Applying a Computer Aided Prototyping System to the Software of an Autonomous Underwater Vehicle

Bihari, Thomas E.; McGhee, Robert B.; Luqi; Lee, Yuh-jeng

Applying a Computer Aided Prototyping System to the Software of an Autonomous Underwater Vehicle

Thomas E. Bihari*, Robert B. McGhee, Luqi, Yuh-jeng Lee

Department of Computer Science
Naval Postgraduate School
Monterey, California 93943
(408) 646-2449

*** Adaptive Machine Technologies, Inc.**
1218 Kinnear Road
Columbus, Ohio 43212
(614) 486-7741

I. Introduction

This workshop addresses an important direction for tool development. Within the current state of the practice, a great deal of duplicated effort is spent developing similar software systems within a particular application domain, such as distributed, intelligent vehicle control. The experience gained during these projects is wasted if it cannot be used to aid the development of subsequent, similar projects.

We believe that this experience can provide the basis for developing a common ground for all the applications in a domain. The development of domain-specific architectures and tools supporting the essential processes and properties of particular application domains will allow reuse of the domain knowledge and domain-specific solution techniques that comprise the most expensive part of the effort to develop new systems.

Expensive tool implementation efforts can be wasted if tool construction is started without a clear idea of the problems the tools are supposed to solve and without a systematic and formalized set of solution techniques to be incorporated in the tools. Constructing tools is both labor intensive and skill intensive, and involves knowledge both of software engineering principles and of the application domain. Because this combination is hard to find, tools developed by software engineering researchers are often "demo driven" and lack applicability, while those developed by application domain experts are often ad hoc and lack strong foundations.

The best tools are those which are based on strong theoretical principles, and also driven by a strong, vocal user community. This can be difficult to achieve when developing tools that push the state of the art, regardless of whether the effort is done by industry or by universities. The perceived reward structure for researchers in the tool provider and user communities usually makes such interaction seem undesirable - more work with little direct payoff. DOD support is essential in building an interaction between the providers and users of domain-specific software development tools. Appropriate modes of interaction should be identified and supported.

This paper presents the result of a study of the potential for interaction between two on-going research projects at the Naval Postgraduate School. The Computer-Aided Prototyping System (CAPS) project[1] is developing models and support tools for rapid prototyping of embedded real-time software. The Autonomous Underwater Vehicle (AUV) project[2] is developing a computer-controlled submersible vehicle.

The purpose of this study was to examine the goals of the AUV project and its resulting software requirements, and the goals and capabilities of the CAPS project, and to determine the benefits of pursuing joint research in the application of CAPS to the AUV software.

II. The Computer-Aided Prototyping System Project

The goal of the Computer Aided Prototyping System (CAPS) project [Luqi88] [Luqi88a] [Luqi88b] [Luqi89] is to enable rapid prototyping of parallel and distributed real-time software, as a way of increasing productivity and decreasing software costs. The CAPS project focuses on automated methods for retrieving, adapting, and combining reusable components based on normalized module specifications; establishing feasibility of real-time constraints via scheduling algorithms; simulating unavailable components via algebraic specifications; automatically generating translators and real-time schedules for supporting execution; constructing a prototyping project database using derived mathematical models; providing automated design completion and error checking facilities in a designer interface; and establishing a convenient graphical interface for design and debugging.

CAPS is a set of software tools, sharing a common basis consisting of a rapid-prototyping software development methodology, an enhanced-dataflow computational model and a prototyping language. The CAPS tool set includes a graphical editor, a syntax directed editor, a database of existing software components, a database of existing software designs, a translator which converts the prototyping language into a particular implementation language (e.g., Ada), static and dynamic task schedulers, a debugger, and others. The tool set is running on a Sun SPARCstation under UNIX and X-Windows, and is portable to any system with UNIX and X-Windows. The product produced by the system is Ada code, which is portable to any system with an adequate Ada compiler. The system can be used to design distributed and intelligent systems [Luqi89a].

CAPS' support for the rapid-prototyping methodology makes it possible for prototypes to be designed quickly and to be executed to validate the requirements. CAPS manages the entire prototyping process, from the development of the software design, through the retrieval or creation of reusable Ada software components, to the generation, execution, and analysis of the resulting Ada program. CAPS users may iterate through this process until they are satisfied with the software's behavior.

1. The CAPS project is directed by Dr. Luqi and is supported by the National Science Foundation under grant number CCR-8710737.

2. The AUV project is directed by Dr. A.J. Healey and is supported by Naval Postgraduate School funding.

The CAPS computational model and tools provide the designer of a software system with a way to draw an augmented dataflow diagram which contains the necessary timing and control constraints for specifying embedded software systems. The model maximizes parallelism by enforcing timing and control constraints only where necessary. The graphical design and the constraints drawn through the CAPS graphical editor for an embedded control system are based on the syntactical structure of the Prototype System Description Language (PSDL) [Luqi88a].

The specification part of the PSDL program describes the basic attributes of required software components. (Currently the components are Ada units, but other languages can also be supported.) This information is used by a tool which searches for appropriate reusable components stored in the software base. If no suitable component is found in the existing software base, the designer may choose to create a completely new component from scratch or to create a new component by combining or modifying an existing set of components. When the design is completed, the PSDL program is translated into Ada code which has the structures for realizing the timing and control constraints built in. The Ada program is then compiled.

The designer may then execute the program and evaluate the prototype's behavior against the behavior that he expected it to have. If the comparison results are not satisfactory, the designer may modify the prototype and evaluate the prototype again. This process continues until the prototype meets the requirements.

CAPS was designed to be used for developing prototypes for real-time systems. In CAPS, a hard real-time constraint is a bound on the response of a process which must be satisfied under all operating conditions. CAPS specifications can represent a variety of real-time constraints, including (1) maximum execution times for modules, (2) minimum calling periods, (3) synchronization of processes with sporadically-arriving external data or interrupts, (4) delays required by limitations on input/output devices, (5) maximum response times, and (6) periodic system actions.

The CAPS model and tool set have been applied to real-time software designs in several areas, including C3I [Luqi89] and process control [Luqi88a].

III. The Autonomous Underwater Vehicle Project

III.1. Overview of the AUV-II

The AUV-II is the second in a series of autonomous underwater vehicles developed at the Naval Postgraduate School. It is described in detail in [Cloutier90] [Healey89] [Kwak90]. Briefly, the AUV-II is a self-contained vehicle, approximately 16 inches wide, 10 inches deep, and 93 inches long. It displaces approximately 387 pounds and is powered by on-board batteries.

The AUV-II is a research vehicle designed as a testbed for research in mission planning, path planning, sonar data analysis and world modeling, navigation through obstacle fields, and other intelligent behaviors.

The AUV-II is propelled by two main screws (port and starboard aft) and four tunnel thrusters (fore and aft vertical, and fore and aft athwartships). These may be used to control five degrees of freedom; the AUV-II's roll axis is not controlled. However, when the AUV-II is moving with sufficient speed, the control surfaces (bow planes, stern planes, and fore and aft rudders) may be used in conjunction with the screws and thrusters to control all six degrees of freedom, including roll.

The AUV-II's sensor system consists of four pencil-beam sonar transducers mounted in the AUV-II's nose, a full suite of inertial sensors (three rate gyros, three accelerometers, a vertical gyro and a directional gyro), a flux gate compass, a paddle-wheel speed sensor, and individual motor RPM sensors.

Because the AUV-II is a research vehicle, its computational requirements are subject to change. It is important that the on-board computer hardware be modifiable and extensible, by adding more raw computing power, memory, and I/O devices, and by adding different types of these components.

The on-board computer is centered around a 12-slot GESPAC G-96 bus. The bus currently hosts one GESPAC MPU-20HF single-board computer (25 MHz Motorola 68020 and 68882 processors, 2.5 Mb of RAM, and up to 4 Mb of EPROM), and 5 other boards containing interfaces to a 200 Mb hard disk, parallel and serial communication ports, analog-to-digital input channels interfaced to the AUV-II's sensors, and digital-to-analog output channels interfaced to the AUV-II's effectors. There are currently 6 free bus slots. These are expected to be used for additional GESPAC MPU-30HF (68030-based) boards, a Transputer board, and other devices as necessary.

The GESPAC computer uses Microware's OS-9 real-time operating system. OS-9 is a full operating system, with a file system, native compilers and other development tools. OS-9 uses a time-slicing, prioritized, task scheduler. Intertask communication is via global memory, pipes, signals, and BSD4.2 sockets for inter-processor communication.

In addition to the AUV-II itself, a laboratory computer identical to that in the AUV-II, and a graphical simulation of the AUV-II and its environment (running on a Silicon Graphics workstation), are available for software development.

III.2. Characteristics of the AUV-II Software

The AUV-II's software is described in detail in [Cloutier90]. Our main interests for this study were not in the particular guidance and control algorithms, but in the duties performed by the software, the software's real-time requirements, the overall software design, and the expected life cycle of the software.

The AUV-II's software is designed as a layered architecture in which higher levels pass requested vehicle states to lower levels. The lower levels attempt to meet the requests, possibly modifying them to make them feasible, and may pass information back to the higher levels, allowing the higher levels to modify future requests. There are currently three levels. The top level consists of the Mission Planner (which is off-board), and the Mission Replanner (an on-board planning subsystem which may override the off-board planner).

These subsystems generate sets of paths describing particular missions.

The middle level consists of the Guidance sub-system, which receives paths from the Mission (Re)Planner and calculates individual "postures" to be achieved by the AUV-II. The bottom level consists of the Autopilot subsystem, which servos the AUV-II's effectors to achieve the requested postures. This is performed on a 100 ms period.

To provide input to the Autopilot's servo control loop, the state of the AUV-II must be determined from the inertial, depth, and speed sensors. This must be done by the Navigation sub-system at rates sufficient to provide an accurate current state. The AUV-II state is currently updated every 100 ms. Project goals call for data from the sonar sensors to be integrated with other sensor data, and with pre-loaded obstacle maps, in several phases. Initially, sonar data will be used to correct inertial sensor drift. Eventually, sonar data will be used for collision avoidance and for revising the existing world model. Sonar data will be collected at 100 ms periods, and world modeling will occur at somewhat longer periods.

The relative steering effectiveness of the thrusters vs the control surfaces depends on the AUV-II's forward velocity. It is anticipated that the control surfaces' effectiveness will vary from zero at zero velocity to approximately four times that of the thrusters at maximum velocity. Therefore, the AUV-II motion control strategy, and the control software, has been divided into two modes: Hovering Mode and Transit Mode. The software must be able to cleanly switch between these modes while in operation.

In addition to these "normal" operations, reflex actions like collision avoidance may be triggered by special circumstances and must produce quick responses, sometimes overriding existing activities.

The software characteristics of the AUV-II are both similar to and different from those of the Adaptive Suspension Vehicle (ASV), a three-ton, self-contained, six-legged walking vehicle[1] with which we have been associated in the past [Bihari89]. Both vehicles perform sensing, world modeling, motion planning, and servo control in real time.

For the most part, the AUV-II's guidance and control software is not required to meet extremely tight real-time requirements. Sensing and servo control periods are on the order of 100 ms or greater, with some allowable slippage. This is easily within the capabilities of existing computer hardware, real-time operating system, and software technologies. The ASV has tighter real-time requirements than the AUV-II. For example, the ASV's leg servo control software executes with periods of 10 ms or less, with serious consequences if servo cycles are missed. The ASV's real-time requirements are well-defined and generally situation-independent, however.

The AUV-II is required to be completely autonomous, while the ASV has an on-board operator performing many of the high-level world modeling and motion planning duties. The AUV-II must maintain a larger view of time (e.g., for an entire mission). For example,

1. Sponsored by the Defense Advanced Research Projects Agency under contracts MDA903-82-K-0058, DAAE07-84-K-R001, and MDA972-88-C-0031.

planning may take a significant amount of time, and the amount of time may be situation-dependent. The AUV-II must be capable of reasoning about time, and of "planning to plan", and the enforcement of the resulting timing constraints must be handled by the underlying operating system and support tools.

Furthermore, much of the information contained in the AUV-II system is time-dependent. That is, the AUV-II's perception of the state of itself, obstacles, and mission plans is dependent on the relationship between the current time and the time at which the information was created (e.g., the age of the data). Portions of the AUV-II's software may resemble a temporal database.

The AUV-II is experimental, and the software's duties range from low-level sensor data processing and servo control to high-level planning and world modeling. Ideally, the AUV-II software development environment would support the integration of a variety of programming paradigms, including procedural, functional, object-oriented, logic-based, and rule- or frame-based. Practically, a system supporting Ada and Common Lisp could provide a basis for most of these paradigms. (This would be a step forward in the state of the practice. Almost all existing AUVs, including the AUV-II, are programmed in C.)

In summary, the AUV-II software system has the following characteristics:

From a software architecture standpoint:

1. It is hierarchically structured, and it can best be understood by viewing it at different levels of abstraction for different purposes.

2. It consists of subsystems, some of which are tightly coupled, others of which are loosely coupled (and execute at different rates).

3. It operates in at least two separate modes.

4. It must occasionally perform reflex actions which override normal operations.

5. Most of the computations have real-time constraints.

6. It includes time-dependent representations of the states of the AUV-II and environment.

From a software management standpoint:

1. The specification, design, and implementation of the entire system (mechanical hardware, electrical and electronic hardware, and software) will evolve as existing research questions are answered and new questions are asked.

2. Small changes to the software can be expected to occur frequently. That is, software development will follow an experimental, iterative, implement-execute-evaluate cycle. The software may also need to be specially configured for specific missions.

3. Multiple versions of the software may be "active" at the same time, as different

researchers conduct independent experiments using specialized components integrated with a common software base.

4. The software base can be expected to outlive (in a project sense) most of the software developers. Software development methodology support and enforcement is important.

5. It must be possible for different people to understand and manipulate the system at different levels of abstraction (e.g., as "black boxes"), so they not have to learn the entire system in order to perform useful research. It must not take too long to "come up to speed".

6. Different languages and programming paradigms may be most effective for different components (or different versions of the same component). A uniform framework for managing these disparate components is needed.

IV. The Potential for Further CAPS-AUV Project Cooperation

In theory, the interaction of a real-time software tool provider (the CAPS project) with a real-time software tool user (the AUV project) has many advantages. The CAPS project would benefit from the availability of a realistic application. The AUV project would benefit from an improved software development methodology and support tools. In practice, the interaction of two such research projects must be realistic and well-defined if it is to be beneficial to both parties.

In our view, CAPS provides an appropriate and extremely useful methodology for developing real-time control software like that of the AUV project. The concepts supported by CAPS generally match those we expect for the AUV-II's life cycle. The integrated tool set should lead to easier software development and strict enforcement of the software development methodology. PSDL seems to contain the features necessary for the AUV software.

There are practical considerations, however. For example, the current AUV software is written in C, while CAPS supports only Ada at this time. The CAPS tools currently run under X Windows on a Sun SPARCstation, while the AUV tools are running on an IBM PC/AT compatible. Resolution of these practical matters could consume valuable "research" time. Some care is also needed because a complete treatment of the problem requires solutions to two unsolved research problems: real-time databases and real-time scheduling. Domain-specific assumptions and approaches must be developed to provide adequate solutions to these problems. Some progress in these directions has already been made [Galik88] [Guentenburg89] [Huskins90] [Mostov90] [Sun90] [White89], but these solutions have not yet been incorporated into the current implementation of CAPS.

We see the potential for a step by step increase in interaction between the CAPS and AUV projects. This should begin by establishing a realistic set of goals. Those goals might include, for example:

1. Formulate the AUV-II software design in PSDL and critique the design.

2. Translate the AUV-II's existing C code to Ada, and move the AUV-II development

environment to a platform with appropriate Ada tools and the X-Windows support needed by CAPS (e.g., Sun or DEC MicroVAX).

3. Form the AUV-II's Ada modules into CAPS reusable components and develop a complete AUV-II software version under CAPS.

And so on.

It is important to avoid over-integrating the two projects. In order to avoid delaying the progress of either project, the projects should maintain independent critical paths. For example, the AUV programmers should continue to develop C code until the Ada development environment is fully operational. A significant benefit might be gained by interaction at the design level (e.g., Goal 1) regardless of the eventual implementation of AUV-II software under CAPS.

V. Conclusion

The number and complexity of intelligent, autonomous, real-time systems are expected to grow, driven by the need to perform missions for which human supervision is unavailable or not cost-effective. The development and maintenance of software for such systems is an important area of research. We believe that progress in this area is achieved best by the cooperation of the providers of real-time software engineering technology (e.g., CAPS) and the users of that technology (e.g., AUV). Appropriate modes of interaction must be found.

References

[Bihari89] Bihari, T., Walliser, T. and Patterson, M., "Controlling the Adaptive Suspension Vehicle", IEEE COMPUTER, pp. 59-65, June 1989.

[Cloutier90] Cloutier, M., "Guidance and Control System for an Autonomous Vehicle", Masters Thesis, Naval Postgraduate School, June 1990.

[Galik88] Galik, D., "A Conceptual Design of a Software Base Management System for the Computer Aided Prototyping System", Masters Thesis, Naval Postgraduate School, December 1988.

[Guentenburg89] Guentenburg, H., "Automatic Generation of an Aircraft Inertial Navigation System", Masters Thesis, Naval Postgraduate School, May 1989.

[Healey89] Healey, A., Papoulias, F., and MacDonald, G., "Design and Experimental Verification of a Model Based Compensator for Rapid AUV Depth Control", Proceedings of the 6th Unmanned, Untethered, Submersible Technology Conference, Washington, D.C., June 12-14, 1989.

[Huskins90] Huskins, J., "Issues in Expending the Software Base Management System Supporting the CAPS", Masters Thesis, Naval Postgraduate School, June 1990.

[Kwak90] Kwak, S., Ong, S. and McGhee R., "A Mission Planning Expert System for an Autonomous Underwater Vehicle", IEEE Symposium on Autonomous Underwater Vehicle

Technology, pp. 123-128, June 1990.

[Luqi88] Luqi and Berzins, V., "Rapidly Prototyping Real-Time Systems", IEEE Software, pp. 25-36, September 1988.

[Luqi88a] Luqi, Berzins, V. and Yeh, R., "A Prototyping Language for Real-Time Software", IEEE Transactions on Software Engineering, pp. 1409-1423, October 1988.

[Luqi88b] Luqi, "Knowledge-Based Support for Rapid Software Prototyping", IEEE Expert, pp. 9-18, Winter 1988.

[Luqi89] Luqi and Davis, T., "A Software Prototype of the Message Processor in Navy C3I Station", Naval Postgraduate School Technical Report NPS52-90-010, August 1989.

[Luqi89a] Luqi, Berzins, V., Kraemer, B., and White, L., "A Proposed Design for a Rapid Prototyping Language", Naval Postgraduate School Technical Report NPS52-89-045, March 1989.

[Mostov90] Mostov, I., "A Model of Software Maintenance for Large Scale Military Systems", Masters Thesis, Naval Postgraduate School, June 1990.

[Sun90] Sun, J., "Developing Portable User Interfaces for Ada Command & Control Software", Masters Thesis, Naval Postgraduate School, June 1990.

[White89] White, L., "The Development of a Rapid Prototyping Environment", Masters Thesis, Naval Postgraduate School, December 1989.
