1988

An Integrated Tool Environment for Embedded Real-Time Software

Galik, Daniel; Luqi

Naval Postgraduate School

D. Galik and Luqi, "An Integrated Tool Environment for Embedded Real-Time Software", Technical Report NPS 52-88-008, Computer Science Department, Naval Postgraduate School, 1988-.

https://hdl.handle.net/10945/65220

T 16

NPS52-88-008

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AN INTEGRATED TOOL ENVIRONMENT
FOR EMBEDDED REAL-TIME SOFTWARE

Daniel Galik

LuQi

April 1988

Prepared for:

National Science Foundation
Washington, DC 20550

This report was prepared for the Naval Postgraduate School. The work reported herein was supported in part by the National Science Foundation.

This report was prepared by:

[redacted]

LUQI
Associate Professor
of Computer Science

Reviewed by:

[redacted]

VINCENT Y. LUM
Chairman
Department of Computer Science

Released by:

[redacted]

JAMES M. FREMGEN
Acting Dean of Information
and Policy Science

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPS52-88-008	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School — 6b. OFFICE SYMBOL (If applicable) 52	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION National Science Foundation — 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Agreement No. CCR-8710737 dated 27 Jul 87
8c. ADDRESS (City, State, and ZIP Code) Washington, D.C. 20550	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
AN INTEGRATED TOOL ENVIRONMENT FOR EMBEDDED REAL-TIME SOFTWARE (U)

12. PERSONAL AUTHOR(S) GALIK, Daniel, LUQI

13a. TYPE OF REPORT	13b TIME COVERED FROM ____ TO ____	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
		April 1988	26

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Ada, hard real-time system, embedded system, rapid prototyping, design methodology, specification language, reusability

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
The goal of such an environment is the design of large real-time software systems. A key component of this automated prototyping system is a high level prototyping language called PSDL (Prototype System Description Language). The main goals of using the Computer Aided Prototyping System (CAPS) and PSDL are to construct a prototype with a high degree of module independence and to do so rapidly. CAPS will provide an automated environment with facilities for retrieving reusable Ada software components based on PSDL specifications. The PSDL language, its associated prototyping method, and the use of reusable components from a software base make highly automated software tools practical, resulting in an ability to effectively produce reliable and cost-efficient models of hard real-time software systems.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] [✓ SAME AS RPT.] ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL LUQI	22b. TELEPHONE (Include Area Code) (408)646-2735	22c. OFFICE SYMBOL 52Lq

AN INTEGRATED TOOL ENVIRONMENT FOR EMBEDDED REAL-TIME SOFTWARE

Daniel Galik, LCDR USN

Luqi

Computer Science Department
Naval Postgraduate School
Monterey, CA. 93943

ABSTRACT

The goal of such an environment is the design of large real-time software systems. A key component of this automated prototyping system is a high level prototyping language called PSDL (Prototype System Description Language). The main goals of using the Computer Aided Prototyping System (CAPS) and PSDL are to construct a prototype with a high degree of module independence and to do so rapidly. CAPS will provide an automated environment with facilities for retrieving reusable Ada software components based on PSDL specifications. The PSDL language, its associated prototyping method, and the use of reusable components from a software base make highly automated software tools practical, resulting in an ability to effectively produce reliable and cost-efficient models of hard real-time software systems.

KEYWORDS

Ada, hard real-time system, embedded system, rapid prototyping, design methodology, specification language, reusability

INTRODUCTION

With the demand for hard real-time and embedded computer systems increasing, it is becoming critical that new approaches be proposed for the development of large software systems. Typically, these large embedded systems have similar requirements for critical real-time control and high reliability. Software engineers and end-users would benefit from an automated methodology which allows validation of design specifications and functional requirements early in the development life cycle. A fast, efficient, easy-to-use tool would increase designer productivity and also allow the user to feel more confident that the final product is feasible. The Computer Aided Prototyping System (CAPS) is a currently conceptualized tool which provides these capabilities. It implements the rapid prototyping concept utilizing a high level prototyping language called Prototype System Description Language (PSDL) [Ref. 1]. This paper reviews some of the recent work which has been accomplished in implementing components of the CAPS system, and proposes that the use of PSDL and its associated automated environment is well-suited for the prototype modeling of hard real-time software systems.

The goal of rapid prototyping is to develop an executable model of the intended system. When utilized during the early stages of the development life cycle, rapid prototyping allows validation of the requirements, specifications, and initial design, before valuable time and effort are expended on implementation software. Rapid prototyping initially establishes an iterative process between the user and the designer to concurrently define specifications and requirements for the time critical aspects of the envisioned system. The designer then constructs a model or prototype of the system in a high level, prototyping language (PSDL). This prototype is a partial representation of the system, including only those critical attributes necessary for meeting user requirements, and is used as an aid in analysis and design rather than as production software [Ref. 2: pp. 2-5]. During demonstrations of the prototype, the user validates the prototype's actual performance against its expected performance. If the prototype fails to execute properly or to meet any critical timing constraints, the user identifies required modifications and redefines the critical specifications and requirements (see Fig. 1).

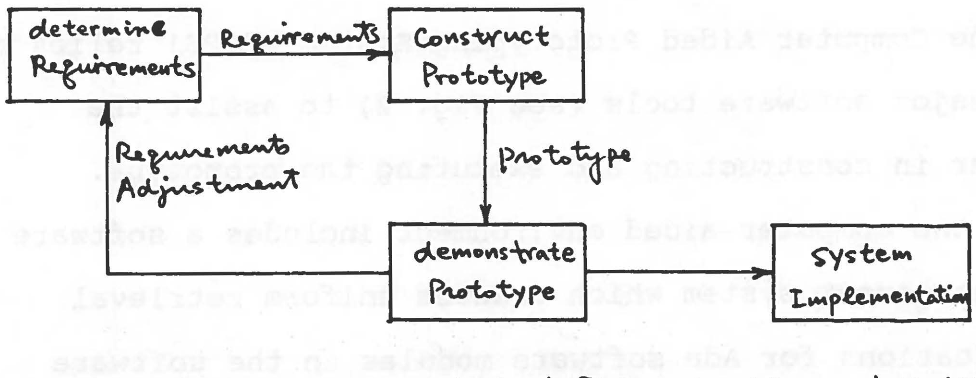

Fig. 1 Process of Regirements Determination and Validation

This process continues until the user and the designer both agree that the prototype successfully meets the time critical aspects of the envisioned system. A key aspect of the process is the feedback from the user to the designer. This iterative communication process should result in a model that should ultimately meet the intended requirements of the user. Following this final validation, the designer uses the prototype as a basis for the design and eventual hand coding of the production software. The design of large scale Ada software systems typically have particularly strict requirements on:

* accuracy
* safety
* reliability

These are difficult to meet without extensive prototyping. A rapid prototyping environment for creating and modifying an executable prototype is needed. The PSDL language, its associated prototyping method, and programming environment apply well to the modeling and design of hard real-time embedded Ada software systems.

The Computer Aided Prototyping System (CAPS) relies on three major software tools (see Fig. 2) to assist the designer in constructing and executing the prototype. First, the computer-aided environment includes a software base management system which creates uniform retrieval specifications for Ada software modules in the software database and later retrieves these reusable modules for assembling the executable prototype. Second, a graphics-capable user interface including a syntax-directed editor expedites the designers' data entry at a terminal and prevents syntax errors in the design. Finally, an execution support system demonstrates and measures the prototype's performance and analyzes the accuracy of design specifications [Ref. 11]. Rapid construction of the prototype relies on application of the rapid prototyping methodology along with a support environment which automates the steps involved.

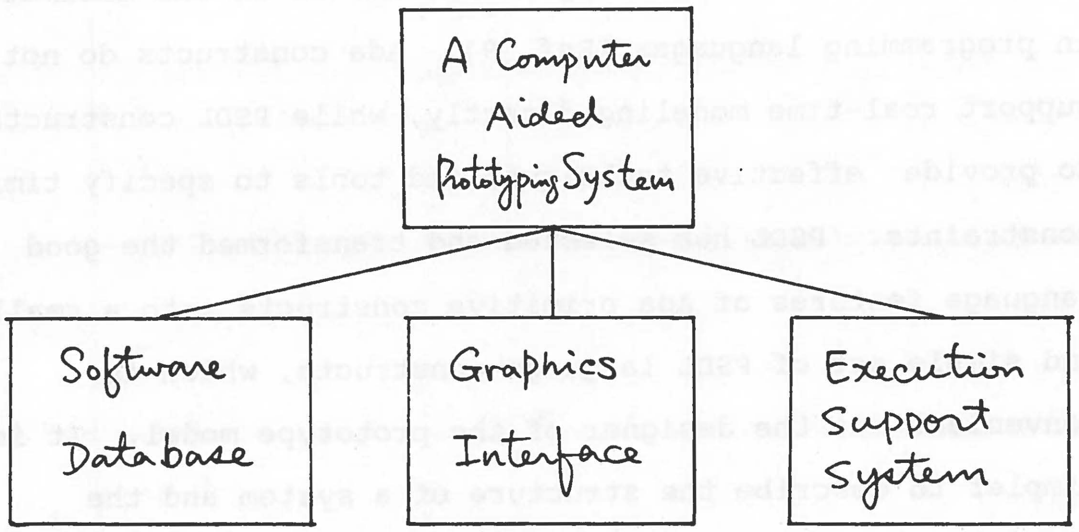

Figure 2. MAJOR SOFTWARE TOOLS

PROTOTYPE SYSTEM DESCRIPTION LANGUAGE

PSDL was designed as the primary connection between the designer and the components of the CAPS. By definition, PSDL is a high-level prototyping language with special features appropriate for defining critical real-time constraints, and is applied at the specification or design stage [Ref. 4: pp. 3, 23]. PSDL was specifically designed for the modeling of embedded and hard real-time software. PSDL and CAPS use as an implementation language the language Ada, which the Department of Defense has mandated as the required language to be used for the design of hard real-time embedded systems. Ada supports key software engineering principles including software portability and reuse. However, it is not perfect, nor is it the ultimate in programming languages [Ref. 9]. Ada constructs do not support real-time modeling directly, while PSDL constructs do provide effective techniques and tools to specify timing constraints. PSDL has selected and transformed the good language features of Ada primitive constructs into a small and simple set of PSDL language constructs, which is convenient for the designer of the prototype model. It is simpler to describe the structure of a system and the relation between system components in PSDL than in Ada since PSDL allows a designer to express his thoughts and ideas clearly, easily, and significantly faster at a specification or a design level with notations based on abstractions. The important points are that the software tools and the prototyping method of PSDL lead to a well structured prototype and that the resulting PSDL prototype is executable. PSDL components can be mapped into Ada directly. The methodology involves the use of specification and prototyping languages during the early design phase, and then use a language such as Ada as the implementation language for the final product [Ref. 11]. Ada is a large and powerful programming language. It is a good underlying programming or implementation language for PSDL. However, it is too hard and too cumbersome to use as a design language directly. The mapping between PSDL and Ada and the use of reuseable Ada components are the keys to making PSDL prototypes executable and useful in large Ada projects [Ref. 1].

In order to rapidly construct a prototype, PSDL aids the designer in systematically refining and decomposing each critical component into its lower level components. Uniform PSDL specifications associated with each lower level description act as templates for retrieving reusable Ada software components having similar specifications from the CAPS Software Database. Thus, the use of PSDL produces a computational model consisting of the basic building blocks needed to decribe the abstractions and concepts of the hierarchically structured prototype. The PSDL execution support environment then verifies the design and the validity of the prototype's real-time requirements. The actual execution of the prototype model demonstrates whether these critical timing constraints will perform in an acceptable manner that meets the timing constraints of the system as a whole [Ref. 1: pp. 2-7].

We will now present some of the technical details and major constructs of the PSDL language as developed and discussed by Luqi [Ref. 1]. PSDL supports the prototyping of large embedded real-time systems by providing a simple computational model that is close to the designer's view of the system. PSDL is based on a computational model containing OPERATORS that communicate via DATA STREAMS. Each data stream carries values of a fixed abstract type, and is a communication link connecting exactly two operators. The operators may be either data driven or periodic. Periodic operators have traditionally been the basis for most real-time system design, while the importance of data driven operators for real-time systems is beginning to be recognized [Ref. 10]. The PSDL computational model is based on enhanced data flow diagrams, which are directed graphs with associated timing and control constraints. The nodes of the graph are operators and the arcs are data flow paths, (see Fig. 3, where the numbers 10 and 20 above the bubbles A, B, C, are the maximum execution times given in the requirements of the real-time prototype).

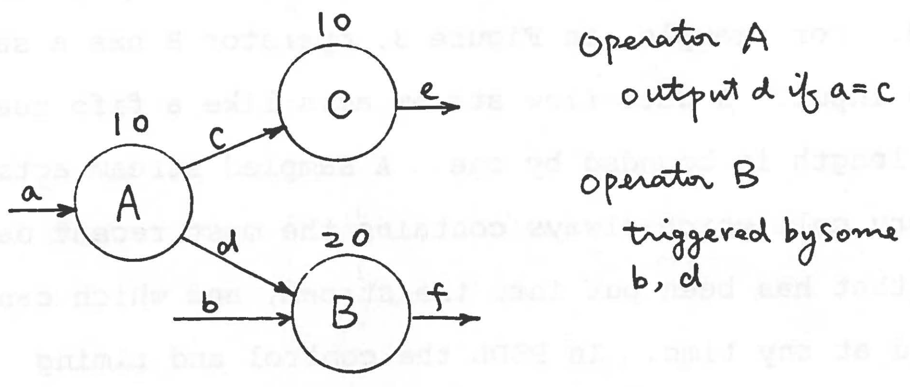

Fig. 3 operators & streams

Figure 3. OPERATORS AND STREAMS

Each operator is either a FUNCTION or a STATE MACHINE. When an operator fires, it reads one input value from each incoming arc, and puts at most one computed output value on each outgoing arc. The firing of an operator can be triggered either by the arrival of a specified set of input data values or by a periodic timing constraint. The firing of an operator and the production of an output value can also be subject to conditional control constraints that depend on locally available data values. This limited facility for interconnecting operators is well matched to the needs of real-time systems, in which each operator must complete its task within a fixed time limit [Ref. 11].

Each data stream is either a DATAFLOW STREAM, which guarantees each data element that enters is delivered once, or a SAMPLED STREAM, which guarantees a data element can always be entered into or delivered from the stream on demand. For example, in Figure 3, operator B has a sampled stream input. A data flow stream acts like a fifo queue whose length is bounded by one. A sampled stream acts like a memory cell which always contains the most recent data value that has been put into the stream, and which can be updated at any time. In PSDL the control and timing constraints of the operator receiving a stream determine whether the stream is of the dataflow or sampled variety, in a way that guarantees there will be data values present on all of the input streams of an operator whenever it fires [Ref. 12].

In PSDL each operator can have a maximum execution time and a maximum response time, which are treated as hard real-time constraints. Operators with real-time constraints are either periodic (synchronous) or sporadic (asynchronous). The firing frequency of each synchronous operator is specified by giving its period. The minimum period between firings is also specified for each sporadic operator, recording the necessary assumptions about worse case operating conditions with respect to asynchronous external events. The individual timing constraints of a real-time system are relatively easy to describe using the facilities described above. However, large real-time systems often contain a mixture of periodic and sporadic operators. The interactions between such timing constraints can be quite complex and very difficult to analyze without some type of computer aid [Ref. 11]

PSDL operators have two major parts: the SPECIFICATION and the IMPLEMENTATION. The specification part contains attributes describing the form of the interface, the timing characteristics, and both formal and informal descriptions of the observable behavior of the operator. The attributes both specify the operator and form the basis for retrievals from a reusable component library or software base. The implementation part determines whether the operator is atomic or composite. Atomic operators have a keyword specifying the underlying programming language (Ada in our application), followed by the name of the implementation module implementing the operator. This name is filled in as the result of a successful retrieval from the software base, or is supplied by the designer in case the module cannot be constructed from reuseable components and must be coded manually by the designer. Timing constraints are an essential part of specifying real-time systems. This is especially true for embedded, hard real-time systems which have hardware interface and synchronization factors to deal with. The most basic timing constraints are given in the specification part of a PSDL module, and consist of the MAXIMUM EXECUTION TIME, the MAXIMUM RESPONSE TIME, and the MINIMUM CALLING PERIOD. The maximum execution time is an upper bound on the length of time between the instant a module begins execution and the instant when it completes. The maximum response time for a sporadic operator is an upper bound on the time between arrival of a new data value and the time when the last value is put into the output streams of the operator in response to the arrival of the new data value. The minimum calling period is a constraint on the environment of a sporadic operator, consisting of a lower bound between the arrival of one set of inputs and the arrival of the next set. More complicated timing constraints can be given in the implementation part, using event controlled timers, triggering conditions, and output conditions.

Control of sporadic operators is signified by the PSDL token TRIGGERED BY. This token will be qualified by either the additional token ALL or SOME. TRIGGERED BY ALL indicates that an operator is to be fired when new data values have arrived on all the input streams to the operator. TRIGGERED BY SOME implies that the operator will be fired by the arrival of a new data value on any one, or possibly all, of the input streams to the operator. For example, in Figure 3, operator B is triggered by a sample stream. The new data value arriving in either "b" or "d" will trigger B. Another PSDL construct that is useful in the development of real-time systems is TIMER. It is used for such things as measuring the length of time between two events, or the length of time the system or an operator has remained in a particular state. For example, in Figure 4, T is defined as a TIMER (see below).

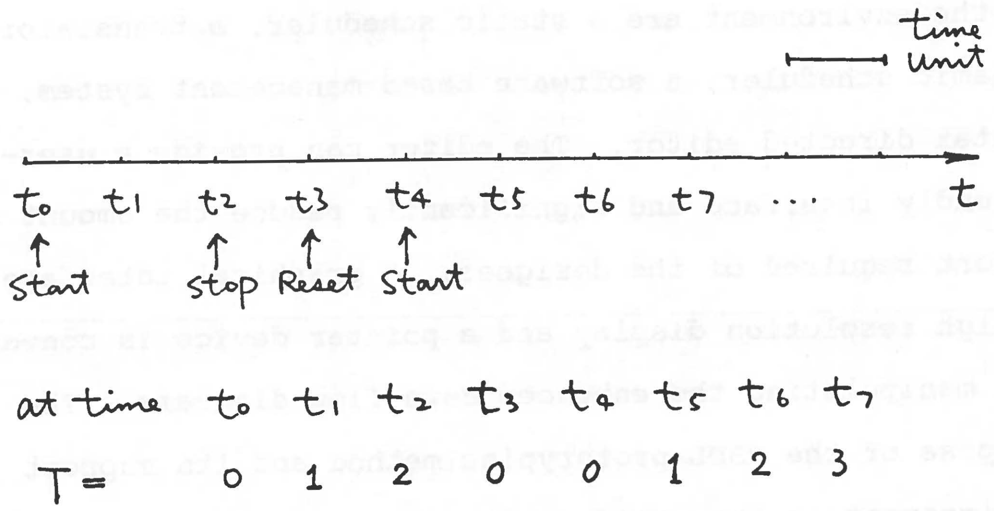
time unit
t0 t1 t2 t3 t4 t5 t6 t7 ... t
Start
stop
Reset
Start
at time t0 t1 t2 t3 t4 t5 t6 t7
T= 0 1 2 0 0 1 2 3

Figure 4. TIMER EXAMPLE

THE PSDL PROTOTYPING METHOD

The rapid construction of a prototype in PSDL is made possible by the associated prototyping method and support environment. The support environment reduces the efforts of the designer by automating some of the tasks involved in prototype construction. A PSDL prototype is constructed as a hierarchy of subsystems, referred to as the components of the prototype. Each component is either a reuseable module available in the software base or is defined in terms of the PSDL computational model. The code of the prototype usually cannot be used in the final implementation because the prototype is not a complete representation of the intended system. PSDL was designed to support an automated environment for rapid prototyping. The most important parts of the environment are a static scheduler, a translator, a dynamic scheduler, a software based management system, and a syntax directed editor. The editor can provide a user-friendly interface and significantly reduce the amount of effort required of the designer. A graphical interface with a high resolution display and a pointer device is convenient for manipulating the enhanced data flow diagrams. The purpose of the PSDL prototyping method and its support environment is the rapid construction of executable prototypes for large real-time systems. The PSDL prototyping method develops a hierarchically structured design by a perocess of stepwise refinement, guided by the computational model and the software base. At each level, the system at the center of attention is modeled as an enhanced data flow diagram (see Fig 5).

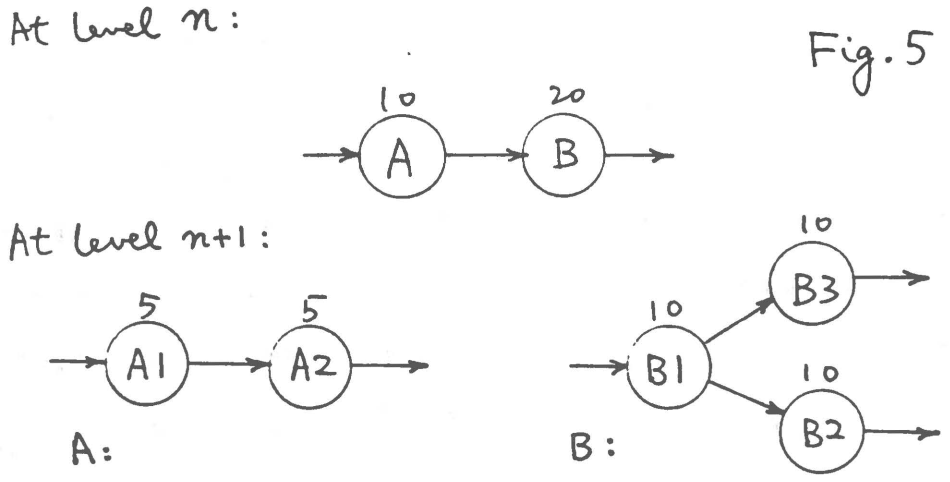

Fig. 5

While the model is created mostly in a top-down fashion, the process is guided by a tool for browsing through the reuseable components in the software base. Each of the operators and each of the data types associated with the data streams is subjected to further refinement [Ref. 1].

In the iterative rapid prototyping environment, the initial version of the prototype will meet some but not all of the requirements, and will provide an initial structure from which to work. A prototype is evaluated and tested through execution by the designer. Once the designer believes that the prototype meets all the requirements, the prototype is demonstrated for the user. Identified faults are reviewed along with the specifications, and the iterative process of refinement continues. The advantage of PSDL and its associated automated prototyping tool is that this iterative refinement process can proceed quickly and ultimately should result in a model of a more reliable system that the user agrees will meet the requirements. Hard real-time constraints of embedded systems can be more effectively modeled using PSDL.

The execution of PSDL prototype models is made possible through the use of an automated Execution Support System (ESS) which consists of three component parts, the Translator (TL), the Static Scheduler (SS), and the Dynamic Scheduler (DS). We will first examine the implementation of the Translator [Ref. 6]. Its basic function is to convert the rapid prototyping model PSDL source code that is input by the designer into Ada source code. Output from the TL is provided to the Ada compiler/linker along with some additional information from the SS to produce Ada object code. The object code is then exported to the operating system and can be run for test and demonstration purposes. The TL passes real time constraints through without translation. The TL creates code to implement the PSDL operators as procedures which will be called by the main subprogram/schedule created by the SS.

The effort required to produce a translator for CAPS was largely dependent on making use of a tool which makes possible the automatic generation of translators. That tool is known as the Kodiyak system. It is an Attribute Grammar based tool that was developed by Robert M. Herndon as a doctoral dissertation [Ref. 5]. To produce a translator with Kodiyak, the user must create a source file containing a listing of the terminal and non-terminal tokens of the source language to be translated. It also contains a listing of the valid attributes which each token may take on, as well as any precedence relationships which may be required to properly evaluate ambiguous cases in the grammar. Finally, the file contains a listing of attribute equations. These equations describe the relationship between the source language (in this case PDSL) and the target language (Ada). The Kodiyak translator generator system utilizes these equations to produce a translator in executable C code. The translator thus created is an executable program. By running this program with a text file in the source language as input, an output file is created which contains the equivalent code in the target language.

The Static Scheduler (SS) subsystem of the ESS alone represents the single most important component of CAPS as the basic requirement for computer-aided rapid prototyping of hard real-time systems [Ref. 7]. The SS specifically addresses only those PSDL component operators with critical timing constraints whose precise performance determines whether the system, as designed, will meet the required timing specifications. The primary purpose of the SS is creation of a static schedule which gives the precise execution order and timing of PSDL component operators with hard real-time constraints in such a manner that all timing constraints are guaranteed to be met [Ref. 2: p. 7]. Assuming that such a schedule is feasible given the system specifications, the static schedule contains the pre-allocated starting time and execution time for each critical operator. This structure implicitly denotes the precedence relationships between the operators. Without the benefit of a Static Scheduler, execution of the prototype would rely on basic control flow and processor scheduling as currently utilized in the majority of software systems. Rapid prototyping in general would benefit from CAPS without a static schedule. However, the SS provides CAPS with the unique capability required to realize increased gains in designer productivity and system reliability during development of hard real-time systems.

The third subsystem component of the ESS is the Dynamic Scheduler (DS). It operates at runtime along with the prototype model and is designed to control the execution of all non-critical operators within the program. A non-critical operator is one which is not subject to hard real-time constraints. The DS is invoked each time there is spare time within the static runtime schedule created by the SS. At that time the DS commences execution of the next available module in its set of operators and continues to invoke non-critical modules until the available time is exhausted. At that point, operation of the DS is interrupted and control is returned to the SS to continue the time critical operations. Future enhancements identified in addition to the current Dynamic Scheduler design would provide debugging capabilities and statistical information [Ref. 8].

CONCLUSION

This paper has presented a brief introduction and overview of PSDL and its associated automated prototyping tool, the Computer Aided Prototyping System. CAPS and the use of PSDL offer the potential of serving as a valuable software development tool specifically for hard real-time systems. The prototyping process is speeded up by automation and by reducing the conceptual burden of the analyst and prototype designer. PSDL was designed to interface to an automated support environment with a software base containing reusable software components. PSDL has constructs for recording timing constraints, for defining operator and data abstractions, and for specifying non-procedural control constraints. The most important aspects of real-time systems design are enforcing maximum response time and data synchronization constraints. PSDL handles both of these aspects well. The execution support system allows the designer to check the feasibility of a set of real time constraints by monitoring and evaluating the execution of the prototype model. PSDL offers a software tool that is a most practical way to support rapid prototyping of hard real-time software systems. This together with the features of PSDL for large scale software design make it a good candidate for inclusion in an advanced Ada programming environment. An experienced PSDL user should be able to construct a prototype significantly faster than an experienced Ada user [Ref. 1]. We can validate the hard real time system constraints by executing the constructed prototype. Further research efforts are in progress to implement all the subsystem components of the automated support environment of PSDL.

REFERENCES

1. Luqi. "Rapid Prototyping for Large Software System Design". Ph.D dissertation, University of Minnesota, 1987.

2. Luqi. "Execution of Real-Time Prototypes". ACM First International Workshop on Computer-Aided Software Engineering, Cambridge, Mass. May 1987, Vol 2, pp. 870-884; also Monterey: Naval Postgraduate School, 1987. Technical Report NPS52-87-012.

3. Berzins, V. and Luqi, "Rapid Prototyping of Real-Time Systems", to appear in IEEE Software 1988, also Monterey Naval Postgraduate School, 1987, Technical Report NPS52-87-005.

4. Luqi and Berzins, Valdis. "Handbook of Computer-Aided Software Engineering". "Languages for Specification, Design, and Prototyping". Van Nostrand Reinhold, 1988.

5. Herndon, Robert. "Automatic Construction of Language Translators". Ph.D. dissertation, University of Minnesota, 1988.

6. Moffitt, Charlie R. "A Language Translator for a Computer Aided Rapid Prototyping System", M.S. thesis, Naval Postgraduate School. Monterey, California. March 1988.

7. Janson, Dorothy M. "A Static Scheduler for Hard Real-Time Constraints in the Computer Aided Prototyping System (CAPS)", M.S. thesis, Naval Postgraduate School. Monterey, California. March 1988.

8. Eaton, Susan L. "A Dynamic Scheduler for the Computer Aided Prototyping System (CAPS)", M.S. thesis, Naval Postgraduate School, Monterey, California. March 1988.

9. Booch, Grady. "Software Engineering With Ada", Benjamin Cummings Publishing Company, 1983.

10. MacLaren, L. "Evolving Toward Ada in Real Time Systems", Proc. ACM SIGPLAN Notices Symp. on the Ada Programming Language, November 1980, pp. 146-155.

11. Berzins, V. and Luqi, "Software Engineering With Abstractions: An Integrated Approach to Software Development Using Ada". Addison-Wesley 1988.

12. Luqi, Berzins, V., and Yeh, R., "A Prototyping Language for Real-Time Systems", to appear in IEEE TSE, 1988. Also Technical Report 86-04, University of Minnesota, 1986.

Initial Distribution List

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Center for Naval Analysis 4401 Ford Avenue Alexandria, VA 22302-0268	1
Office of the Chief of Naval Operations Code OP-941 Washington, D.C. 20350	2
Office of the Chief of Naval Operations Code OP-945 Washington, D.C. 20340	2
Commander Naval Telecommunications Command Naval Telecommunications Command Headquarters 4401 Massachusetts Avenue NW Washington, D.C. 20390-5290	2
Commander Naval Data Automation Command Washington Navy Yard Washington, D.C. 20374-1662	1
Office of Naval Research Office of the Chief of Naval Research Attn. CDR Michael Gehl, Code 1224 Arlington, VA 22217-5000	1
Director, Naval Telecommunications System Integration Center NAVCOMMUNIT Washington Washington, D.C. 20363-5100	1
Space and Naval Warfare Systems Command Attn: Dr. Knudsen, Code PD50 Washington, D.C. 20363-5100	1

Ada Joint Program Office OUSDRE(R&AT) The Pentagon Washington, D.C. 230301	1
Naval Sea Systems Command Attn: CAPT Joel Crandall National Center #2, Suite 7N06 Washington, D.C. 22202	1
Office of the Secretary of Defense Attn: CDR Barber The Star Program Washington, D.C. 20301	1
Naval Ocean Systems Center Attn: Linwood Sutton, Code 423 San Diego, CA 92152-5000	1
Director of Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943	1
Chairman, Code 52 Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	1
LuQi Code 52Lq Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	150
National Science Foundation ATTN: Dr. Kent Curtis, Director of Computer and Computation Research 1800 G Street, NW Washington, DC 20550	1