Software Evolution Approach for the Development of Command and Control Systems [*]

Luqi, V. Berzins, M. Shing, N. Nada and C. Eagle
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943
{luqi, berzins, mantak, nnada, cseagle}@cs.nps.navy.mil

Abstract

This paper addresses the problem of how to produce reliable software that is also flexible and cost effective for the DoD distributed software domain. DoD software systems fall into two categories: information systems and war fighter systems. Both types of systems can be distributed, heterogeneous and network-based, consisting of a set of components running on different platforms and working together via multiple communication links and protocols. We propose to tackle the problem using prototyping and a "wrapper and glue" technology for interoperability and integration. This paper describes a distributed development environment, CAPS (Computer-Aided Prototyping System), to support rapid prototyping and automatic generation of wrapper and glue software based on designer specifications. The CAPS system uses a fifth-generation prototyping language to model the communication structure, timing constraints, I/O control, and data buffering that comprise the requirements for an embedded software system. The language supports the specification of hard real-time systems with reusable components from domain specific component libraries. CAPS has been used successfully as a research tool in prototyping large war-fighter control systems (e.g. the command-and-control station, cruise missile flight control system, missile defense systems) and demonstrated its capability to support the development of large complex embedded software.

1. Introduction

DoD software systems are currently categorized into Management Information Systems (MIS) and War Fighter/Embedded Real-time Systems. Both types of systems can be distributed, heterogeneous and network-based, consisting of a set of subsystems, running on different platforms that work together via multiple communication links and protocols. This paper addresses the problem of how to produce reliable software that is also flexible and cost effective for the DoD distributed software system domain, as depicted in the shaded area in Figure 1.

[*] This research was supported in part by the U. S. Army Research Office under contract/grant number 35037-MA and 40473-MA.

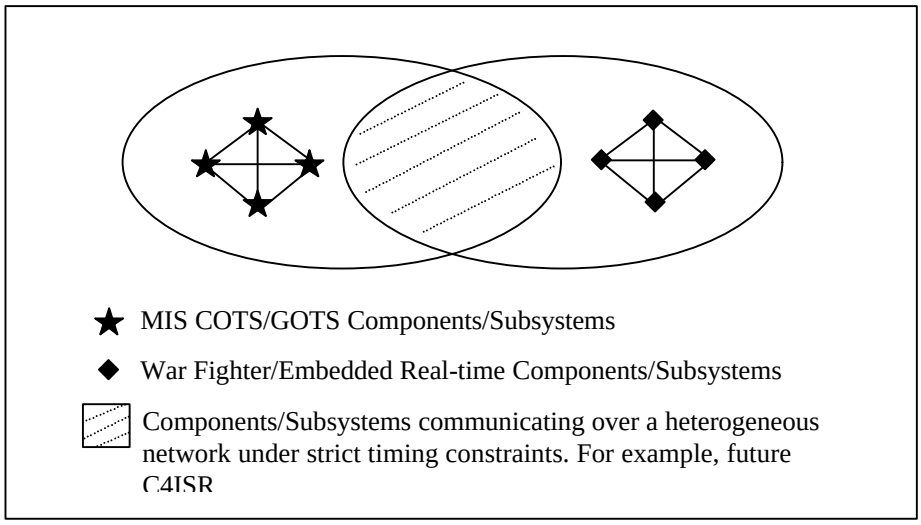


Figure 1. DoD Computer-based systems

Many DoD information systems are COTS/GOTS based (commercial/government off-the-shelf, including "legacy systems"). While using individual COTS/GOTS components saves DoD money, it shifts problems from software development to software integration and interoperability. It is a common belief that interoperability problems are caused by incompatible interface and data formats, and can be fixed "easily" using interface converters and data formatters. However, the real challenges in fixing interoperability problems are incompatible data interpretations, inconsistent assumptions, requirement extensions triggered by global integration issues, and timely data communication between components. Many DoD information systems, especially C4ISR systems, operate under tight timing constraints. Builders of COTS/GOTS based systems have no control over the network on which components communicate. They have to work with available infrastructure and need tools and methods to assist them in making correct design decisions to integrate COTS/GOTS components into a distributed network based system. Similar integration and interoperability problems are common in the commercial sector, and real-time issues are a growing concern. For example, just-in-time manufacturing, on-demand accounting, and factory automation all involve timing requirements. Although software engineers have more control over interfaces and data compatibility between individual components of war fighter systems, they encounter similar data communication problems when they need to connect these components via heterogeneous networks.

We can tackle the problem using prototyping and a "wrapper and glue" technology for interoperability and integration. Our approach is based on a distributed architecture where components collaborate via message passing over heterogeneous networks. It uses a generic interface that allows system designers to specify communication and operating requirements between components as parameters, based on properties of COTS/GOTS components. A separate parameterized model of network characteristics constrains the concrete "glue" software generated for each node. The model enables partial specification of requirements by the system designers, and allows them to explore design alternatives and determine missing parameters via rapid prototyping.

2. The Wrapper and Glue Approach

The cornerstone of our approach is automatic generation of wrapper and glue software based on designer specifications. This software bridges interoperability gaps between individual COTS/GOTS components. Wrapper software provides a common message-passing interface for components that frees developers from the error prone tasks of implementing interface and data conversion for individual components. The glue software schedules time-constrained actions and carries out the actual communication between components. (See Figure 2)

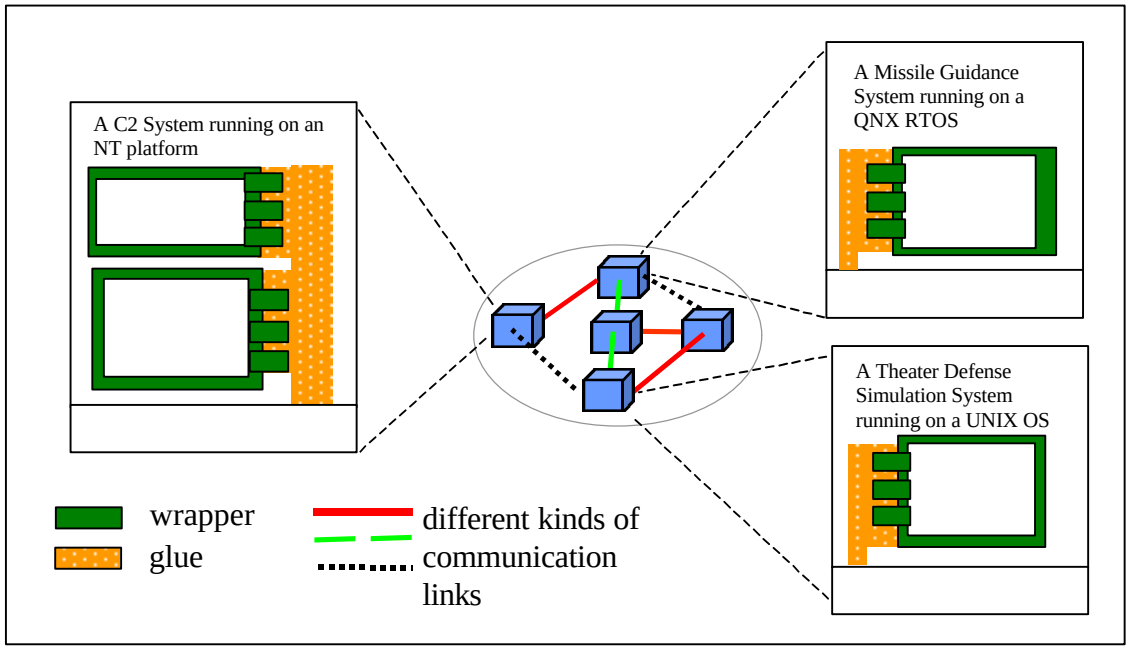


Figure 2. The wrapper and glue software

Our glue-and-wrapper approach uses rapid prototyping and automated software synthesis to improve reliability. It differs from proxy and broker patterns in the object-oriented design literature [4] in that it provides a formal model to support hardware/software co-design. Existing pattern approaches focus on low level data transfer issues. Our approach allows system designers to concentrate on the difficult interoperability problems and issues, while freeing them from implementation details. Prototyping with engineering decision support can help identify and resolve requirements conflicts and semantic incompatibilities.

Glue code works on two levels. It controls the orderly execution of components within a subsystem, and ensures the timely delivery of information between components across a network. Automated generation of glue code depends on automated local and distributed scheduling of actions on heterogeneous computing platforms. Identifying timing constraint conflicts and assessing constraint feasibility are critical in designing and constructing real-time software quickly. Checking whether a set of timing and task precedence constraints can be met on a chosen hardware configuration is known to be a difficult problem. Computer aid is needed in tackling such problem.

3. **The Computer Aided Prototyping System (CAPS)**

The value of computer aided prototyping in software development is clearly recognized. It is a very effective way to gain understanding of the requirements, reduce the complexity of the problem and provide an early validation of the system design. Bernstein estimated that for every dollar invested in prototyping, one can expect a $1.40 return within the life cycle of the system development [1]. To be effective, prototypes must be constructed and modified rapidly, accurately, and cheaply [8]. Computer aid for rapidly and inexpensively constructing and modifying prototypes makes it feasible [10]. The Computer-Aided Prototyping System (CAPS), a research tool developed at the Naval Postgraduate School, is an integrated set of software tools that generate source programs directly from high level requirements specifications [7] (Figure 3). It provides the following kinds of support to the prototype designer:

(1) timing feasibility checking via the scheduler,
(2) consistency checking and automated assistance for project planning, configuration management, scheduling, designer task assignment, and project completion date estimation via the Evolution Control System,
(3) computer-aided design completion via the editors,
(4) computer-aided software reuse via the software base, and
(5) automatic generation of wrapper and glue code.

The efficacy of CAPS has been demonstrated in many research projects at the Naval Postgraduate School and other facilities.

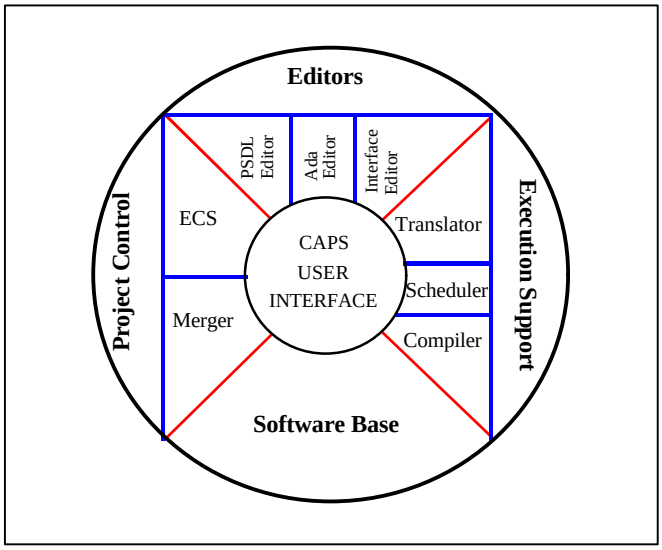


Figure 3. The CAPS Rapid Prototyping Environment

3.1 *Overview of the Caps Method*

There are four major stages in the CAPS rapid prototyping process: software system design, construction, execution, and requirements evaluation/modification (Figure 4).

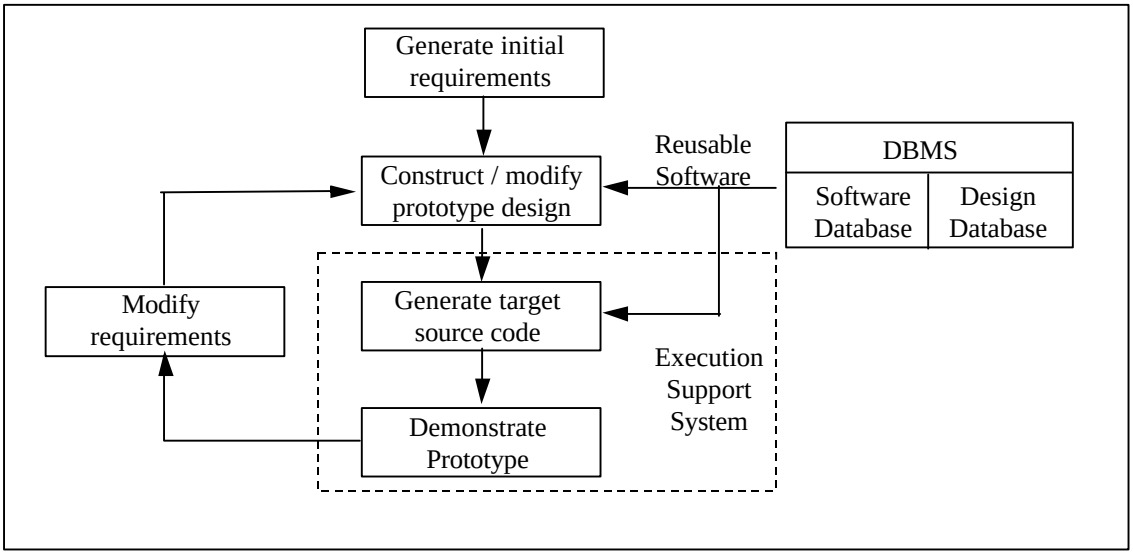


Figure 4. Iterative Prototyping Process in CAPS

The initial prototype design starts with an analysis of the problem and a decision about which parts of the proposed system are to be prototyped. Requirements for the prototype are then generated, either informally (e.g. English) or in some formal notation. These requirements may be refined by asking users to verify their completeness and correctness.

After some requirements analysis, the designer uses the CAPS PSDL editor to draw dataflow diagrams annotated with nonprocedural control constraints as part of the specification of a hierarchically structured prototype, resulting in a preliminary, top-level design free from programming level details. The user may continue to decompose any software module until its components can be realized via reusable components drawn from the software base or new atomic components.

This prototype is then translated into the target programming language for execution and evaluation. Debugging and modification utilize a design database that assists the designers in managing the design history and coordinating change, as well as other tools shown in Figure 3.

3.2 *CAPS as a Requirements Engineering Tool*

The requirements for a software system are expressed at different levels of abstraction and with different degrees of formality. The highest level requirements are usually informal and imprecise, but they are understood best by the customers. The lower levels are more technical, precise, and better suited for the needs of the system analysts and designers, but they are further removed from

the user's experiences and less well understood by the customers. Because of the differences in the kinds of descriptions needed by the customers and developers, it is not likely that any single representation for requirements can be the "best" one for supporting the entire software development process. CAPS provides the necessary means to bridge the communication gap between the customers and developers. The CAPS tools are based on the Prototype System Description Language (PSDL), which is designed specifically for specifying hard real-time systems [5, 6]. It has a rich set of timing specification features and offers a common baseline from which users and software engineers describe requirements. The PSDL descriptions of the prototype produced by the PSDL editor are very formal, precise and unambiguous, meeting the needs of the system analysts and designers. The demonstrated behavior of the executable prototype, on the other hand, provides concrete information for the customer to assess the validity of the high level requirements and to refine them if necessary.

3.3 *CAPS as a System Testing and Integration Tool*

Unlike throw-away prototypes, the process supported by CAPS provides requirements and designs in a form that can be used in construction of the operational system. The prototype provides an executable representation of system requirements that can be used for comparison during system testing. The existence of a flexible prototype can significantly ease system testing and integration. When final implementations of subsystems are delivered, integration and testing can begin before all of the subsystems are complete by combining the final versions of the completed subsystems with prototype versions of the parts that are still being developed.

3.4 *CAPS as an Acquisition Tool*

Decisions about awarding contracts for building hard real-time systems are risky because there is little objective basis for determining whether a proposed contract will benefit the sponsor at the time when those decisions must be made. It is also very difficult to determine whether a delivered system meets its requirements. CAPS, besides being a useful tool to the hard real-time system developers, is also very useful to the customers. Acquisition managers can use CAPS to ensure that acquisition efforts stay on track and that contractors deliver what they promise. CAPS enables validation of requirements via prototyping demonstration, greatly reducing the risk of contracting for real-time systems.

3.5 *A Platform Independent User Interface*

The current CAPS system provides two interfaces for users to invoke different CAPS tools and to enter the prototype specification. The main interface (Figure 5) was developed using the TAE+ Workbench [11]. The Ada source code generated automatically from the graphic layout uses libraries that only work on SUNOS 4.1.X operating systems. The PSDL editor (Figure 6), which allows users to specify the prototype via augmented dataflow diagram, was implemented in C++ and can only be executed under SUNOS 4.1.X environments. A portable implementation of the CAPS main interface and the PSDL editor was needed to allow users to use CAPS to build PSDL prototypes on different platforms. We choose to overcome these limitations by reimplementing

the main interface (Figure 7) and the PSDL editor (Figure 8) using the Java programming language [2].

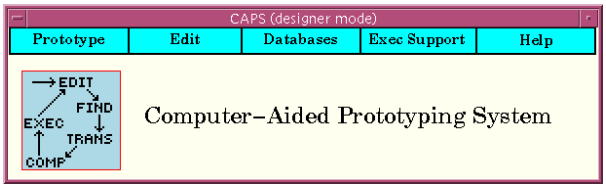


Figure 5. Main Interface of CAPS Release 2.0

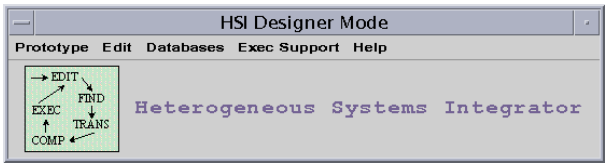


Figure 7. Main Interface of the new CAPS

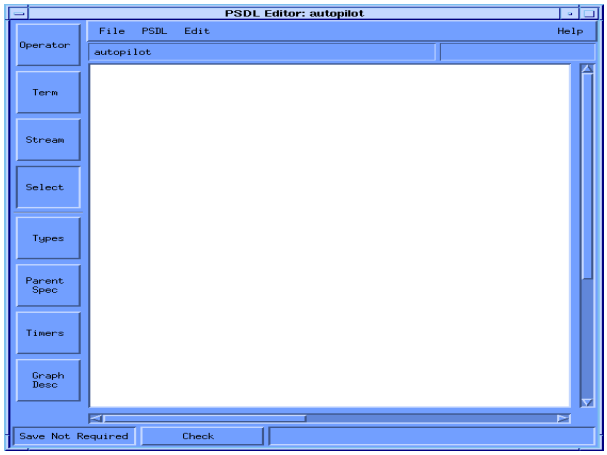


Figure 6. PSDL Editor of CAPS Release 2.0

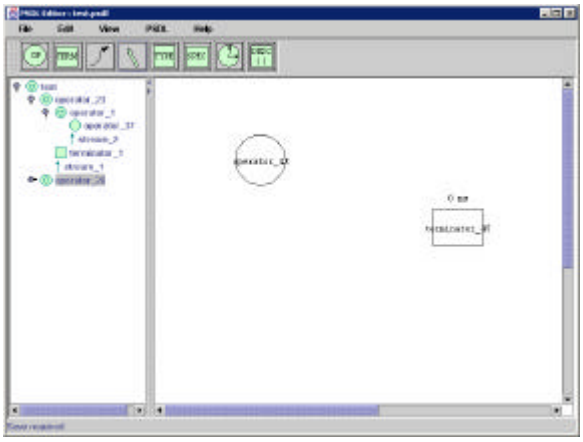


Figure 8. PSDL Editor of the new CAPS

The new graphical user interface, called the Heterogeneous Systems Integrator (HSI), is similar to the previous CAPS. Users of previous CAPS versions will easily adapt to the new interface. There are some new features in this implementation, which do not affect the functionality of the program, but provide a friendlier interface and easier use. The major improvement is the addition of the tree panel on the left side of the editor. The tree panel provides a better view of the overall prototype structure since all of the PSDL components can be seen in a hierarchy. The user can navigate through the prototype by clicking on the names of the components on the tree panel. Thus, it is possible to jump to any level in the hierarchy, which was not possible earlier.

4. A Simple Example: Prototyping a C3I Workstation

To create a first version of a new prototype, users can select “New” from the “Prototype” pull-down menu of the CAPS main interface (Figure 9). The user will then be asked to provide the name of the new prototype (say “c3i_system”) and the CAPS PSDL editor will be automatically invoked with a single initial root operator (with a name same as that of the prototype).

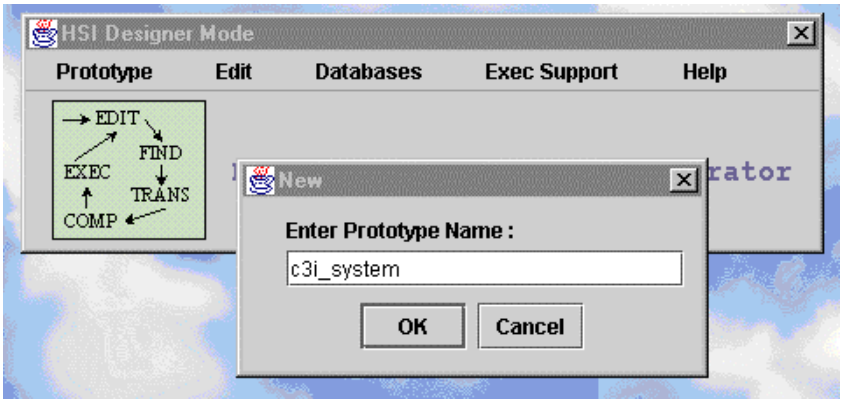


Figure 9. Creating a new prototype called C3I_System

CAPS allows the user to specify the requirements of prototypes as augmented dataflow graphs. Using the drawing tools provided by the PSDL editor, the user can create the top-level dataflow diagram of the c3i_system prototype as shown in Figure 10, where the c3i_system prototype is modeled by nine modules, communicating with each other via data streams. To model the dynamic behavior of these modules, the dataflow diagram is augmented with control and timing constraints. For example, the user may want to specify that the weapons_interface module has a maximum response time of 3 seconds to handle the event triggered by the arrival of new data in the weapon_status_data stream, and it only writes output to the weapon_emrep stream if the status of the weapon_status_data is damage, service_required, or out_of_ammunition. CAPS allow the user to specify these timing and control constraints using the pop-up operator property menu (Figure 11), resulting in a top-level PSDL program shown in Figure 12.

To complete the specification of the c3i_system prototype, the user must specify how each module will be implemented by choosing the implementation language for the module via the operator property menu. The implementation of a module can be in either the target programming language or PSDL. A module with an implementation in the target programming language is called an atomic operator. A module that is decomposed into a PSDL implementation is called a composite operator. Module decomposition can be done by selecting the corresponding operator in the tree-panel on the left side of the PSDL editor.

CAPS supports an incremental prototyping process. The user may choose to implement all nine modules as atomic operators (using dummy components) in the first version, so as to check out the global effects of the timing and control constraints. Then, he/she may choose to decompose the comms_interface module into more detailed subsystems and implement the sub-modules with reusable components, while leaving the others as atomic operators in the second version of the prototype, and so on.

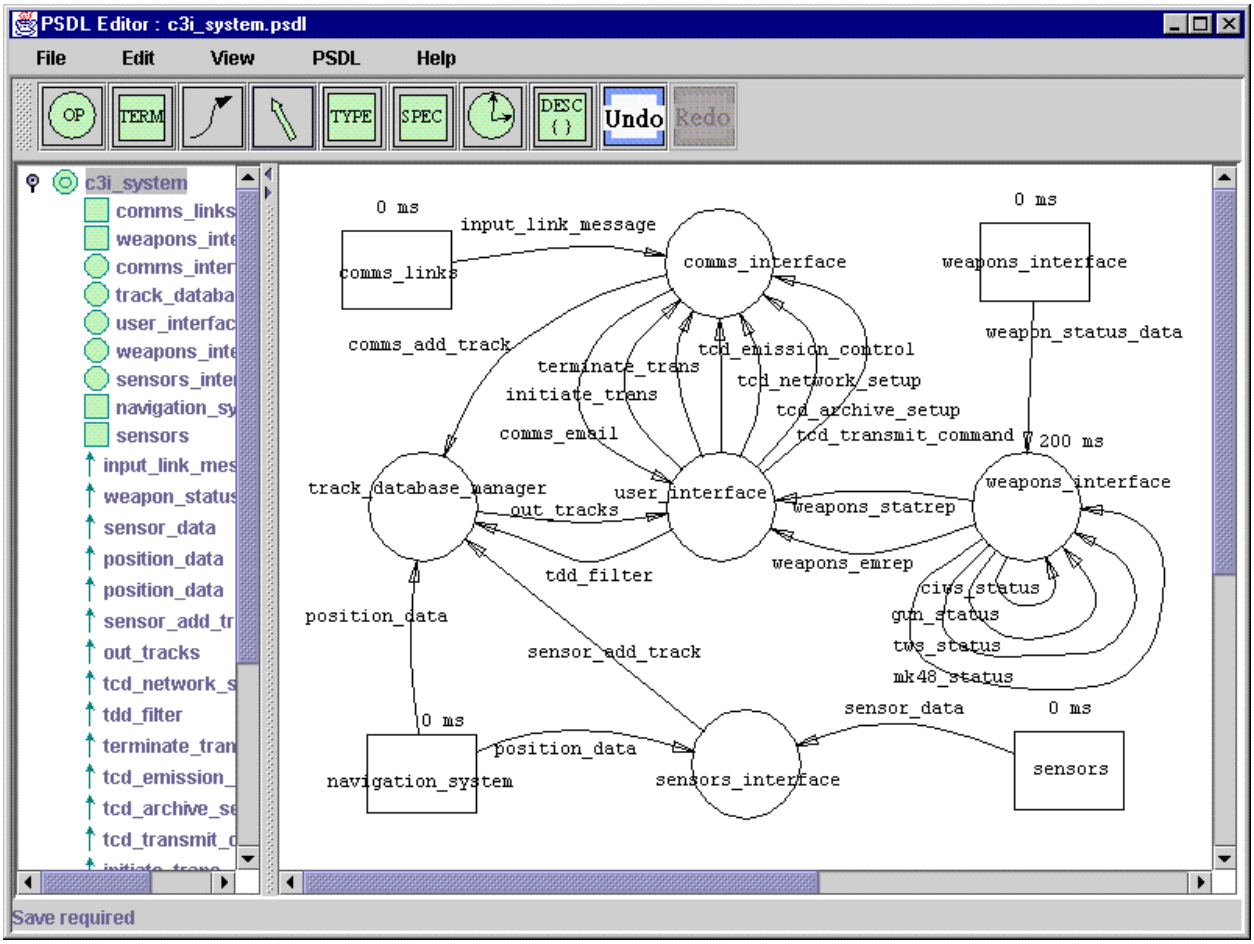


Figure 10. Top-level Dataflow Diagram of the c3i_system.

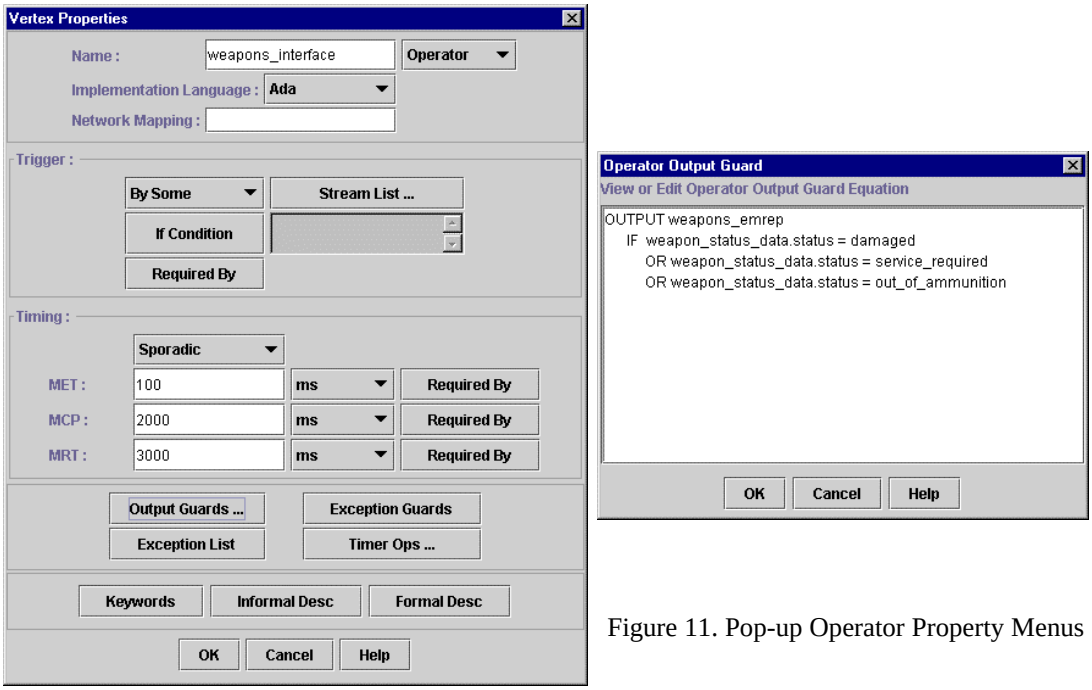
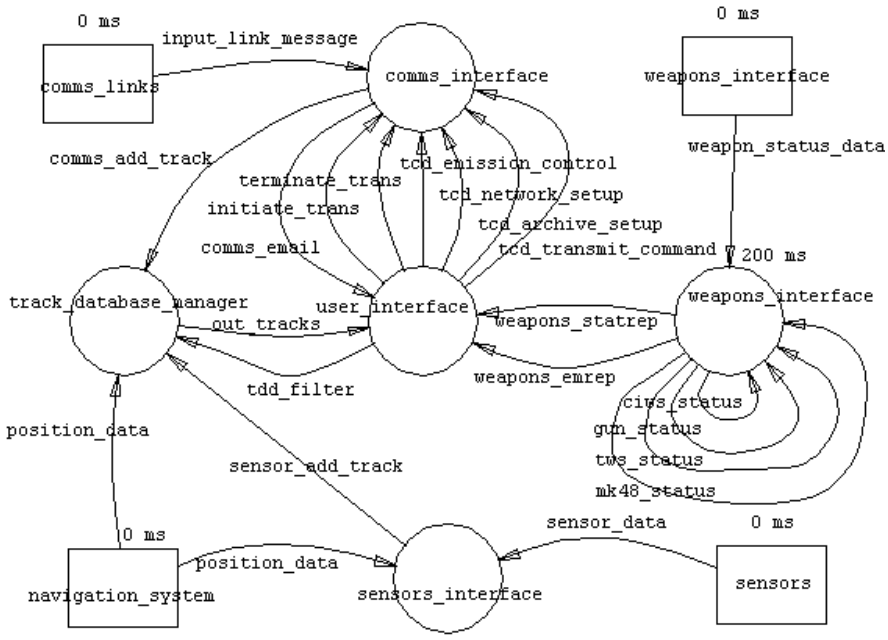


Figure 11. Pop-up Operator Property Menus

```
OPERATOR c3i_system
SPECIFICATION
   DESCRIPTION
      {This module implements a simplified version of
       a generic C3I workstation.}
END
IMPLEMENTATION
    GRAPH
```




```
    DATA STREAM
       -- Type declarations for the data streams in the graph go here.
    CONTROL CONTRAINTS
       OPERATOR comms_links
          PERIOD 30000 MS

       OPERATOR navigation_system
          PERIOD 30000 MS

       OPERATOR sensors
          PERIOD 30000 MS

       OPERATOR weapons_systems
          PERIOD 30000 MS

       OPERATOR weapons_interface
          TRIGGERED BY SOME
             weapon_status_data
          MINIMUM CALLING PERIOD 2000 MS
          MAXIMUM RESPONSE TIME 3000 MS
          OUTPUT
             weapons_emrep
             IF weapon_status_data.status =
                      damaged
             OR weapon_status_data.status =
                      service_required
             OR weapon_status_data.status =
                      out_of_ammunition
END
```

Figure 12. Top-level Specification of the c3i_system

To facilitate the testing of the prototypes, CAPS provides the user with an execution support system that consists of a translator, a scheduler and a compiler. Once the user finishes specifying the prototype, he/she can invoke the translator and the scheduler from the CAPS main interface to analyze the timing constraints for feasibility and to generate a supervisor module for each subsystem of the prototype in the target programming language. Each supervisor module consists of a set of driver procedures that realize all the control constraints, a high priority task (the static schedule) that executes the time-critical operators in a timely fashion, and a low priority dynamic schedule task that executes the non-time-critical operators when there is time available. The supervisor module also contains information that enables the compiler to incorporate all the software components required to implement the atomic operators and generate the binary code automatically. The translator/scheduler also generates the glue code needed for timely delivery of information between subsystems across the target network.

For prototypes which require sophisticated graphic user interfaces, the CAPS main interface provides an interface editor to interactively sculpt the interface. In the c3i_system prototype, we choose to decompose the comms_interface, the track_database_manager and the user_interface modules into subsystems, resulting in hierarchical design consisting of 8 composite operators and twenty-six atomic operators. The user interface of the prototype has a total of 14 panels, four of which are shown in Figure 13. The corresponding Ada program has a total of 10.5K lines of source code. Among the 10.5K lines of code, 3.5K lines comes from supervisor module that was generated automatically by the translator/scheduler and 1.7K lines that were automatically generated by the interface editor [9].

5. **Conclusion**

CAPS has been used successfully as a research tool in prototyping large war-fighter control systems (e.g. the command-and-control station, cruise missile flight control system, missile defense systems) and demonstrated its capability to support the development of large complex embedded software. Specific payoffs include:

(1) Formulate/validate requirements via prototype demonstration and user feedback
(2) Assess feasibility of real-time system designs
(3) Enable early testing and integration of completed subsystems
(4) Support evolutionary system development, integration and testing
(5) Reduce maintenance costs through systematic code generation
(6) Produce high quality, reliable and flexible software
(7) Avoid schedule overruns

In order to evaluate the benefits derived from the practice of computer-aided prototyping within the software acquisition process, we conducted a case study in which we compared the cost (in dollar amounts) required to perform requirements analysis and feasibility study for the c3i system using the 2167A process, in which the software is coded manually, and the rapid prototyping process, where part of the code is automatically generated via CAPS [3]. We found that, even under very conservative assumptions, using the CAPS method resulted in a cost reduction of $56,300, a 27% cost saving. Taking the results of this comparison, then projecting to a mission control software system, the command and control segment (CCS), we estimated that there would

be a cost saving of 12 million dollars. Applying this concept to an engineering change to a typical component of the CCS software showed a further cost savings of $25,000.

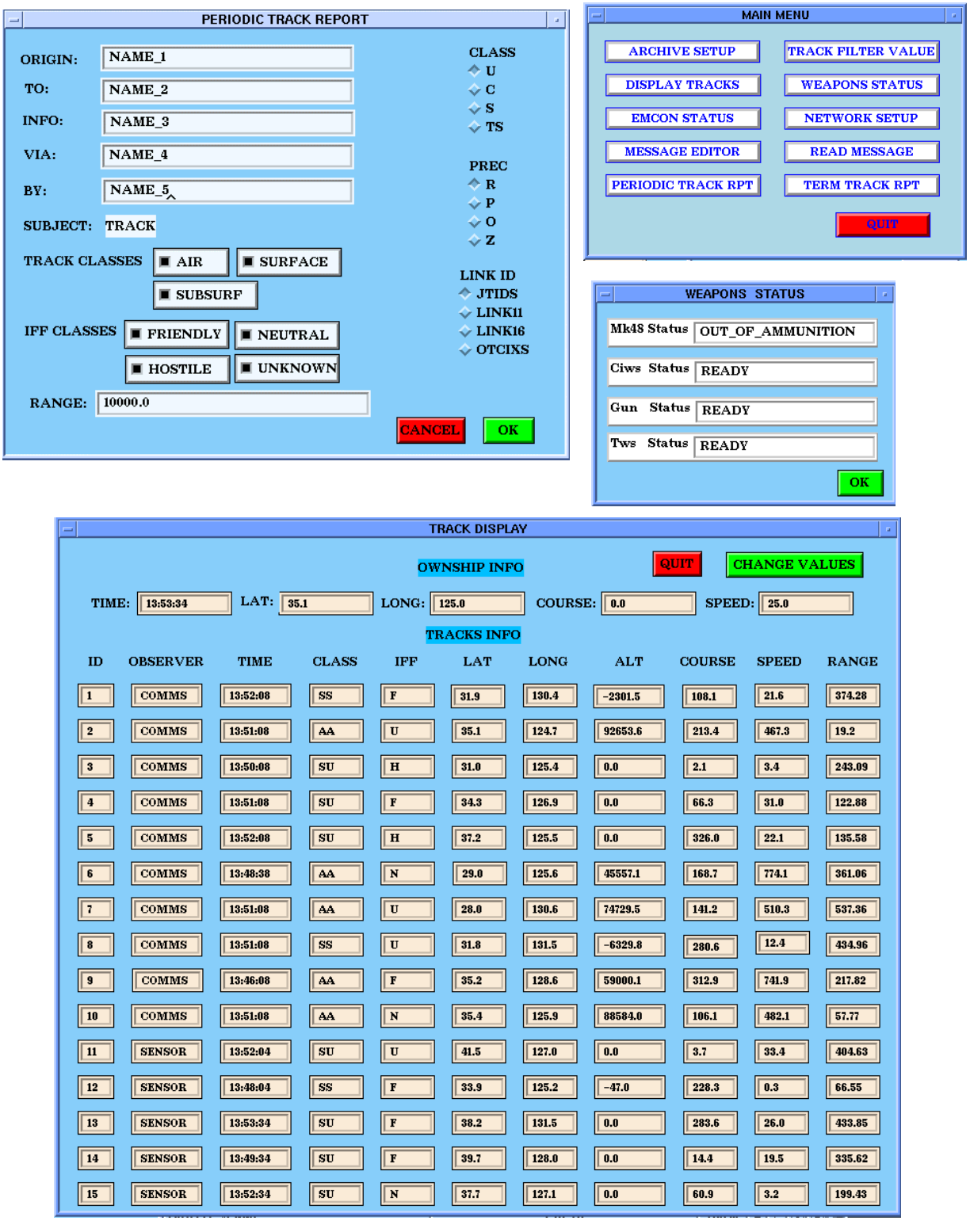


Figure 13. User Interface of the c3i_system

6. **References**

[1] L. Bernstein, “Forward: Importance of Software Prototyping”, *Journal of Systems Integration - Special Issue on Computer Aided Prototyping*, 6(1), pp. 9-14, 1996.

[2] I. Duranlioglu, *Implementation of a Portable PSDL Editor for the Heterogeneous Systems Integrator*, Master’s thesis, Naval Postgraduate School, Monterey, California, March 1999.

[3] M. Ellis, *Computer-Aided Prototyping Systems (CAPS) within the software acquisition process: a case study*, Master’s thesis, Naval Postgraduate School, Monterey, California, June 1993.

[4] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, 1995.

[5] B. Kraemer, Luqi and V. Berzins, “Compositional Semantics of a Real-Time Prototyping Language”, *IEEE Transaction on Software Engineering*, 19(5), pp. 453-477, 1993.

[6] Luqi, V. Berzins and R. Yeh, “A Prototyping Language for Real-Time Software”, *IEEE Transaction on Software Engineering*, 14(10), pp. 1409-1423, 1988.

[7] Luqi and M. Ketabchi, “A Computer-Aided Prototyping System”, *IEEE Software*, 5(2), pp. 66-72, 1988.

[8] Luqi, “Computer-Aided Software Prototyping”, *IEEE Computer*, pp. 111-112, September 1991.

[9] Luqi, “Computer-Aided Prototyping for a Command-and-Control System Using CAPS”, *IEEE Software*, 9(1), pp. 56-67, 1992.

[10] Luqi, “System Engineering and Computer-Aided Prototyping”, *Journal of Systems Integration - Special Issue on Computer Aided Prototyping*, 6(1), pp. 15-17, 1996.

[11] TAE Plus Programmer’s Manual (Version 5.1). Prepared for: NASA Goddard Space Flight Center, Greenbelt, Maryland. Prepared by: Century Computing, Inc., Laural, Maryland, April 1991.