the_operator_id, the_stop_srbm, the_stop_rb); }
        |
        ;

constraint_options
        : constraint_options OUTPUT_TOKEN
```

```
          id_list IF_TOKEN expression reqmts_trace
            { the_output_id.op := the_operator_id;
              for id: psdl_id in
                  psdl_id_sequence_pkg.scan($3.psdl_id_sequence_value)
              loop
                  the_output_id.stream := id;
                  bind(the_output_id, $5.expression_value,
                      the_out_guard);
                  bind(id, $6.psdl_id_sequence_value, the_o_srbm);
              end loop; }
        | constraint_options EXCEPTION_TOKEN IDENTIFIER
          opt_if_predicate reqmts_trace
            { the_excep_id.op := the_operator_id;
              the_excep_id.excep := $3.psdl_id_value;
              bind(the_excep_id, $4.expression_value,
                  the_excep_trigger);
              bind($3.psdl_id_value, $5.psdl_id_sequence_value,
                  the_e_srbm); }
        | constraint_options timer_op IDENTIFIER
          opt_if_predicate reqmts_trace
            { the_timer_op_record.op_id := $2.timer_op_id_value;
              the_timer_op_record.timer_id := $3.psdl_id_value;
              the_timer_op_record.guard := $4.expression_value;
              timer_op_set_pkg.add (the_timer_op_record,
                                    the_timer_op_set);
              bind(the_operator_id, the_timer_op_set, the_timer_op);
              case the_timer_op_record.op_id is
                when t_reset =>
                  bind($3.psdl_id_value, $5.psdl_id_sequence_value,
the_reset_srbm);
                when t_start =>
                  bind($3.psdl_id_value, $5.psdl_id_sequence_value,
the_start_srbm);
                when t_stop =>
                  bind($3.psdl_id_value, $5.psdl_id_sequence_value,
the_stop_srbm);
                when t_none => null;
                  -- This case is impossible but the compiler can't
recognize that.
              end case; }
        |
        ;

opt_trigger
        : TRIGGERED_TOKEN trigger opt_if_predicate reqmts_trace
          { bind(the_operator_id, $3.expression_value, the_exec_guard);
            bind(the_operator_id, $4.psdl_id_sequence_value,
the_eg_rb); }
        |
        ;


trigger
        : BY_ALL_TOKEN id_list
```

```
            { the_trigger.tt := by_all;
              the_trigger.streams := to_set($2.psdl_id_sequence_value);
              bind(the_operator_id, the_trigger, the_trigger_map); }
        | BY_SOME_TOKEN id_list
            { the_trigger.tt := by_some;
              the_trigger.streams := to_set($2.psdl_id_sequence_value);
              bind(the_operator_id, the_trigger, the_trigger_map); }
        |   { the_trigger.tt := by_none;
              the_trigger.streams := empty;
              bind(the_operator_id, the_trigger, the_trigger_map); }
        ;


opt_period
        : PERIOD_TOKEN time reqmts_trace
            { bind(the_operator_id, $2.integer_value, the_per);
              bind(the_operator_id, $3.psdl_id_sequence_value,
the_per_rb); }
        |
        ;

opt_finish_within
        : FINISH_TOKEN WITHIN_TOKEN time reqmts_trace
            { bind(the_operator_id, $3.integer_value, the_fw);
              bind(the_operator_id, $4.psdl_id_sequence_value,
              the_fw_rb); }
        |
        ;

opt_mcp
        : MINIMUM_TOKEN CALL_PERIOD_TOKEN time reqmts_trace
            { bind(the_operator_id, $3.integer_value, the_mcp);
              bind(the_operator_id, $4.psdl_id_sequence_value,
              the_mcp_rb); }
        |
        ;

opt_mrt
        : max_resp_time time reqmts_trace
            { bind(the_operator_id, $2.integer_value, the_mrt);
              bind(the_operator_id, $3.psdl_id_sequence_value,
              the_mrt_rb); }
        |
        ;

max_resp_time
        : MAXIMUM_TOKEN RESPONSE_TOKEN TIME_TOKEN
        ;

timer_op
        : RESET_TOKEN
            { $$ := (token_category => timer_op_id_cat,
                     timer_op_id_value => t_reset); }
        | START_TOKEN
```

```
            { $$ := (token_category => timer_op_id_cat,
                     timer_op_id_value => t_start); }
        | STOP_TOKEN
            { $$ := (token_category => timer_op_id_cat,
                     timer_op_id_value => t_stop); }
        ;

opt_if_predicate
        : IF_TOKEN expression
          { $$ := (token_category => expression_cat,
                   expression_value => $2.expression_value); }
        | { $$ := (token_category => expression_cat,
                   expression_value => true_expression); }
        ;

       ------------------------------------------------------------------
---
       -- The expression sequence
       -- is used by procedure bind_initial_state together with
       -- the states map to construct the init_map.
       ------------------------------------------------------------------
---

initial_expression_list
        : initial_expression_list ',' initial_expression
            { $$ := (token_category =>  expression_seq_cat,
                     expression_seq_value =>
                       expression_sequence_pkg.add($3.expression_value,

$1.expression_seq_value )); }
        | initial_expression
            { $$ := (token_category => expression_seq_cat,
                     expression_seq_value =>
                       expression_sequence_pkg.add($1.expression_value,
                                                   empty_exp_seq )); }
        ;

       ------------------------------------------------------------------
---
       -- There is one and only one initial state(initial expression)
       -- for each state variable. This production returns one
       -- expression to the parent rule corresponding to one state.
       -- This is done by using the internal stack ($$ convention).
       ------------------------------------------------------------------
---

initial_expression
        : TRUE
            { $$ := (token_category => expression_cat,
                     expression_value => true_expression); }
        | FALSE
            { $$ := (token_category => expression_cat,
                     expression_value => false_expression); }
        | INTEGER_LITERAL
```

```
        { $$ := (token_category => expression_cat,
                 expression_value =>
                 create_integer_literal($1.integer_value)); }
    | REAL_LITERAL
        { $$ := (token_category => expression_cat,
                 expression_value =>
                 create_real_literal($1.text_value)); }
    | STRING_LITERAL
        { $$ := (token_category => expression_cat,
                 expression_value =>
                 create_string_literal($1.text_value)); }
    | IDENTIFIER
        { $$ := (token_category => expression_cat,
                 expression_value =>
                 create_identifier($1.psdl_id_value)); }
    | type_name '.' IDENTIFIER
        { $$ := (token_category => expression_cat,
                 expression_value =>
                 create_function_call($1.type_name_value,
                                      psdl_id($3.psdl_id_value),
                                      empty_exp_seq)); }
    | type_name '.' IDENTIFIER '(' initial_expression_list ')'
        { $$ := (token_category => expression_cat,
                 expression_value =>
                 create_function_call($1.type_name_value,
                                      psdl_id($3.psdl_id_value),
                                      $5.expression_seq_value));}
    | '(' initial_expression ')'
        { $$ := (token_category => expression_cat,
                 expression_value => $2.expression_value) ;
                   }
    | initial_expression log_op initial_expression
      %prec logical_operator
        { $$ := (token_category => expression_cat,
                 expression_value =>
                 create_binary_op ($1.expression_value,
                                   $2.psdl_id_value,
                                   $3.expression_value));
}
    | initial_expression rel_op initial_expression
      %prec relational_operator
        { $$ := (token_category => expression_cat,
                 expression_value =>
                 create_binary_op ($1.expression_value,
                                   $2.psdl_id_value,
                                   $3.expression_value));
}
    | '-' initial_expression
       %prec unary_adding_operator
        { $$ := (token_category => expression_cat,
                 expression_value =>
                   create_unary_op(convert("-"),
                                   $2.expression_value )); }
    | '+' initial_expression
```

```
          %prec unary_adding_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                       create_unary_op(convert("+"),
                                       $2.expression_value ));}
        | initial_expression bin_add_op initial_expression
          %prec binary_adding_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_binary_op ($1.expression_value,
                                       $2.psdl_id_value,
                                       $3.expression_value)); }
        | initial_expression bin_mul_op initial_expression
          %prec multiplying_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_binary_op ($1.expression_value,
                                       $2.psdl_id_value,
                                       $3.expression_value)); }
        | initial_expression EXP_TOKEN initial_expression
          %prec highest_precedence_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_binary_op ($1.expression_value,
                                       convert("**"),
                                       $3.expression_value));
}
        | NOT_TOKEN initial_expression
          %prec highest_precedence_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                       create_unary_op(convert("NOT"),
                                       $2.expression_value )); }
        | ABS_TOKEN initial_expression
          %prec highest_precedence_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                       create_unary_op(convert("ABS"),
                                       $2.expression_value )); }
        | '?'  { $$ := (token_category => expression_cat,
                     expression_value => undefined_expression ); }
        ;

log_op
        : AND_TOKEN
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("AND") ); }
        | OR_TOKEN
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("OR") ); }
        | XOR_TOKEN
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("XOR") ); }
        ;
```

```
rel_op
        : '<'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("<") ); }
        | '>'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert(">") ); }
        | '='
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("=") ); }
        | GREATER_THAN_OR_EQUAL
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert(">=") ); }
        | LESS_THAN_OR_EQUAL
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("<=") ); }
        | INEQUALITY
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("/=") ); }
        ;

bin_add_op
        : '+'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("+") ); }
        | '-'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("-") ); }
        | '&'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("&") ); }
        ;

bin_mul_op
        : '*'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("*") ); }
        | '/'
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("/") ); }
        | MOD_TOKEN
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("MOD") ); }
        | REM_TOKEN
            { $$ := (token_category => psdl_id_cat,
                     psdl_id_value => convert("REM") ); }
        ;


time
        : time_number MICROSEC_TOKEN
            { $$ := (token_category => integer_cat,
                     integer_value => ($1.integer_value + 999)/1000); }
```

```
        | time_number MS_TOKEN
            { $$ := (token_category => integer_cat,
                     integer_value => $1.integer_value); }
        | time_number SEC_TOKEN
            { $$ := (token_category => integer_cat,
                     integer_value => $1.integer_value * 1000); }
        | time_number MIN_TOKEN
            { $$ := (token_category => integer_cat,
                     integer_value => $1.integer_value * 60000); }
        | time_number HOURS_TOKEN
            { $$ := (token_category => integer_cat,
                     integer_value => $1.integer_value * 3600000); }
        ;

time_number
        : INTEGER_LITERAL
            { $$ := (token_category => integer_cat,
                     integer_value => $1.integer_value); }
        ;


expression_list
        : expression_list ',' expression
            { $$ := (token_category =>  expression_seq_cat,
                     expression_seq_value =>
                     expression_sequence_pkg.add($3.expression_value,

$1.expression_seq_value )); }
        | expression
            { $$ := (token_category =>  expression_seq_cat,
                     expression_seq_value =>
                     expression_sequence_pkg.add($1.expression_value,
                                                 empty_exp_seq )); }
        ;

-------------------------------------------------------------------
-- Expressions can appear in guards appearing in control constraints.
-- These guards can be associated with triggering conditions, or
-- conditional outputs, conditional exceptions, or conditional timer
-- operations. Similar to initial expression, except that tim e values
-- and references to timers and data streams are allowed.
-------------------------------------------------------------------
expression
        : TRUE
            { $$ := (token_category => expression_cat,
                     expression_value => true_expression); }
        | FALSE
            { $$ := (token_category => expression_cat,
                     expression_value => false_expression); }
        | INTEGER_LITERAL
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_integer_literal($1.integer_value)); }
        | REAL_LITERAL
```

```
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_real_literal($1.text_value)); }
| STRING_LITERAL
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_string_literal($1.text_value)); }
| IDENTIFIER
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_identifier($1.psdl_id_value)); }
 -- The only difference from the initial expression
| time
    { $$ := (token_category => expression_cat,
             expression_value =>
    create_time_literal (natural($1.integer_value)));}
| type_name '.' IDENTIFIER
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_function_call($1.type_name_value,
                                  psdl_id($3.psdl_id_value),
                                  empty_exp_seq)); }
| type_name '.' IDENTIFIER '(' expression_list ')'
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_function_call($1.type_name_value,
                                  psdl_id($3.psdl_id_value),
                                  $5.expression_seq_value));}
| '(' expression ')'
    { $$ := (token_category => expression_cat,
             expression_value => $2.expression_value);
              }
| expression log_op expression   %prec logical_operator
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_binary_op ($1.expression_value,
                               $2.psdl_id_value,
                               $3.expression_value)); }
| expression rel_op expression %prec relational_operator
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_binary_op ($1.expression_value,
                               $2.psdl_id_value,
                               $3.expression_value)); }
| '-' expression                %prec unary_adding_operator
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_unary_op  (convert("-"),
                               $2.expression_value)); }
| '+' expression                 %prec unary_adding_operator
    { $$ := (token_category => expression_cat,
             expression_value =>
             create_unary_op  (convert("+"),
                               $2.expression_value)); }
```

```
        | expression bin_add_op expression
          %prec binary_adding_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_binary_op ($1.expression_value,
                                       $2.psdl_id_value,
                                       $3.expression_value)); }
        | expression bin_mul_op expression
          %prec multiplying_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_binary_op ($1.expression_value,
                                       $2.psdl_id_value,
                                       $3.expression_value)); }
        | expression EXP_TOKEN expression
          %prec highest_precedence_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_binary_op ($1.expression_value,
                                       convert("**"),
                                       $3.expression_value)); }
        | NOT_TOKEN expression          %prec
highest_precedence_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_unary_op  (convert("NOT"),
                                       $2.expression_value)); }
        | ABS_TOKEN expression          %prec
highest_precedence_operator
            { $$ := (token_category => expression_cat,
                     expression_value =>
                     create_unary_op  (convert("ABS"),
                                       $2.expression_value)); }
        | '?'  { $$ := (token_category => expression_cat,
                     expression_value => undefined_expression ); }

        ;


%%

----------============================================================----------
--
--                          package spec parser
--
----------============================================================----------
  with psdl_program_pkg; use psdl_program_pkg;
  with text_io; use text_io;
package parser is
  procedure get(item: in out psdl_program);
  procedure get(file: in file_type; item: in out psdl_program);

  syntax_error: exception;
  semantic_error: exception;
```

```
end parser;

----------==========================================----------
--
--                          package body  parser
--
----------==========================================----------

  with parser_tokens; use parser_tokens;
  with parser_goto; use parser_goto;
  with parser_shift_reduce; use parser_shift_reduce;
  with parser_lex; use parser_lex;
  with parser_lex_dfa; use parser_lex_dfa;
  with psdl_component_pkg; use psdl_component_pkg;
  with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
  with psdl_id_pkg; use psdl_id_pkg;
  with ada_id_pkg; use ada_id_pkg;
  with psdl_graph_pkg; use psdl_graph_pkg;
  with expression_pkg; use expression_pkg;
  with spec_req_map_pkg; use spec_req_map_pkg;
  with cc_req_map_pkg; use cc_req_map_pkg;
  with cc_req_map_map_pkg; use cc_req_map_map_pkg;
  with psdl_io; use psdl_io;
package body parser is
  subtype exp_seq is expression_sequence_pkg.sequence;

  function empty_exp_seq return expression_sequence
    renames expression_sequence_pkg.empty;
    -- Returns an empty expression sequence.

  type attribute_type is (gen_par, input, output, state, exc, met);

  -- global variables used by the parser.
  the_program: psdl_program;
  the_component: psdl_component;
  the_operator: operator;
  the_atomic_type: atomic_type;
  the_atomic_operator: atomic_operator;
  the_composite_type: composite_type;
  the_composite_operator: composite_operator;
  the_ada_name: ada_id;
  the_imp_lang: psdl_id;
  the_gen_par: type_declaration;
  the_type_gen_par: type_declaration;
  the_keywords: psdl_id_set;
  the_description: text;
  the_axioms: text;
  the_output_id: output_id;
    -- a temporary variable to hold output_id to construct out_guard
map
  the_excep_id: excep_id;
    -- a temporary variable to hold excep_id to construct excep_trigger
map
  the_model: type_declaration;
```

```
  the_operation_map: operation_map;
  the_data_structure: type_name;
  the_input: type_declaration;
  the_output: type_declaration;
  the_state: type_declaration;
  the_states_token_line: natural;
  the_states_token: text;
  the_initial_expression_map: init_map;
  the_exceptions: psdl_id_set;
  the_specified_met: millisec;
  the_graph: psdl_graph;
  the_streams: type_declaration;
  the_timers: psdl_id_set;
  the_trigger_map: trigger_map;
  the_exec_guard: exec_guard_map;
  the_out_guard: out_guard_map;
  the_excep_trigger: excep_trigger_map;
  the_timer_op: timer_op_map;
  the_per: timing_map;
  the_fw: timing_map;
  the_mcp: timing_map;
  the_mrt: timing_map;
  the_operator_id: op_id;
    -- is used for storing the operator id's in control constraints
part
  is_atomic_type: boolean;
    -- true if the psdl_component is an atomic type.
  is_atomic_operator: boolean;
    -- true if the psdl_component is an atomic operator.
  is_specification: boolean;
   -- True if the current unit is a psdl specification part.
  the_init_exp_seq: exp_seq;
    -- Holds the initial expressions for all state variables in an
operator spec.
  the_type_name: type_name;
  the_type_decl: type_declaration;
    -- Used to hold an inherited/synthesized attribtue pair.
  the_trigger: trigger;
  the_timer_op_record: timer_op;
  the_timer_op_set: timer_op_set;
  the_impl_desc: text;
  the_attribute_type: attribute_type;

  the_gen_par_rb: spec_req_map;
  the_input_rb: spec_req_map;
  the_output_rb: spec_req_map;
  the_state_rb: spec_req_map;
  the_exception_rb: spec_req_map;
  the_spec_met_rb: psdl_id_sequence;

  the_eg_rb: cc_req_map;
  the_per_rb: cc_req_map;
  the_fw_rb: cc_req_map;
  the_mcp_rb: cc_req_map;
```

```
the_mrt_rb: cc_req_map;

the_o_srbm: spec_req_map;
the_e_srbm: spec_req_map;
the_reset_srbm: spec_req_map;
the_start_srbm: spec_req_map;
the_stop_srbm: spec_req_map;

the_o_rb: cc_req_map_map;
the_e_rb: cc_req_map_map;
the_reset_rb: cc_req_map_map;
the_start_rb: cc_req_map_map;
the_stop_rb: cc_req_map_map;

has_syntax_error: boolean := false;

-----------------------------------------------------------------
-- procedure initialize_state_variables
-----------------------------------------------------------------
procedure initialize_state_variables is
begin
  yyval := (token_category => no_value_cat);
end initialize_state_variables;

-----------------------------------------------------------------
-- procedure yyparse
-----------------------------------------------------------------
procedure yyparse; -- Body is automatically generated.

-----------------------------------------------------------------
-- procedure yyerror
-----------------------------------------------------------------
procedure yyerror(s: in string := "syntax error";
                  err_line: natural := current_line;
                  err_token: text := convert(yytext)) is
  space: integer;
begin
  new_line;
  put_line(standard_error,
           "line" & integer'image(err_line) & ": "
           & convert(err_token));
  space := integer'image(err_line)'length +
           integer(convert(err_token)'length) + 5;
  for i in 1 .. space loop put(standard_error, "-"); end loop;
  put_line(standard_error, "^ " & s);
  has_syntax_error := true;
end yyerror;

-----------------------------------------------------------------
-- given a string of characters corresponding to a natural number,
-- returns the natural value
-----------------------------------------------------------------
function convert_to_natural(string_digit: string) return natural is
  digit, value : natural := 0;
```

```
  begin
    for i in 1 ..  string_digit'length loop
      case string_digit(i) is
        when '0' => digit := 0;
        when '1' => digit := 1;
        when '2' => digit := 2;
        when '3' => digit := 3;
        when '4' => digit := 4;
        when '5' => digit := 5;
        when '6' => digit := 6;
        when '7' => digit := 7;
        when '8' => digit := 8;
        when '9' => digit := 9;
        when others => return value;
      end case;
      value := (10 * value) + digit;
    end loop;
    return value;
  end convert_to_natural;

  ------------------------------------------------------------------
  --                        procedure get
  --
  --  reads the standard input, parses it, and creates the
  --  psdl adt.
  ------------------------------------------------------------------
procedure get(item: in out psdl_program) is
begin
  initialize_state_variables;
  initialize_yylex;
  yyparse;
  if has_syntax_error then
     assign(item, empty_psdl_program);
     raise syntax_error;
  end if;
  assign(item, the_program);
end get;

  ------------------------------------------------------------------
  --                        procedure get

  --  reads the psdl source file from a file,
  --  parses it, and creates the psdl adt.
  ------------------------------------------------------------------
  procedure get(file: in file_type;
                item: in out psdl_program ) is
  begin
    set_input(file);
    get(item);
    set_input(standard_input);
  end get;

  ------------------------------------------------------------------
  --                     procedure bind_type_declaration
```

```
  --
  --bind each id in id the id
  --set to the type name
  --return temp_type_decl
  ------------------------------------------------------------------
  procedure bind_type_declaration(i_s: in psdl_id_sequence; tn: in
type_name;
                                  td: in out type_declaration) is
  begin
     for id : psdl_id in psdl_id_sequence_pkg.scan(i_s) loop
       bind(id, tn, td);
     end loop;
  end bind_type_declaration;

  ------------------------------------------------------------------
  --                     procedure bind_initial_state
  --
  -- Bind each id in the state map domain
  -- set to the type name initial expression
  ------------------------------------------------------------------
  procedure bind_initial_state(state: in type_declaration; init_seq: in
exp_seq;
                               . init_exp_map: in out init_map) is
    i: natural := 0;
  begin
-- Added by Dave Dampier 20 April 1994 to
-- eliminate use of the M4 Macros,
-- and adopt use of the new generator processor for loops.

    for id : psdl_id, td : type_name in
type_declaration_pkg.scan(state) loop
      i := i + 1;
      if i > expression_sequence_pkg.length(init_seq) then
         yyerror("semantic error - some states are not initialized.",
                  the_states_token_line, the_states_token);
         raise semantic_error;
      else bind(id,
                expression_sequence_pkg.fetch(init_seq, i),
                init_exp_map);
      end if;
    end loop;
-- End of Added Code.
-- Also eliminated old M4 code.

    if i < expression_sequence_pkg.length(init_seq) then
       yyerror("semantic error -
                there are more initializations than the states",
                the_states_token_line, the_states_token);
       raise semantic_error;
    end if;
  end bind_initial_state;

  ------------------------------------------------------------------
  --                     procedure bind_spec_rb
```

```
  -- Bind each id in the ids to the required-by sequence
  -- in the appropriate spec requirements map.
  ------------------------------------------------------------------
  procedure bind_spec_rb(a_t: attribute_type; ids, reqs :
psdl_id_sequence) is
  begin
    case a_t is
      when gen_par =>
        for id : psdl_id in psdl_id_sequence_pkg.scan(ids) loop
          bind(id, reqs, the_gen_par_rb);
        end loop;
      when input =>
        for id : psdl_id in psdl_id_sequence_pkg.scan(ids) loop
          bind(id, reqs, the_input_rb);
        end loop;
      when output =>
        for id : psdl_id in psdl_id_sequence_pkg.scan(ids) loop
          bind(id, reqs, the_output_rb);
        end loop;
      when state =>
        for id : psdl_id in psdl_id_sequence_pkg.scan(ids) loop
          bind(id, reqs, the_state_rb);
        end loop;
      when exc =>
        for id : psdl_id in psdl_id_sequence_pkg.scan(ids) loop
          bind(id, reqs, the_exception_rb);
        end loop;
      when met =>
        -- This case is different because the specified-met required-by
        -- applies to the entire operator.
        -- Normally there should be at most one specified met,
        -- but if there is more than one, the requirements traces are
        -- combined.
        for id : psdl_id in psdl_id_sequence_pkg.scan(reqs) loop
          if not member(id, the_spec_met_rb) then
             add(id, the_spec_met_rb);
          end if;
        end loop;
    end case;
  end bind_spec_rb;

  function remove_braces(t: text) return text is
    s: string := to_string(t);
    len: natural := length(t);
  begin
    return to_text(s(2 .. len-1));
  end remove_braces;

  -- Generated body of yyparse goes here.
##%procedure_parse
end parser;
--**********************
-- Parser Tokens
--**********************
```

```
with  Psdl_Concrete_Type_Pkg, Expression_Pkg, Psdl_Id_Pkg;
use   Psdl_Concrete_Type_Pkg, Expression_Pkg, Psdl_Id_Pkg;
package Parser_Tokens is


    type token_category_type is (integer_cat,
                                 text_cat,
                                 psdl_id_cat,
                                 psdl_id_sequence_cat,
                                 op_id_cat,
                                 operator_name_cat,
                                 opt_arg_cat,
                                 type_name_cat,
                            type_decl_cat,
                                 timer_op_id_cat,
                                 expression_cat,
                                 expression_seq_cat,
                                 property_map_cat,
                                 no_value_cat);

    type yystype (token_category: token_category_type := no_value_cat)
is
      record
         case token_category is
           -- lexical token attributes:
           when integer_cat =>
             integer_value: integer;
           when text_cat =>
             text_value: text;
           -- grammar psdl_id attributes:
           when psdl_id_cat =>
             psdl_id_value: psdl_id;
           when psdl_id_sequence_cat =>
             psdl_id_sequence_value: psdl_id_sequence;
           when op_id_cat =>
             op_id_value: op_id;
           when operator_name_cat =>
             type_name_part, op_name_part: psdl_id;
           when opt_arg_cat =>
             input_value, output_value: psdl_id_sequence;
           when type_name_cat =>
             type_name_value: type_name;
           when type_decl_cat =>
             type_decl_value: type_declaration;
          when timer_op_id_cat =>
             timer_op_id_value: timer_op_id;
           when expression_cat =>
             expression_value: expression;
           when expression_seq_cat =>
             expression_seq_value: expression_sequence;
           when property_map_cat =>
             property_map_value: init_map;
```

```
            when no_value_cat => null;
         end case;
      end record;

   YYLVal, YYVal : YYSType;
   type Token is
       (End_Of_Input, Error, '?', '{',
        '}', Illegal_Token, '(',
        ')', ',', '[',
        ']', ':', '.',
        '|', Arrow, True,
        False, Axioms_Token, By_All_Token,
        Req_By_Token, By_Some_Token, Call_Period_Token,
        Control_Token, Constraints_Token, Data_Token,
        Description_Token, Edge_Token, End_Token,
        Exceptions_Token, Exception_Token, Execution_Token,
        Finish_Token, Generic_Token, Graph_Token,
        Hours_Token, If_Token, Implementation_Token,
        Initially_Token, Input_Token, Keywords_Token,
        Maximum_Token, Minimum_Token, Microsec_Token,
        Min_Token, Ms_Token, Mod_Token,
        Not_Token, Operator_Token, Or_Token,
        Output_Token, Period_Token, Property_Token,
        Reset_Token, Response_Token, Sec_Token,
        Specification_Token, Start_Token, States_Token,
        Stop_Token, Stream_Token, Time_Token,
        Timer_Token, Triggered_Token, Type_Token,
        Vertex_Token, Within_Token, Identifier,
        Integer_Literal, Real_Literal, String_Literal,
        Text_Token, And_Token, Xor_Token,
        Logical_Operator, '<', '>',
        '=', Greater_Than_Or_Equal, Less_Than_Or_Equal,
        Inequality, Relational_Operator, '+',
        '-', '&', Binary_Adding_Operator,
        Unary_Adding_Operator, '*', '/',
        Rem_Token, Multiplying_Operator, Exp_Token,
        Abs_Token, Highest_Precedence_Operator );

   Syntax_Error : exception;

end Parser_Tokens;
```

```
--**********************
-- AWK script for Parser
--**********************


/procedure YY\_USER\_ACTION is/ {
  # keep this line and the "begin" that follows.
  print $0
  getline
  print $0
  # replace the body of the procedure with our code
  print "    increment_column_number(yytext'length);"
  # get the "null;" and discard it
  getline
  next
  }

  #
  # otherwise pass the line through
  #
  { print $0 }
```

APPENDIX C: SELECTED SOURCE CODE (PSDL EDITOR)

```
#  Makefile for PSDL Editor


SHELL = /usr/bin/csh

C_DIR = ./C_Code

INCLUDE_FLAGS = \
  -I. \
  -I$(PT_DIR) \
  -I$(PT_DIR)/GENERIC_TYPES \
  -I$(PT_DIR)/INSTANTIATIONS \
  -I$(PARSER_DIR)

LIBS = -lXm -lXt -lXext -lX11 -lm -lg++ -lgcc

PSDL_TYPE = \
  $(PT_DIR)/psdl_io.ali

GE_OBJECTS = \
  $(GE_DIR)/graph_editor.o \
  $(GE_DIR)/operator_object.o \
  $(GE_DIR)/stream_object.o \
  $(GE_DIR)/spline_object.o \
  $(GE_DIR)/graph_object_list.o \
  $(GE_DIR)/font_table.o \
  $(GE_DIR)/graph_object.o \
  $(GE_DIR)/setcursor.o \
  $(GE_DIR)/gettopshell.o \
  $(GE_DIR)/postpopup.o \
  $(GE_DIR)/build_option.o \
  $(GE_DIR)/timer_tool.o \
  $(GE_DIR)/action_area.o \
  $(GE_DIR)/warning.o \
  $(GE_DIR)/ge_utilities.o \
  $(GE_DIR)/stream_property_menu.o \
  $(GE_DIR)/operator_property_menu.o \
  $(GE_DIR)/windows.o \
  $(GE_DIR)/get_unique_id.o \
  $(GE_DIR)/ge_utilities_debug.o \
  $(GE_DIR)/report_errors.o

PARSERS = $(PARSER_DIR)/parser_pkg.ali

LOCAL_OBJECTS = \
  $(C_DIR)/main.o \
  $(C_DIR)/sde_structure.o \
  $(C_DIR)/sde_globals.o \
  $(C_DIR)/ge_support.o
```

```
GENERATED_ADA = \
  analysis_pkg.adb \
  at_list_pkg.adb \
  ge_utilities.adb \
  id_list_pkg.adb \
  io_utilities.adb \
  op_list_pkg.adb \
  psdl_utilities_pkg.adb \
  st_list_pkg.adb \


#  editor_io_pkg.adb \
#  editor_io_pkg-utilities.adb

SOURCES = \
  action_node_pkg.ads \
  action_node_pkg.adb \
  analysis_pkg.ads \
  analysis_pkg.adb \
  at_list_pkg.ads \
  at_list_pkg.adb \
  c_boolean_pkg.ads \
  c_boolean_pkg.adb \
  c_int_pkg.ads \
  c_string_pkg.ads \
  c_string_pkg.adb \
  c_unsigned_int_pkg.ads \
  duration_type_pkg.ads \
  error_msgs_pkg.ads \
  error_msgs_pkg.adb \
  ge_action_pkg.ads \
  ge_action_pkg.adb \
  ge_interface_pkg.ads \
  ge_op_id_pkg.ads \
  ge_op_id_pkg.adb \
  ge_operator_pkg.ads \
  ge_operator_pkg.adb \
  ge_trigger_type_pkg.ads \
  ge_trigger_type_pkg.adb \
  ge_utilities.ads \
  ge_utilities.adb \
  graph_desc_pkg.ads \
  graph_desc_pkg.adb \
  id_list_pkg.ads \
  id_list_pkg.adb \
  io_utilities.ads \
  io_utilities.adb \
  op_list_pkg.ads \
  op_list_pkg.adb \
  property_names_pkg.ads \
  property_names_pkg.adb \
  psdl_editor.ads \
  psdl_editor.adb \
  psdl_editor_pkg.ads \
```

```
  psdl_editor_pkg.adb \
  psdl_utilities_pkg.ads \
  psdl_utilities_pkg.adb \
  spline_ptr_pkg.ads \
  spline_ptr_pkg.adb \
  st_id_pkg.ads \
  st_list_pkg.ads \
  st_list_pkg.adb \
  stream_pkg.ads \
  stream_pkg.adb \
  time_trigger_type_pkg.ads \
  time_trigger_type_pkg.adb \
  time_unit_type_pkg.ads \
  time_unit_type_pkg.adb \
  unique_id_pkg.ads


SCRIPTS = \
  Makefile \
  gen.sed

.SUFFIXES:

all:
      (cd $(C_DIR) ; make all)
      (cd $(PARSER_DIR) ; make parsers)
      (cd $(GE_DIR) ; make ge)
      (cd $(PT_DIR) ; make gen parsers)
      make generated_sources
      make psdl_editor

psdl_editor::
      gnatmake -g -o psdl_editor $(INCLUDE_FLAGS) psdl_editor.adb \
               -bargs -n $(PARSERS) \
               -largs $(GE_OBJECTS) $(LOCAL_OBJECTS) $(LIBS)

generated_sources: $(GENERATED_ADA) $(PARSERS)

at_list_pkg.adb: at_list_pkg.g
      gen < at_list_pkg.g > tmp
      sed -f gen.sed tmp > at_list_pkg.adb
      /bin/rm tmp

id_list_pkg.adb: id_list_pkg.g
      gen < id_list_pkg.g > id_list_pkg.adb

io_utilities.adb: io_utilities.g
      gen < io_utilities.g > io_utilities.adb

op_list_pkg.adb: op_list_pkg.g
      gen < op_list_pkg.g > op_list_pkg.adb

st_list_pkg.adb: st_list_pkg.g
      gen < st_list_pkg.g > st_list_pkg.adb
```

```
psdl_utilities_pkg.adb: psdl_utilities_pkg.g
      gen < psdl_utilities_pkg.g > psdl_utilities_pkg.adb

analysis_pkg.adb: analysis_pkg.g
      gen < analysis_pkg.g > analysis_pkg.adb

ge_utilities.adb: ge_utilities.g
      gen < ge_utilities.g > ge_utilities.adb

ci: generated_sources
      ci_files -tRCS/desc *.[Cgly] *.ad[sb] $(SCRIPTS)
      (cd C_Code ; make ci)

ci_export: generated_sources
      ci_files -tRCS/desc *.[Cgly] *.ad[sb] $(SCRIPTS)
      (cd C_Code ; make ci)
      sleep 1
      touch *.ali *.o

edit: all
      cp psdl_editor ../CAPS.RELEASE.1.2/bin
      (cd TEST ; psdl_editor test.psdl ;  psdl_editor autopilot.psdl)

clean:
      rm *.ali *.o

.IGNORE:
# ignore nonzero exit codes below.

xref:
      gnatf -e -f -x6 $(INCLUDE_FLAGS) *.ad*
```

```
------------------------------------------------------------
-- This is the main procedure of the psdl editor.
-- It is called from a C main program to enable
-- the loader to find all the code.
--
-- The problem is that gnatbind does not allow multiple
-- Ada root programs with an Ada main program.
-- Since we have Ada calling C with calls
-- back to Ada, and there are no direct links between
-- the Ada driver at the top and the Ada parsers at the
-- bottom, the binder needs multiple roots to enable
-- the loader to find all the Ada modules.
------------------------------------------------------------

procedure psdl_editor;
pragma Export (C, psdl_editor, "psdl_editor");
```

```
--*********************************************************************
-- This is the main program of the PSDL_EDITOR
--
-- Usage:
--     psdl_editor <psdl-input-filename>
--
-- Change:
--
--*********************************************************************

  with Ada.Command_Line;
  with Ada.Exceptions; use Ada.Exceptions;
  with Text_Io; use Text_Io;
  with psdl_editor_pkg; use psdl_editor_pkg;
procedure psdl_editor is
  PSDL_File: File_Type;
begin
  if Ada.Command_Line.Argument_Count = 1 then
     declare    -- right number of arguments, get the file name.
       filename: string  := Ada.Command_Line.Argument(1);
     begin
       if (filename'length >= 6) and then
          filename(filename'last-4 .. filename'last) = ".psdl"
       then -- File name is ok, open the file and edit it.
          begin
            Open(PSDL_File, In_File, filename);
          exception
            when name_error =>
              begin
                -- The file does not exist, try to create one.
                Create(PSDL_File, In_File, filename);
              exception
                when others =>
                  PUT_LINE(STANDARD_ERROR,
                           "psdl_editor: couldn't open or create "
                              & filename);
              end;
            when others => PUT_LINE(STANDARD_ERROR,
                                    "psdl_editor: couldn't open "
                                 & filename);
          end;

          -- Got a PSDL file, edit and update it under user control.

          edit_program(PSDL_File => PSDL_File,
                       prototype_name =>
                        filename(1 .. filename'last-5));

       else PUT_LINE(STANDARD_ERROR, "error: bad file name");
            PUT_LINE(STANDARD_ERROR,
                     "usage: psdl_editor prototype_name.psdl");
       end if;
     end;
  else
```

```
        PUT_LINE(STANDARD_ERROR, "psdl_editor:
                                  error, wrong number of arguments");
        PUT_LINE(STANDARD_ERROR, "usage:
                                  psdl_editor prototype_name.psdl");
-- begin debugging patch
        PUT_LINE(STANDARD_ERROR, "using file test.psdl");
      declare    -- right number of arguments, get the file name.
        filename: string  := "test.psdl";
      begin
        if (filename'length >= 6) and then
           filename(filename'last-4 .. filename'last) = ".psdl"
        then -- File name is ok, open the file and edit it.
           begin
             Open(PSDL_File, In_File, filename);
           exception
             when name_error =>
               begin
                 -- The file does not exist, try to create one.
                 Create(PSDL_File, In_File, filename);
               exception
                 when others =>
                   PUT_LINE(STANDARD_ERROR,
                            "psdl_editor: couldn't open or create "
                             & filename);
               end;
             when others => PUT_LINE(STANDARD_ERROR,
                                     "psdl_editor: couldn't open "
                                      & filename);
           end;

           -- Got a PSDL file, edit and update it under user control.

           edit_program(PSDL_File => PSDL_File,
                        prototype_name =>
                        filename(1.. filename'last-5));

        else PUT_LINE(STANDARD_ERROR, "error: bad file name");
             PUT_LINE(STANDARD_ERROR, "usage:
                           psdl_editor prototype_name.psdl");
        end if;
      end;
-- end debugging patch
  end if;
exception
  when the_exception: others =>
    PUT_LINE(STANDARD_ERROR, "Internal error: unexpected exception " &
                             Exception_Name(the_exception));
end psdl_editor;
```

```
--*****************************
-- GE_UTILITIES SPECIFICATION
--*****************************

  with psdl_program_pkg; use psdl_program_pkg;
  with psdl_component_pkg; use psdl_component_pkg;
  with GE_Operator_pkg; use GE_Operator_pkg;
  with st_list_pkg; use st_list_pkg;
  with op_list_pkg; use op_list_pkg;
  with psdl_graph_pkg; use psdl_graph_pkg;

package ge_utilities is

  procedure add_edges(streams: in St_List; ops: in Op_List; g: in out
psdl_graph);
    -- Includes adding edge properties.


  procedure modify_child(child: in GE_Operator; edges: in St_List;
                         prototype: in psdl_program;
                         edited_prototype: in out psdl_program);

  procedure modify_child_type_op(child: in GE_Operator;
                                 edges: in St_List;
                                 prototype: in psdl_program;
                                 edited_prototype:
                                          in out psdl_program);

  procedure modify_child_operator(child: in GE_Operator;
                                  edges: in St_List;
                                  prototype: in psdl_program;
                                  edited_prototype:
                                          in out psdl_program);

  procedure update_type_operation_names(current_op:
                                          in composite_operator;
                                        edited_op:
                                          in out composite_operator);
end ge_utilities;
```

```
--*****************************
-- GE_UTILITIES BODY
--*****************************

  with stream_pkg; use stream_pkg;
  with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
  with psdl_utilities_pkg; use psdl_utilities_pkg;
  with psdl_id_pkg; use psdl_id_pkg;
  with ada_id_pkg; use ada_id_pkg;
  with expression_pkg; use expression_pkg;
  with spec_req_map_pkg; use spec_req_map_pkg;
  with time_unit_type_pkg; use time_unit_type_pkg;
  with property_names_pkg; use property_names_pkg;
  with id_list_pkg; use id_list_pkg;
  with spline_ptr_pkg; use spline_ptr_pkg;
  with substitution_map_pkg; use substitution_map_pkg;
  with vertex_substitution_map_pkg; use vertex_substitution_map_pkg;

  with text_io; use text_io;
  with psdl_io; use psdl_io;

package body ge_utilities is
  not_found: exception;

  procedure add_edge(S: in STREAM; ops: in Op_List; g: in out
psdl_graph) is
    -- Includes adding edge properties.
    source_op, sink_op: GE_Operator;
    source, sink : op_id;
    str: psdl_id := to_psdl_id(label(S));
    lat : millisec;
  begin
    if not Is_Deleted(S) then
       -- find an op_id to the sending vertex
       source := find_op_id(from(S), ops);
       -- find an op_id to the receiving vertex
       sink := find_op_id(to(S), ops);

       -- create a new edge and assign properties
       lat := Latency(S);
       g := add_edge(source, sink, str, g, lat);
       set_property(source, sink, str, id_p,
              create_integer_literal(Integer(Id(S))), g);
       set_property(source, sink, str, label_font_p,
              create_integer_literal(Label_Font(S)), g);
       set_property(source, sink, str, label_x_offset_p,
              create_integer_literal(Label_X_Offset(S)), g);
       set_property(source, sink, str, label_y_offset_p,
              create_integer_literal(Label_Y_Offset(S)), g);
       set_property(source, sink, str, latency_font_p,
              create_integer_literal(Latency_Font(S)), g);
       set_property(source, sink, str, latency_unit_p,
              create_integer_literal(to_integer(Latency_Unit(S))), g);
       set_property(source, sink, str, latency_x_offset_p,
```

```
                    create_integer_literal(Latency_X_Offset(S)), g);
        set_property(source, sink, str, latency_y_offset_p,
                    create_integer_literal(Latency_Y_Offset(S)), g);
        set_property(source, sink, str, spline_p,
                    create_string_literal(to_text(Arc(S))), g);
      end if;
    end add_edge;

    procedure add_edges(streams: in St_List; ops: in Op_List; g: in out
psdl_graph) is
      -- Includes adding edge properties.
      L: St_List := streams;
      current_edge: STREAM;
    begin
      while not St_List_Is_Null(L) loop
        current_edge := ST(L);
        add_edge(current_edge, ops, g);
        L := Next(L);
      end loop;
    end add_edges;

    procedure modify_child(child: in GE_Operator; edges: in St_List;
                           prototype: in psdl_program;
                           edited_prototype: in out psdl_program) is
      -- Create operator specs for new children and
      -- update operator specs for modified children.
    begin
     if not Is_Deleted(child) then
        if is_type_operation(child) then
           modify_child_type_op(child, edges,
                                prototype, edited_prototype);
        else modify_child_operator(child, edges,
                                   prototype, edited_prototype);
        end if;
     end if;
    end modify_child;

    procedure modify_child_type_op(child: in GE_Operator; edges: in
St_List;
                                   prototype: in psdl_program;
                                   edited_prototype: in out psdl_program)
is
      -- Create operator specs for new children and
      -- update operator specs for modified children.
      child_vertex: op_id := vertex_id(child);
      child_name: psdl_id := base_name(child_vertex);
      child_type_name: psdl_id := child_vertex.type_name;
      child_type: data_type;
      child_type_ops: operation_map;
      child_op: operator;
      exc_list: psdl_id_set;
      exc_rb: spec_req_map;

    begin
```

```
    child_type := fetch(edited_prototype, child_type_name);
    if child_type = null_component then
       -- The type has not been defined yet, create one.
       child_type := make_atomic_type(child_name);
       add(child_type, edited_prototype);
    end if;
    child_type_ops := operations(child_type);
    if (Is_New(child) or Is_Modified(child)) or else
       not member(child_name, child_type_ops)
    then
      to_exceptions(to_string(Exception_List(child)), exc_list,
exc_rb);
      build_psdl_operator(
        c_name => child_name,
        c_a_name => to_ada_id(child_name),
        imp_lang => Impl_Lang(child),
        g_par => empty_type_declaration,    -- GE does not supply this
        gen_par_rb => spec_req_map_pkg.empty,  --GE doesn't supply this
        kwr => to_psdl_id_set(Keyword_List(child)),
        i_desc => Informal_Desc(child),
        f_desc => Formal_Desc(child),
        inp => inputs(Id(child), edges),
        otp => outputs(Id(child), edges),
        st => empty_type_declaration,
        i_exp_map => empty_init_map,
        excps => exc_list,
        s_met => MET(child),
        input_rb => spec_req_map_pkg.empty,  -- GE does not supply this
        output_rb => spec_req_map_pkg.empty,  -- GE doesn't supply this
        state_rb => spec_req_map_pkg.empty,  -- GE does not supply this
        excep_rb => exc_rb,
        smet_rb => MET_Reqmts(child),
        is_atomic => not Is_Composite(child),
        the_opr => child_op);
      bind_operation(child_name, child_op, child_type_ops);
      set_operations(child_type, child_type_ops);
    elsif Is_Modified(child) then
      child_op := fetch(child_type_ops, child_name);
      set_name(child_op, child_name);
      to_exceptions(to_string(Exception_List(child)),
                              exc_list, exc_rb);
      if not Is_Composite(child) then
         set_implementation_language(child_op, Impl_Lang(child));
      end if;
      set_keywords(child_op, to_psdl_id_set(Keyword_List(child)));
      set_informal_description(child_op, Informal_Desc(child));
      set_axioms(child_op, Formal_Desc(child));
      set_inputs(child_op, inputs(Id(child), edges));
      set_outputs(child_op, outputs(Id(child), edges));
      set_exceptions(child_op, exc_list);
      set_specified_met(child_op, MET(child));
      set_specified_met_reqs(child_op, MET_Reqmts(child));
      set_exception_reqs(child_op, exc_rb);
    else
```

```
      child_op := fetch(child_type_ops, child_name);
      set_inputs(child_op, inputs(Id(child), edges));
      set_outputs(child_op, outputs(Id(child), edges));
    end if;
  end modify_child_type_op;

  procedure modify_child_operator(child: in GE_Operator;
                                  edges: in St_List;
                                  prototype: in psdl_program;
                                  edited_prototype: in out
psdl_program) is
    -- Create operator specs for new children and
    -- update operator specs for modified children.
    child_name: psdl_id := name(child);
    child_op: operator;
    exc_list: psdl_id_set;
    exc_rb: spec_req_map;
  begin
    if Is_New(child) then
      to_exceptions(to_string(Exception_List(child)),
                    exc_list, exc_rb);
      build_psdl_operator(
        c_name => child_name,
        c_a_name => to_ada_id(child_name),
        imp_lang => Impl_Lang(child),
        g_par => empty_type_declaration,     -- GE does not supply this
        gen_par_rb => spec_req_map_pkg.empty,  --GE doesn't supply this
        kwr => to_psdl_id_set(Keyword_List(child)),
        i_desc => Informal_Desc(child),
        f_desc => Formal_Desc(child),
        inp => inputs(Id(child), edges),
        otp => outputs(Id(child), edges),
        st => empty_type_declaration,
        i_exp_map => empty_init_map,
        excps => exc_list,
        s_met => MET(child),
        input_rb => spec_req_map_pkg.empty,  -- GE does not supply this
        output_rb => spec_req_map_pkg.empty,  -- GE doesn't supply this
        state_rb => spec_req_map_pkg.empty,  -- GE does not supply this
        excep_rb => exc_rb,
        smet_rb => MET_Reqmts(child),
        is_atomic => not Is_Composite(child),
        the_opr => child_op);
      add(child_op, edited_prototype);
    elsif Is_Modified(child) then
      assign(child_op, find(suffix(child_name), prototype));
      -- Find the old operator based on the op_num,
      -- the name may have changed.
      -- Use assign to make a copy so recycle will be safe.
      set_name(child_op, child_name);
      to_exceptions(to_string(Exception_List(child)),
                    exc_list, exc_rb);
      if not Is_Composite(child) then
         set_implementation_language(child_op, Impl_Lang(child));
```

```
      end if;
      set_keywords(child_op, to_psdl_id_set(Keyword_List(child)));
      set_informal_description(child_op, Informal_Desc(child));
      set_axioms(child_op, Formal_Desc(child));
      set_inputs(child_op, inputs(Id(child), edges));
      set_outputs(child_op, outputs(Id(child), edges));
      set_exceptions(child_op, exc_list);
      set_specified_met(child_op, MET(child));
      set_specified_met_reqs(child_op, MET_Reqmts(child));
      set_exception_reqs(child_op, exc_rb);
      add(child_op, edited_prototype);
    else
      assign(child_op, find(suffix(child_name), prototype));
        -- Use assign to make a copy so recycle will be safe.
      set_name(child_op, child_name);
      set_inputs(child_op, inputs(Id(child), edges));
      set_outputs(child_op, outputs(Id(child), edges));
        -- The inputs and outputs are derived from the graph,
        -- so they could have changed
        -- even if the explicit attributes of the child did not.
      add(child_op, edited_prototype);
    end if;
  end modify_child_operator;

  function find_edge(edge_id: expression; cg: psdl_graph) return edge
is
    e_id: expression;
  begin
    for e: edge in edge_set_pkg.scan(edges(cg)) loop
        e_id := get_property(e.source, e.sink, e.stream_name,
                             id_p, cg);
        if eq(e_id, edge_id) then return e; end if;
    end loop;
    raise not_found;
  end find_edge;

  -- will not work if different arcs of the same
  -- stream are renamed in different ways.
  function make_renaming(cg, eg: psdl_graph) return substitution_map is
    edge_id: expression;
    original_edge: edge;
    result: substitution_map := empty;
  begin
    for e: edge in edge_set_pkg.scan(edges(eg)) loop
        edge_id := get_property(e.source, e.sink, e.stream_name,
                                id_p, eg);
        begin
          original_edge := find_edge(edge_id, cg);
        exception
          when not_found => original_edge := e;
        end;
        bind(original_edge.stream_name, e.stream_name, result);
    end loop;
    return result;
```

```
  end make_renaming;

  procedure update_type_operation_names(current_op: in
composite_operator;
                                        edited_op: in out
composite_operator) is
    cg: psdl_graph := graph(current_op);
    eg: psdl_graph := graph(edited_op);
    id_renaming: substitution_map := make_renaming(cg, eg);
    vertex_renaming: vertex_substitution_map := empty;
    new_name: op_id;
  begin
    for v: op_id in op_id_set_pkg.scan(vertices(eg)) loop
        if is_type_op(v) then
           new_name := transform_vertex(v, id_renaming);
           bind(v, new_name, vertex_renaming);
        end if;
    end loop;
    rename_vertices(edited_op, vertex_renaming);
  end update_type_operation_names;

end ge_utilities;
```

```
--**********************************
-- PSDL_UTILITIES_PKG SPECIFICATION
--**********************************

  with psdl_program_pkg; use psdl_program_pkg;
  with psdl_component_pkg; use psdl_component_pkg;
  with psdl_graph_pkg; use psdl_graph_pkg;
  with psdl_concrete_type_pkg; use psdl_concrete_type_pkg;
  with psdl_id_pkg; use psdl_id_pkg;
  with expression_pkg; use expression_pkg;
  with St_List_pkg; use St_List_pkg;
  with spec_req_map_pkg; use spec_req_map_pkg;
  with ge_op_id_pkg; use ge_op_id_pkg;
  with C_Boolean_pkg; use C_Boolean_pkg;
  with c_string_pkg; use c_string_pkg;

  -- The following are needed in the body
  -- but can't go there because of gen.
  with Ada.characters.handling; use Ada.characters.handling;
  with Ada.strings; use Ada.strings;
  with Ada.strings.fixed; use Ada.strings.fixed;
  with Ada.strings.maps; use Ada.strings.maps;
  with Ada.strings.maps.constants; use Ada.strings.maps.constants;

package PSDL_Utilities_Pkg is

  function get_is_terminator(op_name: psdl_id; parent: operator) return
boolean;
    -- returns true if the named operator is a terminator bubble
    -- in the graph of the parent operator.
    -- If there are several nodes with the given operator name,
    -- uses the properties of the first one it finds.

  function Extract_State_Ids(c : in psdl_component) return psdl_id_set;
    -- returns the set of psdl_ids which are names of state streams
declared in the spec
    -- of the operator


  function Extract_Implementation_Id(c: in psdl_component) return Text;
  -- returns the implementation id of the atomic operator,
  -- e.g. Ada, TAE, C, etc

  function Extract_Vertex_Ids(c: in psdl_component) return op_id_set;

  function Extract_Period_Reqmts_Ids(name : in op_id;
                                     c : in psdl_component)
    return psdl_id_set;

  function Extract_FW_Reqmts_Ids(name : in op_id;
                                 c : in psdl_component)
    return psdl_id_set;

  function Extract_MRT_Reqmts_Ids(name : in op_id;
```

```
                                    c : in psdl_component)
    return psdl_id_set;

  function Extract_MCP_Reqmts_Ids(name : in op_id;
                                    c : in psdl_component)
    return psdl_id_set;

  function Extract_Trigger_Reqmts_Ids(name : in op_id;
                                        c : in psdl_component)
    return psdl_id_set;

  function Extract_Edge_Set(c: in psdl_component) return edge_set;

  function Get_PSDL_O_GENERICS_Text(c : in psdl_component) return Text;
    -- the output string consists of all generic parameter declarations
    -- in the spec of the operator c

  function Get_Psdl_Types_Text(prog : in psdl_program) return Text;
    -- prog is a psdl_program that contains only user-defined types

  function Get_Op_Spec_Text(o : in operator) return Text;
    -- gets the specification of the operator as a text string.

  function Get_Expression_Text(e : in expression) return Text;
    -- white characters are not allowed in the output
    -- string except when they
    -- are blanks inside a string literal, i.e.
    -- between two matching quotes

  function Get_Psdl_Id_Sequence_Text(s : in psdl_id_sequence) return
Text;
    -- the output string consists of a sequence of
    -- non-white characters which
    -- correspons to a list of ids separated by commas

  function Get_Output_Guards_Text(name : in op_id; c : in
psdl_component) return text;
    -- the output string consists of all
    -- the output guards and requirements
    -- traces associated with the vertex "name"
    -- in the control constraint of c

  function Get_Exception_Triggers_Text(name : in op_id; c : in
psdl_component)
    return text;
    -- the output string consists of all the
    -- exception triggers and requirements
    -- traces associated with the vertex
    -- "name" in the control constraint of c

  function Get_Exception_List_Text(name : in op_id;
                                    c : in psdl_component)
    return text;
```

```
  function Get_Timer_Operations_Text(name : in op_id; c : in
psdl_component)
    return text;
    -- the output string consists of all the timer
    -- operations and requirements
    -- traces associated with the vertex "name"
    -- in the control constraint of c

  function Get_Type_Name_Text(name : in type_name) return text;
    -- the output string represents the type_name as
    -- a sequence of non-white characters

  procedure separate_types(prototype: in psdl_program;
                           root_name: in psdl_id;
                           types: in out psdl_program);
    -- Separates the operators from the types.

  procedure create_root_name(prototype_name: in string;
                             prototype: in out psdl_program;
                             root_name: out psdl_id);
    -- Produces the name of the root node if there is one,
    -- otherwise constructs one from the prototype name
    -- and creates a corresponding root node.

  procedure check_suffixes(prototype: in out psdl_program);
    -- Check for suffixes and generates them if not there.

  function to_psdl_id_set(s: psdl_id_sequence) return psdl_id_set;
    -- converts the sequence to a set.

  function to_type_name(s: c_string) return type_name;
    -- converts the string to a psdl type_name.

  function to_op_id(label: text) return op_id;
    -- converts the label and suffixes to a psdl op_id.

  function to_expression(s: c_string) return expression;
    -- converts the string to a psdl expression.

  function to_operator(spec: text) return operator;
    -- returns an atomic operator with the given name
    -- and psdl specification.

  procedure to_exceptions(s: in string; exc_list: out psdl_id_set;
                          exc_rb: out spec_req_map);

  procedure to_out_guard_map(s: in string; og: out out_guard_map;
                             ogrb: out spec_req_map);

  procedure to_exception_guard_map(s: string;
                                   eg: out excep_trigger_map;
                                   egrb: out spec_req_map);

  procedure to_timer_op_set(s: in string; timer_op: out timer_op_set;
```

```
                                resetrb, startrb, stoprb:
                                  in out spec_req_map);

  procedure add_output_guards(id : in op_id;
                                og : in out_guard_map;
                                o_guard: in out out_guard_map);

  procedure add_exception_guards(id : in op_id;
                                eg : in excep_trigger_map;
                                e_guard: in out excep_trigger_map);

end PSDL_Utilities_Pkg;
```

```
--*****************************************
-- PSDL_UTILITIES_PKG BODY
--*****************************************

  with raw_text_file_pkg; use raw_text_file_pkg;
  with text_io; use text_io;
  with io_utilities; use io_utilities;
  with psdl_io; use psdl_io;
  with expression_io; use expression_io;
  with type_name_io; use type_name_io;
  with op_id_io; use op_id_io;
  with output_guard_io; use output_guard_io;
  with exception_guard_io; use exception_guard_io;
  with exception_io; use exception_io;
  with timer_op_guard_io; use timer_op_guard_io;

  with Unique_Id_Pkg; use Unique_Id_Pkg;
  with property_names_pkg; use property_names_pkg;
  with cc_req_map_pkg; use cc_req_map_pkg;
  with substitution_map_pkg; use substitution_map_pkg;
  with vertex_substitution_map_pkg; use vertex_substitution_map_pkg;

package body PSDL_Utilities_Pkg is

  function get_is_terminator(op_name: psdl_id; parent: operator) return
boolean is
    -- returns true if the named operator is a terminator bubble
    -- in the graph of the parent operator.
    -- If there are several nodes with the given operator name,
    -- uses the properties of the first one it finds.
    g: psdl_graph;
  begin
    g := graph(parent);
    -- Find the graph vertex corresponding to the given operator name.
    for v: op_id in op_id_set_pkg.scan(vertices(g)) loop
        if eq(base_name(v), op_name) then -- found it.
           return eq(get_property(v, is_terminator_p, g),
           true_expression);
        end if;
    end loop;
    -- Should never get here.
    put_line(standard_error, "get_is_terminator: node name "
             & convert(op_name));
    put_line(standard_error, "  not found in the graph of "
             & convert(name(parent)));
    return false;
  exception
    when others =>
      put_line(standard_error,
               "get_is_terminator: unexpected exception");
      return false;
  end get_is_terminator;
```

```
  function Extract_Input_Ids(c : in psdl_component) return psdl_id_set
is
    -- returns the set of psdl_ids which are names of input streams to
the operator
    input_ids : psdl_id_set;
    td : type_declaration;
  begin
    td := inputs(c);
    assign(input_ids, map_domain(td));
    return input_ids;
  end Extract_Input_Ids;

  function Extract_Output_Ids(c : in psdl_component) return psdl_id_set
is
    -- returns the set of psdl_ids which are names of output
    -- streams to the operator
    output_ids : psdl_id_set;
    td : type_declaration;
  begin
    td := outputs(c);
    assign(output_ids, map_domain(td));
    return output_ids;
  end Extract_Output_Ids;

 function Extract_State_Ids(c : in psdl_component) return psdl_id_set
is
    -- returns the set of psdl_ids which are names of state
    -- streams declared in the spec
    -- of the operator
    state_ids :  psdl_id_set;
    td : type_declaration;
  begin
    td := states(c);
    assign(state_ids, map_domain(td));
    return state_ids;
  end Extract_State_Ids;

  function Extract_Input_Reqmts_Ids(name : in psdl_id;
                                    c : in psdl_component)
    return psdl_id_set is
    -- returns the set of reqmts associated with the name
    -- input stream of the operator
    td : psdl_id_sequence;
    input_reqmts_ids : psdl_id_set;
  begin
    td := input_reqs(c, name);
    return to_psdl_id_set(td);
  end Extract_Input_Reqmts_Ids;

  function Extract_Output_Reqmts_Ids(name : in psdl_id;
                                     c : in psdl_component)
    return psdl_id_set is
    --returns the set of reqmts associated with
    --the name output stream of the operator
```

```
    td : psdl_id_sequence;
  begin
    td := output_reqs(c, name);
    return to_psdl_id_set(td);
  end Extract_Output_Reqmts_Ids;

  function Extract_State_Reqmts_Ids(name : in psdl_id;
                                    c : in psdl_component)
    return psdl_id_set is
    -- returns the set of reqmts associated with the name
    -- state stream of the operator
    td : psdl_id_sequence;
  begin
    td := state_reqs(c, name);
    return to_psdl_id_set(td);
  end Extract_State_Reqmts_Ids;

  function Extract_Exception_Reqmts_Ids(name : in psdl_id;
                                        c : in psdl_component)
    return psdl_id_set is
    -- returns the set of reqmts associated with the name
    -- exception of the operator
    td : psdl_id_sequence;
  begin
    td := exception_reqs(c, name);
    return to_psdl_id_set(td);
  end  Extract_Exception_Reqmts_Ids;

  function Extract_Met_Reqmts_Ids(c : in psdl_component) return
psdl_id_set is
    td : psdl_id_sequence;
  begin
    td := specified_maximum_execution_time_reqs(c);
    return to_psdl_id_set(td);
  end Extract_Met_Reqmts_Ids;

  function Extract_Implementation_Id(c: in psdl_component) return Text
is
    -- returns the implementation id of the
    -- atomic operator, e.g. Ada, TAE, C, etc
  begin
    return to_text(implementation_language(c));
  end Extract_Implementation_Id;

  function Extract_Vertex_Ids(c: in psdl_component) return op_id_set is
  begin
    return vertices(graph(c));
  end Extract_Vertex_Ids;

  function Extract_Edge_Set(c: in psdl_component) return edge_set is
  begin
    return edges(graph(c));
  end Extract_Edge_Set;
```

```
  function Extract_Period_Reqmts_Ids(name : in op_id;
                                     c : in psdl_component)
    return psdl_id_set is
    period_reqmts_ids : psdl_id_sequence;
  begin
    period_reqmts_ids := fetch(period_reqs_map(c), name);
    return to_psdl_id_set(period_reqmts_ids);
  end Extract_Period_Reqmts_Ids;

  function Extract_FW_Reqmts_Ids(name : in op_id;
                                 c : in psdl_component)
    return psdl_id_set is
    fw_reqmts_ids : psdl_id_sequence;
  begin
    fw_reqmts_ids := fetch(finish_within_reqs_map(c), name);
    return to_psdl_id_set(fw_reqmts_ids);
  end Extract_FW_Reqmts_Ids;

  function Extract_MCP_Reqmts_Ids(name : in op_id;
                                  c : in psdl_component)
    return psdl_id_set is
    mcp_reqmts_ids : psdl_id_sequence;
  begin
    mcp_reqmts_ids := fetch(minimum_calling_period_reqs_map(c), name);
    return to_psdl_id_set(mcp_reqmts_ids);
  end Extract_MCP_Reqmts_Ids;

  function Extract_MRT_Reqmts_Ids(name : in op_id;
                                  c : in psdl_component)
    return psdl_id_set is
    mrt_reqmts_ids : psdl_id_sequence;
  begin
    mrt_reqmts_ids := fetch(maximum_response_time_reqs_map(c), name);
    return to_psdl_id_set(mrt_reqmts_ids);
  end Extract_MRT_Reqmts_Ids;

  function Extract_Trigger_Reqmts_Ids(name : in op_id;
                                      c : in psdl_component)
    return psdl_id_set is
    trigger_reqmts_ids : psdl_id_sequence;
  begin
    trigger_reqmts_ids := fetch(execution_guard_reqs_map(c), name);
    return to_psdl_id_set(trigger_reqmts_ids);
  end Extract_Trigger_Reqmts_Ids;

  function Get_PSDL_O_GENERICS_Text(c : in psdl_component) return Text
is
    -- the output string consists of all generic parameter declarations
    -- in the spec of the operator c
  begin
    -- not finished, probably not needed.
    return empty;
  end Get_PSDL_O_GENERICS_Text;
```

```
  function Get_Expression_Text(e : in expression) return Text is
    -- white characters are not allowed in the output
    -- string except when they
    -- are blanks inside a string literal, i.e.
    -- between two matching quotes
    out_f: text_file;
    in_f: raw_text_file;
    result: text;
  begin
    -- Write the expression to a temporary file
    temporary_text_file(out_f);
    Set_Output(out_f);
    put_expression(e);
    Set_Output(Standard_Output);
    Close(out_f);

    -- Read it in as a text string from the temporary file
    temporary_raw_text_file(in_f);
    result := get(in_f);
    raw_text_file_pkg.Delete(in_f);
    remove_last_char(result);
    return result;
  exception
    when others =>
      PUT_LINE(STANDARD_ERROR, "Get_Expression_Text: io error");
      return empty;
  end Get_Expression_Text;

  function Get_Psdl_Id_Sequence_Text(s : in psdl_id_sequence) return
Text is
    -- the output string consists of a
    -- sequence of non-white characters which
    -- correspons to a list of ids separated by commas
    out_f: text_file;
    in_f: raw_text_file;
    result: text;
  begin
    -- Write the id sequence to a temporary file
    temporary_text_file(out_f);
    Set_Output(out_f);
    put_id_seq(s);
    Set_Output(Standard_Output);
    Close(out_f);

    -- Read it in as a text string from the temporary file
    temporary_raw_text_file(in_f);
    result := get(in_f);
    remove_last_char(result);
    raw_text_file_pkg.Delete(in_f);
    return result;
```

```
exception
    when others =>
      PUT_LINE(STANDARD_ERROR, "error: bad file name");
    return result;
  end Get_Psdl_Id_Sequence_Text;

  function Get_Output_Guards_Text(name : in op_id; c : in
psdl_component) return text is
    -- the output string consists of
    -- all the output guards and requirements
    -- traces associated with the vertex
    -- "name" in the control constraint
    -- of c
    out_f: text_file;
    in_f: raw_text_file;
    result: text;
  begin
    -- Write the id sequence to a temporary file
    temporary_text_file(out_f);
    Set_Output(out_f);
    put_output_guard(name, c);
    Set_Output(Standard_Output);
    Close(out_f);

    -- Read it in as a text string from the temporary file
    temporary_raw_text_file(in_f);
    result := get(in_f);
    remove_last_char(result);
    raw_text_file_pkg.Delete(in_f);
    return result;
  exception
    when others =>
      PUT_LINE(STANDARD_ERROR, "error: bad file name");
    return result;
  end Get_Output_Guards_Text;

  function Get_Exception_Triggers_Text(name : in op_id; c : in
psdl_component) return text is
    -- the output string consists of all the
    -- exception triggers and requirements
    -- traces associated with the vertex
    --"name" in the control constraint
    -- of c
    out_f: text_file;
    in_f: raw_text_file;
    result: text;
  begin
    -- Write the id sequence to a temporary file
    temporary_text_file(out_f);
    Set_Output(out_f);
    put_excep_trigger(name, c);
    Set_Output(Standard_Output);
    Close(out_f);
```

```
    -- Read it in as a text string from the temporary file
    temporary_raw_text_file(in_f);
    result := get(in_f);
    remove_last_char(result);
    raw_text_file_pkg.Delete(in_f);
    return result;
  exception
    when others =>
      PUT_LINE(STANDARD_ERROR, "error: bad file name");
    return result;
  end Get_Exception_Triggers_Text;

  function Get_Exception_List_Text(name : in op_id; c : in
psdl_component) return text is
    -- the output string consists of all the
    -- exception triggers and requirements
    -- traces associated with the vertex
    -- "name" in the control constraint
    -- of c
    out_f: text_file;
    in_f: raw_text_file;
    result: text;
  begin
    -- Write the id sequence to a temporary file
    temporary_text_file(out_f);
    Set_Output(out_f);
    put_id_set(exceptions(c), "EXCEPTIONS", exception_reqs_map(c));
    Set_Output(Standard_Output);
    Close(out_f);

    -- Read it in as a text string from the temporary file
    temporary_raw_text_file(in_f);
    result := get(in_f);
    raw_text_file_pkg.Delete(in_f);
    remove_last_char(result);
    return result;
  exception
    when others =>
      PUT_LINE(STANDARD_ERROR, "error: bad file name");
    return result;
  end Get_Exception_List_Text;

  function Get_Timer_Operations_Text(name : in op_id;
                                     c : in psdl_component)
    return text is
    -- the output string consists of all the
    -- timer operations and requirements
    -- traces associated with the vertex "name"
    -- in the control constraint
    -- of c
    out_f: text_file;
    in_f: raw_text_file;
    result: text;
```

```
begin
   -- Write the id sequence to a temporary file
   temporary_text_file(out_f);
   Set_Output(out_f);
   put_timer_op(name, c);
   Set_Output(Standard_Output);
   Close(out_f);

   -- Read it in as a text string from the temporary file
   temporary_raw_text_file(in_f);
   result := get(in_f);
   remove_last_char(result);
   raw_text_file_pkg.Delete(in_f);
   return result;
exception
   when others =>
      PUT_LINE(STANDARD_ERROR, "error: bad file name");
   return result;
end Get_Timer_Operations_Text;

function Get_Type_Name_Text(name : in type_name) return text is
   -- the output string represents the type_name
   -- as a sequence of non-white characters
   out_f: text_file;
   in_f: raw_text_file;
   result: text;
begin
   -- Write the id sequence to a temporary file
   temporary_text_file(out_f);
   Set_Output(out_f);
   put_type_name(name);
   Set_Output(Standard_Output);
   Close(out_f);

   -- Read it in as a text string from the temporary file
   temporary_raw_text_file(in_f);
   result := get(in_f);
   remove_last_char(result);
   raw_text_file_pkg.Delete(in_f);
   return result;
exception
   when others =>
      PUT_LINE(STANDARD_ERROR, "error: bad file name");
   return result;
end Get_Type_Name_Text;

function Get_Op_Spec_Text(o : in operator) return Text is
   -- gets the specification of the operator as a text string.
   out_f: text_file;
   in_f: raw_text_file;
   result: text;
```

```
  begin
    -- Write the id sequence to a temporary file
    temporary_text_file(out_f);
    Set_Output(out_f);
    put_component_spec(o);
    Set_Output(Standard_Output);
    Close(out_f);

    -- Read it in as a text string from the temporary file
    temporary_raw_text_file(in_f);
    result := get(in_f);
    remove_last_char(result);
    raw_text_file_pkg.Delete(in_f);
    return result;
  exception
    when others =>
      PUT_LINE(STANDARD_ERROR, "error: bad file name");
    return result;
  end Get_Op_Spec_Text;

  function Get_Psdl_Types_Text(prog : in psdl_program) return Text is
    -- prog is a psdl_program that contains only user-defined types
    out_f: text_file;
    in_f: raw_text_file;
    result: text;
  begin
    -- Write the id sequence to a temporary file
    temporary_text_file(out_f);
    Set_Output(out_f);
    put(out_f, prog);
    Set_Output(Standard_Output);
    Close(out_f);

    -- Read it in as a text string from the temporary file
    temporary_raw_text_file(in_f);
    result := get(in_f);
    remove_last_char(result);
    raw_text_file_pkg.Delete(in_f);
    return result;
  exception
    when others =>
      PUT_LINE(STANDARD_ERROR, "error: bad file name");
    return result;
  end Get_Psdl_Types_Text;

  procedure separate_types(prototype: in psdl_program; root_name: in
psdl_id;
                           types: in out psdl_program) is
    -- Separates the operators from the types.
    procedure bind(name: in psdl_id; module: in psdl_component;
program: in out psdl_program)
      renames psdl_program_map_pkg.bind;
      -- Make sure we use the internal bind operation.
      -- Means ops and types do not have valid parent pointers,
```

```
    -- and that get_definition will not
    -- work for data type operations.
begin
  for id : psdl_id, c : psdl_component in
      psdl_program_map_pkg.scan(prototype) loop
      if component_category(c) = psdl_type then
         bind(id, c, types);
      end if;
  end loop;
end separate_types;

procedure create_root_name(prototype_name: in string;
                           prototype: in out psdl_program;
                           root_name: out psdl_id) is
  -- Produces the name of the root node if there is one,
  -- otherwise constructs one from the prototype name
  -- and creates a corresponding root node.
  obsolete_roots: id_set := id_set_pkg.empty;
  root, op: operator;
  old_root_name: psdl_id;
begin
  root_name := find_root(prototype);
  begin -- check root name
    if prefix(root_name) /= prototype_name then
       root_name := convert(prototype_name, Get_Unique_Id);
       root := fetch(prototype, root_name);
       set_name(root, root_name);
    end if;
  exception
    when constraint_error =>  -- The root does not
                              -- have an op_num suffix.
       root_name := convert(prototype_name, Get_Unique_Id);
       root := fetch(prototype, root_name);
       set_name(root, root_name);
  end;
exception
  when no_root =>
    root_name := convert(prototype_name, Get_Unique_Id);
    root :=  make_composite_operator(root_name);
    add(root, prototype);
  when multiple_roots =>
    root_name := convert(prototype_name, Get_Unique_Id);
    old_root_name := to_psdl_id(prototype_name);
    root := fetch(prototype, old_root_name);
    if root = null_component then root :=
              make_composite_operator(root_name);
    else set_name(root, root_name);
    end if;
    for id : psdl_id, c : psdl_component in
                      psdl_program_map_pkg.scan(prototype) loop
        if component_category(c) = psdl_operator and
           parent(c) = null_component then
           id_set_pkg.add(id, obsolete_roots);
           op := c;
```

```
            -- recycle(op);
          end if;
        end loop;
        remove(obsolete_roots, prototype);
        add(root, prototype);
    end create_root_name;

    function make_vertex_renaming(g: psdl_graph;
                                  renaming: in substitution_map)
             return vertex_substitution_map is
      result: vertex_substitution_map := empty;
      op_num: ge_op_id;
      new_id: psdl_id;
      new_vertex: op_id;
    begin
      for child_vertex: op_id in op_id_set_pkg.scan(vertices(g)) loop
          if not eq(child_vertex, external) then
             if member(child_vertex.operation_name, renaming) then
                -- We have an old style vertex, the op_id operation name
                -- matches the psdl_id in the original definition.
                new_vertex := child_vertex;
                -- get the renamed operation name.
                new_id := fetch(renaming, child_vertex.operation_name);
                -- add the v_num.
                op_num := ge_op_id(Get_Unique_Id);
                new_id := to_psdl_id(to_text(new_id), op_num);
                -- install the transformed operation name.
                new_vertex.operation_name := new_id;
                bind(child_vertex, new_vertex, result);
             elsif
                 member(to_psdl_id(prefix(child_vertex.operation_name)),
                        renaming) then
                -- Somehow, we got a new-style graph
                -- with vertex numbers but
                -- an old-style operator definition without op_num's.
                new_vertex := child_vertex;
                -- get the renamed operation name.
                new_id := fetch(renaming,
to_psdl_id(prefix(child_vertex.operation_name)));
                -- add the v_num.
                op_num := suffix(child_vertex.operation_name);
                new_id := to_psdl_id(to_text(new_id), op_num);
                -- install the transformed operation name.
                new_vertex.operation_name := new_id;
                bind(child_vertex, new_vertex, result);
             end if;
          end if;
      end loop;
      return result;
    end make_vertex_renaming;

    function member(id: psdl_id; vertices: op_id_set) return boolean is
    begin
      for oid: op_id in op_id_set_pkg.scan(vertices) loop
```

```
        if eq(oid.operation_name, id) then return true; end if;
      end loop;
      return false;
    end member;

    procedure check_suffixes(prototype: in out psdl_program) is
      op_num: ge_op_id;
      old_id, new_id: psdl_id;
      renaming: substitution_map := empty;
      co: composite_operator;
      result: psdl_program := empty_psdl_program;
    begin
      -- Find the bad names, generate names with suffixes, construct a
renaming map,
      -- change the names of the components, and bind them into the
result map.
      for id : psdl_id, c : psdl_component in
psdl_program_map_pkg.scan(prototype) loop
          if component_category(c) = psdl_operator then
             begin
               old_id := name(c);
               op_num := suffix(old_id);
               -- If this works the suffix exists.
               if parent(c) /= null_component and then
                  member(old_id, vertices(graph(parent(c))))
               then
-- We have an old-style graph and definition, which happened
-- to have a numeric suffix. To prevent the original numeric suffix
-- from disappearing from sight, we add another invisble suffix.
                  loop
                    op_num := ge_op_id(Get_Unique_Id);
                    new_id := to_psdl_id(to_text(old_id), op_num);
                    exit when not member(new_id, result);
                  end loop;
                  set_name(c, new_id);
                  add(c, result);
                  bind(old_id, new_id, renaming);
               else add(c, result);
               end if;
             exception
               when constraint_error =>
                 -- The component does not have an op_num suffix.
                 -- We need to create and install a suffix.
                 loop
                   op_num := ge_op_id(Get_Unique_Id);
                   new_id := to_psdl_id(to_text(old_id), op_num);
                   exit when not member(new_id, result);
                 end loop;
                 set_name(c, new_id);
                 add(c, result);
                 bind(old_id, new_id, renaming);
             end;
          else add(c, result);
          end if;
```

```
    end loop;
    -- Now apply the renaming substitution to
    -- the graphs and control constriants
    -- of all the composite operators in the result map.
    for id : psdl_id, c : psdl_component in
        psdl_program_map_pkg.scan(result) loop
        if component_category(c) = psdl_operator and then
           component_granularity(c) = composite
        then co := c;
             rename_vertices(co, make_vertex_renaming(graph(co),
                             renaming));
        end if;
    end loop;
    assign(prototype, result);
  end check_suffixes;

  function to_psdl_id_set(s: psdl_id_sequence) return psdl_id_set is
    -- converts the sequence to a set.
    result: psdl_id_set := empty;
  begin
    for id: psdl_id in psdl_id_sequence_pkg.scan(s) loop
        add(id, result);
    end loop;
    return result;
  end to_psdl_id_set;

  function find_name(op_spec: text) return psdl_id is
    str: string := to_string(op_spec);
    len: natural := length(op_spec);
    id_char_set: character_set := Alphanumeric_set or To_Set('_');
    first, last: natural;
  begin
    Find_Token(str, id_char_set, inside, first, last);
    -- OPERATOR keyword
    Find_Token(str(last+1 .. len), id_char_set, inside, first, last);
    -- The name
    return to_psdl_id(head(str(first .. last), 1 + last - first));
  end find_name;

  function to_operator(spec: text) return operator is
    -- returns an atomic operator with the given name and psdl
specification.
    f: text_file;
    prog : psdl_program;
    name: psdl_id := find_name(spec);
  begin
    temporary_text_file(f);
    put(f, to_string(spec));
    new_line(f);
    put_line(f, "IMPLEMENTATION ADA " & to_string(name) & " END");
    reset(f, input);
    get(f, prog);
    delete(f);
    return fetch(prog, find_root(prog));
```

```
exception
  when others =>
    PUT_LINE(STANDARD_ERROR, "to_operator: I/O error");
    return make_atomic_operator(name);
end to_operator;

function to_type_name(s: c_string) return type_name is
  -- converts the string to a psdl type_name.
  f: text_file;
  result: type_name;
begin
  temporary_text_file(f);
  put(f, value(s));
  reset(f, input);
  get(f, result);
  delete(f);
  return result;
exception
  when others =>
    PUT_LINE(STANDARD_ERROR, "to_type_name: I/O error");
    return null_type;
end to_type_name;

function to_op_id(label: text) return op_id is
  -- converts the string to a psdl op_id.
  f: text_file;
  result: op_id;
begin
  -- convert the label to an op_id
  temporary_text_file(f);
  put(f, to_string(label));
  reset(f, input);
  get(f, result);
  delete(f);
  return result;
exception
  when others =>
    PUT_LINE(STANDARD_ERROR, "to_op_id: I/O error");
    return empty;
end to_op_id;

function is_blank(s: string) return boolean is
begin
  for i in s'range loop
      if not is_control(s(i)) then return false; end if;
  end loop;
  return true;
end is_blank;



function to_expression(s: c_string) return expression is
  -- converts the string to a psdl expression.
```

```
    f: text_file;
    result: expression;
    str: string := value(s);
  begin
    if is_blank(str) then return undefined_expression; end if;
    temporary_text_file(f);
    put(f, str);
    reset(f, input);
    get(f, result);
    delete(f);
    return result;
  exception
    when others =>
      PUT_LINE(STANDARD_ERROR, "to_expression: I/O error");
      return undefined_expression;
  end to_expression;

  procedure to_exceptions(s: in string; exc_list: out psdl_id_set;
                          exc_rb: out spec_req_map) is
    f: text_file;
  begin
    temporary_text_file(f);
    put(f, s);
    reset(f, input);
    get(f, exc_list, exc_rb);
    delete(f);
  exception
    when others =>
      PUT_LINE(STANDARD_ERROR, "to_exceptions: I/O error");
      exc_list := empty;
      exc_rb := empty;
  end to_exceptions;

  procedure to_out_guard_map(s: in string; og: out out_guard_map;
                             ogrb: out spec_req_map) is
    f: text_file;
  begin
    temporary_text_file(f);
    put(f, s);
    reset(f, input);
    get(f, og, ogrb);
    delete(f);
  exception
    when others =>
      PUT_LINE(STANDARD_ERROR, "to_out_guard_map: I/O error");
      og := empty_out_guard_map;
      ogrb := empty;
  end to_out_guard_map;

procedure to_exception_guard_map(s: string; eg: out excep_trigger_map;
                                 egrb: out spec_req_map) is
    f: text_file;
  begin
    temporary_text_file(f);
```

```
  put(f, s);
  reset(f, input);
  get(f, eg, egrb);
  delete(f);
exception
  when others =>
    PUT_LINE(STANDARD_ERROR, "to_exception_guard_map: I/O error");
    eg := empty_excep_trigger_map;
    egrb := empty;
end to_exception_guard_map;

procedure to_timer_op_set(s: in string; timer_op: out timer_op_set;
                          resetrb, startrb, stoprb:
                          in out spec_req_map) is
  f: text_file;
begin
  temporary_text_file(f);
  put(f, s);
  reset(f, input);
  get(f, timer_op, resetrb, startrb, stoprb);
  delete(f);
exception
  when others =>
    PUT_LINE(STANDARD_ERROR, "to_timer_op_set: I/O error");
    timer_op := empty;
    resetrb := empty;
    startrb := empty;
    stoprb := empty;
end to_timer_op_set;

procedure add_output_guards(id : in op_id;
                            og : in out_guard_map;
                            o_guard: in out out_guard_map) is
  tid: output_id;
begin
  for oid: output_id,
        e: expression in out_guard_map_pkg.scan(og) loop
    tid := oid;
      tid.op := id;
      bind(tid, e, o_guard);
  end loop;
end add_output_guards;

procedure add_exception_guards(id : in op_id;
                               eg : in excep_trigger_map;
                               e_guard: in out excep_trigger_map) is
  tid: excep_id;
begin
  for eid: excep_id,
        e: expression in excep_trigger_map_pkg.scan(eg) loop
      tid := eid;
      tid.op := id;
      bind(tid, e, e_guard);
  end loop;
```

```
  end add_exception_guards;

end PSDL_Utilities_Pkg;
```

APPENDIX D: PSDL GRAMMAR

```
-- $Header: $

%%
start_symbol
        : psdl
        ;

psdl
      : psdl component
        |
        ;

component
        : data_type
        | operator
        ;

data_type
        : TYPE_TOKEN IDENTIFIER
          type_spec
          type_impl
        ;

type_spec
        :
          SPECIFICATION_TOKEN optional_generic_param optional_type_decl
          op_spec_list functionality END_TOKEN
        ;

optional_generic_param
        : GENERIC_TOKEN
          list_of_type_decl
        |
        ;

optional_type_decl
        :
          list_of_type_decl
        |
        ;

op_spec_list
        : op_spec_list OPERATOR_TOKEN IDENTIFIER operator_spec
        |
        ;

operator
        : OPERATOR_TOKEN IDENTIFIER
          operator_spec
          operator_impl
        ;
```

```
operator_spec
        : SPECIFICATION_TOKEN
          interface
          functionality END_TOKEN
        ;

interface
        : interface attribute reqmts_trace
        |
        ;

attribute
        : GENERIC_TOKEN
          list_of_type_decl
        | INPUT_TOKEN
          list_of_type_decl
        | OUTPUT_TOKEN
          list_of_type_decl
        | STATES_TOKEN
          list_of_type_decl
          INITIALLY_TOKEN initial_expression_list
        | EXCEPTIONS_TOKEN id_list
        | MAXIMUM_TOKEN EXECUTION_TOKEN TIME_TOKEN time
        ;

-- Initialization of the_type_decl is done by the callers of this rule.
list_of_type_decl
        : list_of_type_decl ',' type_decl
        | type_decl
        ;

type_decl
        : id_list ':' type_name
        ;

type_name
        : IDENTIFIER
          '[' list_of_type_decl ']'
        | IDENTIFIER
        ;

id_list
        : id_list ',' IDENTIFIER
        | IDENTIFIER
        ;

reqmts_trace      -- Ignored in this version.
        : REQ_BY_TOKEN  id_list
        |
        ;


functionality
```

```
        : keywords informal_desc formal_desc
        ;

keywords
        : KEYWORDS_TOKEN id_list
        |
        ;

informal_desc
        : DESCRIPTION_TOKEN TEXT_TOKEN
        |
        ;

formal_desc
        : axioms_TOKEN TEXT_TOKEN
        |
        ;

type_impl
        : IMPLEMENTATION_TOKEN ADA_TOKEN IDENTIFIER END_TOKEN
        | IMPLEMENTATION_TOKEN type_name op_impl_list END_TOKEN
        ;

op_impl_list
        : op_impl_list OPERATOR_TOKEN IDENTIFIER operator_impl
        |
        ;

operator_impl
        : IMPLEMENTATION_TOKEN ADA_TOKEN IDENTIFIER END_TOKEN
        | IMPLEMENTATION_TOKEN psdl_impl END_TOKEN
        ;

psdl_impl
        : data_flow_diagram
          streams
          timers
          control_constraints
          informal_desc
        ;

data_flow_diagram
        :
          GRAPH_TOKEN vertex_list edge_list
        ;

                -- Time is the maximum execution time.
vertex_list
        : vertex_list VERTEX_TOKEN op_id optional_time graph_properties
        |
        ;

                -- Time is the latency.
```

```
edge_list
        : edge_list EDGE_TOKEN IDENTIFIER
          optional_time op_id ARROW op_id graph_properties
        |
        ;

graph_properties
        : graph_properties PROPERTY_TOKEN IDENTIFIER '=' expression
        |
        ;

op_id
        : operator_name opt_arg
        ;

operator_name
        : IDENTIFIER '.' IDENTIFIER
        | IDENTIFIER
        ;

opt_arg
        : '(' optional_id_list '|' optional_id_list ')'
        |
        ;

optional_id_list
        : id_list
        |
        ;

optional_time
        : ':' time
        |
        ;

streams
        : DATA_TOKEN STREAM_TOKEN
          list_of_type_decl
        |
        ;

------------------------------------------------------------
-- The order of id's is not important, so we use psdl_id_set
-- as the data structure to store the timers.
------------------------------------------------------------
timers
        : TIMER_TOKEN id_list
        |
        ;

control_constraints
        : CONTROL_TOKEN CONSTRAINTS_TOKEN
```

```
          constraints
        ;
constraints
        : constraints OPERATOR_TOKEN op_id
          opt_trigger opt_period opt_finish_within
          opt_mcp opt_mrt constraint_options
        | OPERATOR_TOKEN op_id
          opt_trigger opt_period opt_finish_within
          opt_mcp opt_mrt constraint_options
        ;

constraint_options
        : constraint_options OUTPUT_TOKEN
          id_list IF_TOKEN expression reqmts_trace
        | constraint_options EXCEPTION_TOKEN IDENTIFIER
          opt_if_predicate reqmts_trace
        | constraint_options timer_op IDENTIFIER
          opt_if_predicate reqmts_trace
        |
        ;

opt_trigger
        : TRIGGERED_TOKEN trigger opt_if_predicate reqmts_trace
        |
        ;

trigger
        : BY_ALL_TOKEN id_list
        | BY_SOME_TOKEN id_list
        |
        ;


opt_period
        : PERIOD_TOKEN time reqmts_trace
        |
        ;

opt_finish_within
        : FINISH_TOKEN WITHIN_TOKEN time reqmts_trace
        |
        ;

opt_mcp
        : MINIMUM_TOKEN CALL_PERIOD_TOKEN time reqmts_trace
        |
        ;

opt_mrt
        : max_resp_time time reqmts_trace
        |
        ;

max_resp_time
```

```
        : MAXIMUM_TOKEN RESPONSE_TOKEN TIME_TOKEN
        ;

timer_op
        : RESET_TOKEN
        | START_TOKEN
        | STOP_TOKEN
        ;

opt_if_predicate
        : IF_TOKEN expression
        |
        ;

        ------------------------------------------------------------------
        -- The expression sequence
        -- is used by procedure bind_initial_state together with
        -- the states map to construct the init_map.
        ------------------------------------------------------------------

initial_expression_list
        : initial_expression_list ',' initial_expression
        | initial_expression
        ;

        ------------------------------------------------------------------
        -- There is one and only one initial state(initial expression)
        -- for each state variable. This production returns one
        -- expression to the parent rule corresponding to one state.
        -- This is done by using the internal stack ($$ convention).
        ------------------------------------------------------------------

initial_expression
        : TRUE
        | FALSE
        | INTEGER_LITERAL
        | REAL_LITERAL
        | STRING_LITERAL
        | IDENTIFIER
        | type_name '.' IDENTIFIER
        | type_name '.' IDENTIFIER '(' initial_expression_list ')'
        | '(' initial_expression ')'
        | initial_expression log_op initial_expression
          %prec logical_operator
        | initial_expression rel_op initial_expression
          %prec relational_operator
        | '-' initial_expression
          %prec unary_adding_operator
        | '+' initial_expression
          %prec unary_adding_operator
        | initial_expression bin_add_op initial_expression
          %prec binary_adding_operator
        | initial_expression bin_mul_op initial_expression
          %prec multiplying_operator
```

```
        | initial_expression EXP_TOKEN initial_expression
          %prec highest_precedence_operator
        | NOT_TOKEN initial_expression
          %prec highest_precedence_operator
        | ABS_TOKEN initial_expression
          %prec highest_precedence_operator
        ;

log_op
        : AND_TOKEN
        | OR_TOKEN
        | XOR_TOKEN
        ;

rel_op
        : '<'
        | '>'
        | '='
        | GREATER_THAN_OR_EQUAL
        | LESS_THAN_OR_EQUAL
        | INEQUALITY
        ;

bin_add_op
        : '+'
        | '-'
        | '&'
        ;

bin_mul_op
        : '*'
        | '/'
        | MOD_TOKEN
        | REM_TOKEN
        ;


time
        : time_number MICROSEC_TOKEN
        | time_number MS_TOKEN
        | time_number SEC_TOKEN
        | time_number MIN_TOKEN
        | time_number HOURS_TOKEN
        ;

time_number
        : INTEGER_LITERAL
        ;


expression_list
        : expression_list ',' expression
        | expression
        ;
```

```
-------------------------------------------------------------------------
-- Expressions can appear in guards appearing in control constraints.
-- These guards can be associated with triggering conditions, or
-- conditional outputs, conditional exceptions, or conditional timer
-- operations. Similar to initial expression, except that tim e values
-- and references to timers and data streams are allowed.
-------------------------------------------------------------------------
expression
        : TRUE
        | FALSE
        | INTEGER_LITERAL
        | REAL_LITERAL
        | STRING_LITERAL
        | IDENTIFIER
         -- The only difference from the initial expression
        | time
        | type_name '.' IDENTIFIER
        | type_name '.' IDENTIFIER '(' expression_list ')'
        | '(' expression ')'
        | expression log_op expression   %prec logical_operator
        | expression rel_op expression %prec relational_operator
        | '-' expression               %prec unary_adding_operator
        | '+' expression                %prec unary_adding_operator
        | expression bin_add_op expression
          %prec binary_adding_operator
        | expression bin_mul_op expression
          %prec multiplying_operator
        | expression EXP_TOKEN expression
          %prec highest_precedence_operator
        | NOT_TOKEN expression
          %prec highest_precedence_operator
        | ABS_TOKEN expression
          %prec highest_precedence_operator
        ;


%%
```

APPENDIX E: TEST DATA

```
TYPE STACK
SPECIFICATION
    GENERIC
      types : private

    type_2 : public

    OPERATOR PUSH
    SPECIFICATION
        INPUT
          I : INTEGER
        INPUT
          S : STACK
        OUTPUT
          S : STACK
    END

    OPERATOR POP
    SPECIFICATION
        INPUT
          S : STACK
        OUTPUT
          I : INTEGER
        OUTPUT
          S : STACK
    END

    OPERATOR Empty
    SPECIFICATION
        OUTPUT
          dummy : STACK
    END
    KEYWORDS
      stack, adt
    DESCRIPTION
      {This is a generic stack adt}
    AXIOMS
      {push (s,x) = s::x}
END
IMPLEMENTATION ADA STACK

END

OPERATOR Compute_8
SPECIFICATION
    INPUT
      X1 : INTEGER
    INPUT
      X2 : INTEGER
    INPUT
```

```
      X3 : INTEGER
    OUTPUT
      X1 : INTEGER
    OUTPUT
      X2 : INTEGER
    OUTPUT
      X3 : INTEGER
    OUTPUT
      DC : INTEGER
    OUTPUT
      DX : FLOAT
    OUTPUT
      DY : BOOLEAN
    EXCEPTIONS
      E1
    EXCEPTIONS
      E2
    KEYWORDS
      software, bubbles
    DESCRIPTION
      {This is an atomic bubble.
       Line 2.
       Line 3. }
    AXIOMS
      {P = NP }
END
IMPLEMENTATION ADA Compute_8

END

OPERATOR Consumer_4
SPECIFICATION
    INPUT
      DB : INTEGER
    INPUT
      DC : INTEGER
    EXCEPTIONS
      E1
    EXCEPTIONS
      E2
    MAXIMUM EXECUTION TIME 0 MS
END
IMPLEMENTATION ADA Consumer_4

END

OPERATOR OBSOLETE_ROOT_0
SPECIFICATION
END
IMPLEMENTATION ADA OBSOLETE_ROOT_0

END

OPERATOR OP_A_135
```

```
SPECIFICATION
    INPUT
      DX : FLOAT
END
IMPLEMENTATION ADA OP_A_135

END

OPERATOR OP_B_136
SPECIFICATION
    INPUT
      DY : BOOLEAN
END
IMPLEMENTATION ADA OP_B_136

END

OPERATOR Process_Data_5
SPECIFICATION
    INPUT
      DA : Missing__Info
    OUTPUT
      DB : INTEGER
    OUTPUT
      DC : INTEGER
    OUTPUT
      DX : FLOAT
    OUTPUT
      DY : BOOLEAN
    STATES
      X2 : INTEGER
      INITIALLY
        2
    STATES
      X1 : INTEGER
      INITIALLY
        1
    STATES
      S : STACK
      INITIALLY
        EMPTY
    EXCEPTIONS
      E1
    EXCEPTIONS
      E2
    EXCEPTIONS
      E3
    MAXIMUM EXECUTION TIME 150 MS
    KEYWORDS
      Compute, Composite, Parent
    DESCRIPTION
      {This is an composite bubble. }
    AXIOMS
      {P != NP, believe it or not }
```

```
END
IMPLEMENTATION
  GRAPH
    VERTEX STACK.POP_14(S | DB, S) : 120 MS
      PROPERTY x = 456
      PROPERTY y = 110
      PROPERTY radius = 30
      PROPERTY color = 62
      PROPERTY label_font = 0
      PROPERTY label_x_offset =  - 25
      PROPERTY label_y_offset = 84
      PROPERTY met_font = 0
      PROPERTY met_x_offset = 35
      PROPERTY met_y_offset =  - 10
      PROPERTY is_terminator = FALSE

    VERTEX STACK.PUSH_13(DA, S | S)
      PROPERTY x = 210
      PROPERTY y = 110
      PROPERTY radius = 30
      PROPERTY color = 62
      PROPERTY label_font = 0
      PROPERTY label_x_offset =  - 43
      PROPERTY label_y_offset = 82
      PROPERTY met_font = 0
      PROPERTY met_x_offset = 244
      PROPERTY met_y_offset = 78
      PROPERTY is_terminator = FALSE

    VERTEX Compute_8_19
      PROPERTY x = 330
      PROPERTY y = 341
      PROPERTY radius = 30
      PROPERTY color = 62
      PROPERTY label_font = 0
      PROPERTY label_x_offset = 3
      PROPERTY label_y_offset = 34
      PROPERTY met_font = 0
      PROPERTY met_x_offset = 298
      PROPERTY met_y_offset = 205
      PROPERTY is_terminator = FALSE

    EDGE S
      STACK.POP_14(S | DB, S) ->
      STACK.PUSH_13(DA, S | S)
        PROPERTY id = 33
        PROPERTY label_font = 0
        PROPERTY label_x_offset =  - 4
        PROPERTY label_y_offset = 5
        PROPERTY latency_font = 0
        PROPERTY latency_x_offset = 318
        PROPERTY latency_y_offset = 182
        PROPERTY spline = "362 71 "
```

```
EDGE DB
  STACK.POP_14(S | DB, S) ->
  EXTERNAL
    PROPERTY id = 32
    PROPERTY label_font = 0
    PROPERTY label_x_offset = 3
    PROPERTY label_y_offset = 1
    PROPERTY latency_font = 0
    PROPERTY latency_x_offset = 292
    PROPERTY latency_y_offset = 243
    PROPERTY spline = "593 102 675 137 "

EDGE DA
  EXTERNAL ->
  STACK.PUSH_13(DA, S | S)
    PROPERTY id = 31
    PROPERTY label_font = 0
    PROPERTY label_x_offset =  - 19
    PROPERTY label_y_offset =  - 17
    PROPERTY latency_font = 0
    PROPERTY latency_x_offset = 131
    PROPERTY latency_y_offset = 123
    PROPERTY spline = "46 137 142 107 "

EDGE X1
  Compute_8_19 ->
  Compute_8_19
    PROPERTY id = 51
    PROPERTY label_font = 0
    PROPERTY label_x_offset = 15
    PROPERTY label_y_offset = 24
    PROPERTY latency_font = 0
    PROPERTY latency_x_offset = 124
    PROPERTY latency_y_offset = 249
    PROPERTY spline = "437 402 403 443 "

EDGE X2
  Compute_8_19 ->
  Compute_8_19
    PROPERTY id = 51
    PROPERTY label_font = 0
    PROPERTY label_x_offset =  - 29
    PROPERTY label_y_offset = 18
    PROPERTY latency_font = 0
    PROPERTY latency_x_offset = 224
    PROPERTY latency_y_offset = 149
    PROPERTY spline = "287 407 318 444 "

EDGE S
  STACK.PUSH_13(DA, S | S) ->
  STACK.POP_14(S | DB, S)
    PROPERTY id = 34
    PROPERTY label_font = 0
    PROPERTY label_x_offset =  - 1
```

```
    PROPERTY label_y_offset = 12
    PROPERTY latency_font = 0
    PROPERTY latency_x_offset = 154
    PROPERTY latency_y_offset = 196
    PROPERTY spline = "365 185 "

EDGE X3
  Compute_8_19 ->
  Compute_8_19
    PROPERTY id = 50
    PROPERTY label_font = 0
    PROPERTY label_x_offset =  - 10
    PROPERTY label_y_offset =  - 10
    PROPERTY latency_font = 0
    PROPERTY latency_x_offset = 124
    PROPERTY latency_y_offset = 149
    PROPERTY spline = "330 299 383 297 "

EDGE DC
  Compute_8_19 ->
  EXTERNAL
    PROPERTY id = 2909
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 0
    PROPERTY label_y_offset = 0
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "439 309 496 266 "

EDGE DX
  Compute_8_19 ->
  EXTERNAL
    PROPERTY id = 2910
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 0
    PROPERTY label_y_offset = 0
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "482 360 574 351 "

EDGE DY
  Compute_8_19 ->
  EXTERNAL
    PROPERTY id = 2911
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 0
    PROPERTY label_y_offset = 0
    PROPERTY latency_font = 2
    PROPERTY latency_x_offset = 0
    PROPERTY latency_y_offset = 15
    PROPERTY spline = "477 400 566 423 "
```

```
  DATA STREAM
    X3 : INTEGER
  CONTROL CONSTRAINTS

      OPERATOR STACK.POP_14(S | DB, S)
        MINIMUM CALLING PERIOD 100 MS
        MAXIMUM RESPONSE TIME 4500 MICROSEC
        OUTPUT
          S
          IF S /= STACK.EMPTY

        OUTPUT
          DB
          IF DB > 0
        EXCEPTION E3
          IF S = STACK.EMPTY
        EXCEPTION E4
          IF DB < 0
        START TIMER
           Timer1

        RESET TIMER
          Timer2

      OPERATOR STACK.PUSH_13(DA, S | S)
        OUTPUT
          S
          IF S /= STACK.EMPTY
        EXCEPTION E3
          IF S = STACK.EMPTY
        START TIMER
           Timer1

      OPERATOR Compute_8_19
END

OPERATOR Producer_1
SPECIFICATION
    GENERIC
      G1 : FLOAT
    OUTPUT
      DA : Missing__Info
    MAXIMUM EXECUTION TIME 0 MS
END
IMPLEMENTATION ADA Producer_1

END

OPERATOR objects_2
SPECIFICATION
END
IMPLEMENTATION
  GRAPH
    VERTEX Process_Data_5_20 : 150 MS
```

```
    PROPERTY x = 183
    PROPERTY y = 139
    PROPERTY radius = 30
    PROPERTY color = 62
    PROPERTY label_font = 0
    PROPERTY label_x_offset = 5
    PROPERTY label_y_offset = 44
    PROPERTY met_font = 0
    PROPERTY met_x_offset = 8
    PROPERTY met_y_offset =  - 12
    PROPERTY is_terminator = FALSE

  VERTEX Producer_1_12 : 0 MS
    PROPERTY x = 14
    PROPERTY y = 79
    PROPERTY radius = 30
    PROPERTY color = 62
    PROPERTY label_font = 0
    PROPERTY label_x_offset = 20
    PROPERTY label_y_offset = 39
    PROPERTY met_font = 0
    PROPERTY met_x_offset = 63
    PROPERTY met_y_offset = - 5
    PROPERTY is_terminator = TRUE

  VERTEX Consumer_4_11 : 0 MS
    PROPERTY x = 353
    PROPERTY y = 205
    PROPERTY radius = 30
    PROPERTY color = 62
    PROPERTY label_font = 0
    PROPERTY label_x_offset = 14
    PROPERTY label_y_offset = 33
    PROPERTY met_font = 0
    PROPERTY met_x_offset = 63
    PROPERTY met_y_offset =  - 5
    PROPERTY is_terminator = TRUE

  VERTEX OP_A_135_129
    PROPERTY x = 114
    PROPERTY y = 306
    PROPERTY radius = 30
    PROPERTY color = 62
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 28
    PROPERTY label_y_offset = 40
    PROPERTY met_font = 2
    PROPERTY met_x_offset = 144
    PROPERTY met_y_offset = 306
    PROPERTY is_terminator = FALSE

  VERTEX OP_B_136_131
    PROPERTY x = 427
    PROPERTY y = 77
```

```
    PROPERTY radius = 30
    PROPERTY color = 62
    PROPERTY label_font = 2
    PROPERTY label_x_offset = 20
    PROPERTY label_y_offset = 40
    PROPERTY met_font = 2
    PROPERTY met_x_offset = 457
    PROPERTY met_y_offset = 77
    PROPERTY is_terminator = FALSE

  EDGE DA
    Producer_1_12 ->
    Process_Data_5_20
      PROPERTY id = 30
      PROPERTY label_font = 0
      PROPERTY label_x_offset = 6
      PROPERTY label_y_offset =  - 8
      PROPERTY latency_font = 0
      PROPERTY latency_x_offset = 124
      PROPERTY latency_y_offset = 149
      PROPERTY spline = "143 140 "

  EDGE DB
    Process_Data_5_20 ->
    Consumer_4_11
      PROPERTY id = 40
      PROPERTY label_font = 0
      PROPERTY label_x_offset = 1
      PROPERTY label_y_offset =  - 6
      PROPERTY latency_font = 0
      PROPERTY latency_x_offset = 269
      PROPERTY latency_y_offset = 204
      PROPERTY spline = "297 199 "

  EDGE DC
    Process_Data_5_20 ->
    Consumer_4_11
      PROPERTY id = 42
      PROPERTY label_font = 0
      PROPERTY label_x_offset =  - 8
      PROPERTY label_y_offset =  - 11
      PROPERTY latency_font = 0
      PROPERTY latency_x_offset = 0
      PROPERTY latency_y_offset = 16
      PROPERTY spline = "302 358 "

  EDGE DX
    Process_Data_5_20 ->
    OP_A_135_129
      PROPERTY id = 130
      PROPERTY label_font = 2
      PROPERTY label_x_offset =  - 27
      PROPERTY label_y_offset =  - 6
      PROPERTY latency_font = 2
```

```
        PROPERTY latency_x_offset = 184
        PROPERTY latency_y_offset = 237
        PROPERTY spline = "169 250 "

    EDGE DY
      Process_Data_5_20 ->
      OP_B_136_131
        PROPERTY id = 134
        PROPERTY label_font = 2
        PROPERTY label_x_offset =  - 17
        PROPERTY label_y_offset =  - 15
        PROPERTY latency_font = 2
        PROPERTY latency_x_offset = 288
        PROPERTY latency_y_offset = 165
        PROPERTY spline = "335 138 "

  DATA STREAM
    DY : BOOLEAN,
    DX : FLOAT,
    DC : INTEGER,
    DB : INTEGER,
    DA : Missing__Info
  CONTROL CONSTRAINTS

      OPERATOR Process_Data_5_20

      OPERATOR Producer_1_12
        PERIOD 4000 MS
        FINISH WITHIN 100 MS

      OPERATOR Consumer_4_11
        TRIGGERED BY ALL
          DB
          IF DB > 0

      OPERATOR OP_A_135_129

      OPERATOR OP_B_136_131
END
```

LIST OF REFERENCES

1. San Jose Mercury News, Section C, *Tech Ticker*, September 6, 1997.
2. Anderson, Steven E., *Functional Specification for a Generic C3I Workstation*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 90.
3. CAPS Interface Manual, Software Engineering Group, Naval Postgraduate School, Monterey, California.
4. CAPS Executive Briefing, Software Engineering Group, Naval Postgraduate School, Monterey, California.
5. Berzins, V., Luqi, *Software Engineering with Abstractions,* Addison Wesley, 1991.
6. Aho, A., Kernighan, B., Weinberger, P., *The Awk Programming Language*, Addison Wesley, 1988.
7. Feldman, M., Koffman, E., *Ada95: Problem Solving and Program Design*, Addison Wesley, 1991.
8. Booch, G., Bryan, D., *Software Engineering with Ada*, Benjamin/Cummings, Redwood City, California, 1995.
9. Barnes, J., *Programming in Ada95*, Addison Wesley, 1996.
10. Luqi, " *Software Evolution Through Rapid Prototyping"*, IEEE Transactions on Software Engineering, May 1989.
11. Luqi, Berzins, V., Yeh, R., *"A Prototyping Language for Real-Time Software,"* IEEE Transactions on Software Engineering, October 1988.
12. San Jose Mercury News, Section E, *Can Silicon Graphics Evolve*, June 2, 1997.
13. San Jose Mercury News, Section A, *Help Wanted: Tech Grads*, February 2, 1997.
14. The New York Times, Cybertimes, *1 Teensy Little Bug, 1 Humongous Crash*, December 1, 1996.
15. Biggerstaff, T., *"Moore's Law: Change or Die!"*, IEEE Software, Vol 13, No. 1, January 1996.
16. San Jose Mercury News, Section A, *Radar System Failed in Crash, Software Glitch Prevented Warning*, August 11, 1997.
17. J.A. McDermid, ``Safety-critical software: a vignette", Software Engineering Journal, Vol. 8, No. 1, pp. 2-3, 1993

18. Chadwick, C., in personal conversation with the author, June 26, 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Director, Training and Education 1
   MCCDC, Code C46
   1019 Elliot Road
   Quantico, VA 22134-5027

4. Director, Marine Corps Research Center 2
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, VA 22134-5107

5. Director, Studies and Analysis Division 1
   MCCDC, Code C45
   3300 Russell Road
   Quantico, VA 22134-5130

6. Marine Corps Representative.....1
   Naval Postgraduate School
   Code 037, Bldg. 234, HA-220
   Monterey, CA 93940

7. Cmdr Michael J. Holden, Code CS/Hm.....1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

8. Marine Corps Tactical Systems Support Activity 1
Technical Advisory Branch
ATTN: Maj. J. C. Cummiskey
Box 555171
Camp Pendleton, CA 92055-5080

9. Dr. Luqi, Code CS/Lq 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943