1992

CAPS - A Tool for Real-Time System Development and Acquisition

Luqi; Shing, M.

Naval Research Reviews

Luqi and M. Shing, "CAPS - A Tool for Real-Time System Development and Acquisition", Naval Research Reviews, 1992, Vol. XLIV (44), pp. 12-16.
https://hdl.handle.net/10945/65713

CAPS – A Tool For Real-Time System Development and Acquisition

Luqi and Man-Tak Shing
Computer Science Department

Introduction

The Computer Aided Prototyping System (CAPS) is an integrated software development environment aimed at rapidly prototyping hard real-time embedded software systems, such as missile guidance systems, space shuttle avionics systems, and military Command, Control, Communication and Intelligence (C3I) systems. One of the major differences between a hard real-time system and a conventional system is that the application software must meet its deadlines even under worst case conditions[1]. The process of design and development of these systems is often plagued with uncertainty, ambiguity and inconsistency. The timing requirements are difficult for the user to provide and for the analysts to determine. It is also very difficult to determine whether a delivered system meets its requirements.

Traditional software development methods conduct extensive testing near the end of the project in an attempt to ensure proper functioning of the system. The major weakness of this approach is that there is no way to recover from major faults discovered at the end of the project, when available funds have been nearly exhausted. Furthermore, delivered systems that meet faulty requirements can satisfy a contract without being useful to the sponsor. The inadequacy of current software development methods is evident in high software costs and low programmer productivity. Recognizing the increased cost and importance of software development for Command and Control (C2) systems, the Secretary of Navy (SECNAV) promulgated an instruction addressing software development and acquisition[2]. This instruction documents SECNAV concern for defining a DoN acquisition policy for software-intensive systems and increasing user involvement during the design and development stages. The policy combining these two concerns states:

"To promote effective interaction between the user and the developer, software prototyping methods shall be used in the design and construction of C2 information systems. Early delivery of software systems is emphasized through the use of prototyping methods"[2].

The rapid prototyping approach to software development provided by CAPS[3] supports this policy. Rapid prototyping uses rapidly constructed prototypes to help both the developers and their customers visualize the proposed system and assess its properties in an iterative process. A prototype is an executable pilot version of the intended system that accurately reflects chosen aspects of the system. Rapid prototyping may be used in conjunction with or as an alternative to the traditional software development cycle.

The Computer Aided Prototyping System

The Computer Aided Prototyping System (CAPS) provides facilities for computer-aided design, software component reuse, and automated Ada code generation. These tools are developed to help software engineers rapidly construct and adapt software, validate and refine user requirements, and to

check consistency of proposed design[4]. The process supported by CAPS provides requirements and designs in a form that is useful in the construction of the operational system, as well as in the acquisition process to assess optimized implementations delivered by contractors, and to integrate independently developed subsystems. CAPS allows the user to specify the requirements of a proposed system informally in a structured top-down fashion using easy-to-understand graphics, assists the designer in augmenting the informal specifications with formal annotations, and automatically translates the result into executable Ada code.

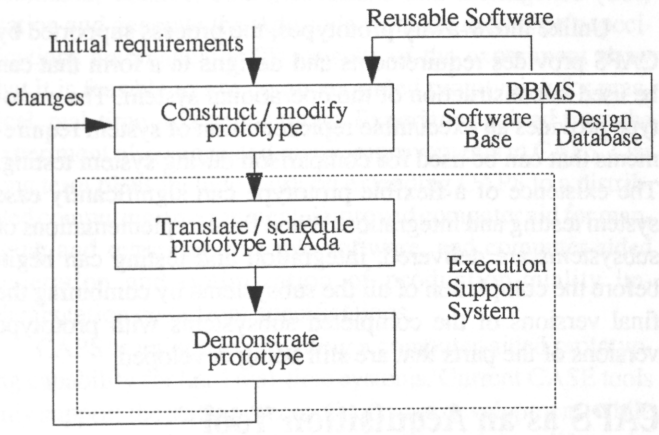
The CAPS Method

There are four major stages in the CAPS rapid prototyping process: software system design, construction, execution, and debugging/modification.

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 the requirements analysis is completed, the designer uses the CAPS graphic editor to draw dataflow diagrams with non-procedural 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 underlying computational model unifies dataflow and control flow, and provides a mechanism for developing top-down decompositions. 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. The construction of the prototype is aided by the syntax-directed editor which helps the designer create syntactically correct timing and control constraints for the operators. This prototype is then translated into Ada for execution and evaluation. Debugging and modification utilize a design database that assists the designers in managing the design history and coordinating changes as well as the other tools described above.

Figure 1.

Iterative Prototyping Process in CAPS



The Prototype System Description Language

The CAPS tools are based on the Prototype System Description Language (PSDL). PSDL is a high-level language designed specifically to support the specification of real-time software systems, as well as to organize and retrieve reusable components in the software base[5]. PSDL lets designers sketch a system using computation graphs, where the vertices are *operators* and the directed edges are *data streams*, and then refine the design by adding timing and control constraints in text form.

Operators are state machines whose internal states are modeled by state variables. Operators with an empty variable set behave like functions. PSDL operators can be triggered by data (*sporadic* operators) or by periodic timing constraints (*periodic* operators). When triggered, an operator will produce output based on input values and values of internal state variables. There are two kinds of operators: *atomic* operators and *composite* operators. An atomic operator is one that can be realized by an implementation stored in the software base or supplied by the software engineers, and a composite operator is one that can be decomposed into a network of more primitive operators represented as enhanced dataflow diagrams.

Operators communicate via two kinds of data streams: *dataflow* streams and *sampled* streams. A dataflow stream can be thought of as a FIFO buffer of capacity one that connects synchronized operators. Data in dataflow streams represents discrete transactions, and is removed from the stream when read. A sampled stream can be thought of as a single memory cell and connects operators with uncoordinated rates. This type of data usually comes from a continuous data source and can be used many times or written over before use, depending on the rate of its input and use.

Real-time applications motivate the timing and non-procedural control constraints of PSDL. Each time critical operator has a *maximum execution time* constraint, representing the maximum time the operator may need to complete execution after it is fired, given access to all required resources. In addition, each periodic operator has a *period* and a *deadline*. The period is the interval between triggering times for the operator and the deadline is the maximum duration from the triggering of the operator to the completion of its operation.

Each sporadic operator has a *maximum response time* and a *minimum calling period*. The minimum calling period is the smallest interval allowed between two successive triggerings of a sporadic operator. The maximum response time is the maximum duration allowed from the triggering of the sporadic operator to the completion of its operation. To model distributed systems, PSDL also provides the option of specifying the *maximum delay* associated with any data stream. Control constraints define data-triggered actions and provide conditional execution and output.

The CAPS Tools

The set of tools provided by CAPS includes the user interface, software database system, and execution support system.

The user interface in CAPS includes a graphic editor, a syntax-directed editor, and a browser. The graphic editor and the syntax-directed editor together provide a user-friendly environment for the user/software engineer to construct a prototype using graphical and textual objects. The browser allows software engineers to view reusable components in the software database system.

The software database system, which consists of a software base and a design database, provides facilities for software reusability, automated system management, and version control. The software base keeps track of the PSDL descriptions and Ada implementations for all reusable software components in CAPS. The design database coordinates the concurrent efforts of a team of software engineers and manages the different versions and alternatives of the design and documents they produce.

The execution support system consists of a translator, a static scheduler and a dynamic scheduler. The translator generates code that binds together the code supplied by the designer and the reusable components extracted from the software base. The static scheduler and the dynamic scheduler together create the real-time schedule and drivers needed for executing the prototype.

Application of CAPS in Real-time Software Development and Acquisition

Rapid prototyping under CAPS is a semi-formal approach to real-time software development. Like Structured Analysis[6], it uses a structured approach to enforce modular design. However, unlike Structured Analysis, CAPS provides tools which span the entire software development process, ranging from requirements analysis to system testing and integration.

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[4]. CAPS provides the necessary means to bridge the communication gap between the customers and developers. The Prototype System Description Language is designed specifically for specifying hard real-time systems. 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 graphic and syntax-directed editors 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. CAPS offers basic requirements traceability through the "by requirements" statement of PSDL, allowing software engineers to link actual requirements to the modules that realize the requirements. Requirement traceability is essential in the development of hard real-time systems, where user requirements change often over the course of the software development.

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 the completion of all the subsystems by combining the final versions of the completed subsystems with prototype versions of the parts that are still being developed.

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 reduce the risk of contracting for construction of systems that do not meet the sponsors' needs. The validated high-level designs resulting from prototyping can be used to formulate subcontracts for subsystems that prevent many system integration problems. Other problems can be detected earlier by testing interactions between delivered subsystems and prototypes of subsystems not yet complete. Prototypes generated by CAPS include structures for experimentally measuring whether critical components meet their timing requirements. These structures can be used to evaluate real-time performance of systems delivered by contractors. Automated functional testing can compare results or delivered systems to results produced by the prototype.

Conclusions

The capabilities provided by CAPS are essential to rapid prototyping. In particular, automatic code generation and instrumentation let designers to try design variations quickly without cutting corners. The diagnostics provided by the automatic instrumentation helps to localize and fix bugs. Automated schedule construction and diagnostic information about timing constraints helps to navigate the maze of interacting resource constraints and evaluate the feasibility of the requirements.

We have recently completed an experiment to evaluate our rapid-prototyping methods and computer-aided design environment[7]. Our experiment was to prototype a generic command, control, communications and intelligence (C3I) station and generate the Ada code from the prototype's specifications automatically. The results of the experiment show that it is feasible to use computer-aided prototyping for practical, real-time Ada applications. Experience gained from the experiment also suggested many improvements to CAPS. Our long term plans include work on extending CAPS to a distributed computing environment, improved computer aid for managing and retrieving reusable software, and computer-aided generation and optimization of production-quality implementation code from specifications.

CAPS is unique in providing a computer-aided prototyping capability for hard real-time systems. Current CASE tools only support the static aspects of software development, while CAPS supports the dynamic aspects of the software life cycle. It will help the contracting and acquisition process for hard real-time systems in establishing that the initial requirements correctly capture the problem, checking that delivered systems meet their requirements, and ensuring that independently developed subsystems work properly when integrated together.

The substantial investment in real-time embedded systems in the Department of Defense (DoD) and the Department of Navy (DoN) points out the tremendous need for improved acquisition methods and supporting tools that can be used to test the design of a software system before coding. The Military Specification MIL-H-48655B calls for requirements analysis, functional specification and verification before the actual coding of the system[8]. The work described in this paper supports these objectives and enables DoN to make better decisions in acquiring systems useful to the Navy.

Biography

Luqi is an associate professor of computer science at the Naval Postgraduate School. She has also worked on software research and development in academia and industry. Her recent work includes rapid prototyping, real-time languages, design methodology, and software development tools. She has received an Engineering Initiation Award and a Presidential Young Investigator Award from NSF. Her research has been supported by ONR, AFSOR, CNO, NAVSEA, Navy Labs and computer industry. Luqi has a BS in computational mathematics from Jilin University, China, and an MS and a PhD in computer science from the University of Minnesota.

Man-Tak Shing is an associate professor of computer science at the Naval Postgraduate School. He has also been on the faculty of the University of California at Santa Barbara. His recent work includes path planning, genetic algorithms, and scheduling real-time systems. Shing received his BS degree from the Chinese University of Hong Kong in 1976, and MS and PhD degrees from the University of California at San Diego in 1978 and 1980.

Acknowledgment

The research reported in this paper was sponsored in part by the Computer System Division of the Office of Naval Research through the direct funding program.

References

1. J.A. Stankovic and K. Ramamritham, *Tutorial on Hard Real-Time Systems*, IEEE Computer Society Press, Washington, D.C., 1988.
2. Department of Navy, SECNAV INSTRUCTION 5200.37, "Acquisition of Software-Intensive C2 Information Systems," 5 January, 1988.
3. Luqi, "Software Evolution via Rapid Prototyping," *Computer*, vol. 22, pp. 13-25, 1989.

4. Luqi, R. Steigerwald, G. Hughes, F. Naveda, and V. Berzins, "CAPS as Requirements Engineering Tool," *Proc. Requirements Engineering and Analysis Workshop*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1991.

5. Luqi, V. Berzins, and R.T. Yeh, "Prototyping Language for Real-Time Software," *IEEE Transactions on Software Engineering*, vol. 14, pp. 1409-1423, 1988.

6. D. Hatley and I. Pirbhai, *Strategies for Real-Time Systems*, Dorset House Publishing Co., New York, NY, 1988.

7. Luqi, "Computer-Aided Prototyping for Command-And-Control Systems using CAPS," *IEEE Software*, pp. 56-67, Jan 1992.

8. Department of Defense, Military Specification MIL-H-48655B, "Human Engineering Requirements for Military Systems, Equipment and Facilities," 31 January 1979.