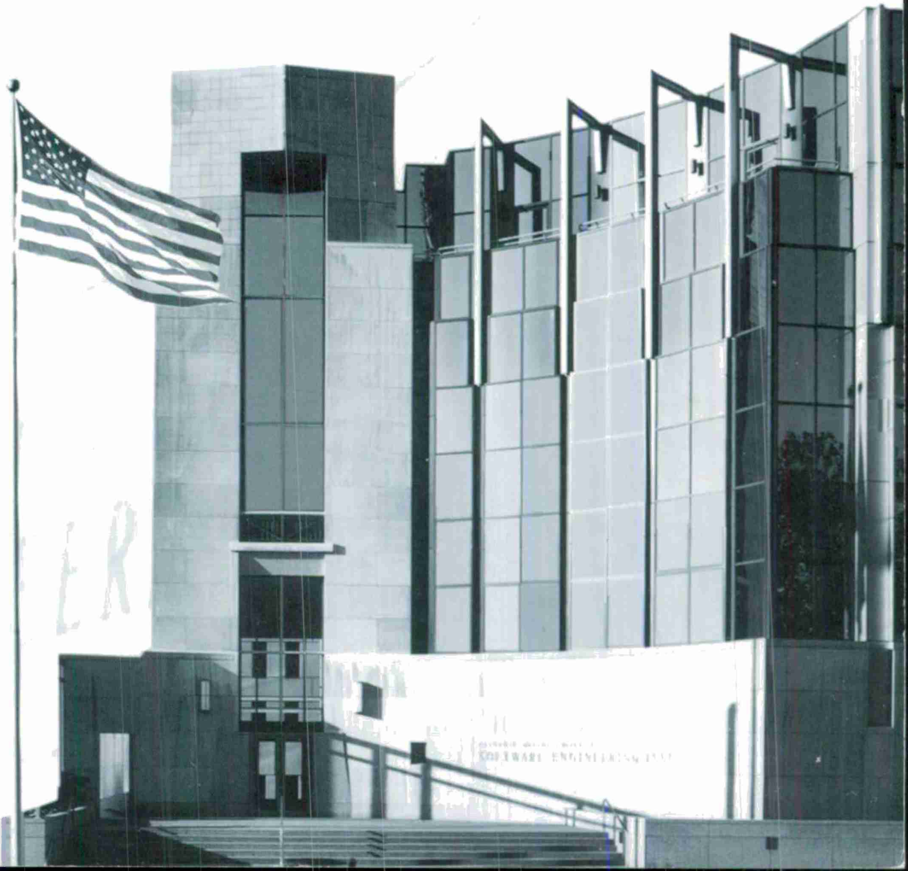
Carnegie Mellon
Software Engineering Institute

Improving Predictability in Embedded Real-Time Systems

Peter H. Feiler, Software Engineering Institute
Bruce Lewis, U.S. Army Aviation and Missile Command
Steve Vestal, Honeywell Technology Center

December 2000

SPECIAL REPORT
CMU/SEI-2000-SR-011
ADA387262



Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Improving Predictability in Embedded Real-Time Systems

CMU/SEI-2000-SR-011

Peter H. Feiler, Software Engineering Institute
Bruce Lewis, U.S. Army Aviation and Missile Command
Steve Vestal, Honeywell Technology Center

December 2000

Dynamic Systems Program

This work was sponsored by the Defense Advanced Research Projects Agency and by the U.S. Army, as well as by the Office of the Under Secretary of Defense's Open System Joint Task Force through Curtis Royster and Col. Glen Logan.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Table of Contents

List of Figures

Abstract

This paper discusses a model-based architectural approach for improving predictability of performance in embedded real-time systems. This approach utilizes automated analysis of task and communication architectures to provide insight into schedulability and reliability during design. Automatic generation of a runtime executive that performs task dispatching and inter-task communication eliminates manual coding errors and results in a system that satisfies the specified execution behavior. The MetaH language and toolset supports this model-based approach. MetaH has been used by the U.S. Army in a pilot project applied to missile guidance systems. Reduced time and cost benefits that have been observed will be discussed as a case study. The paper closes by outlining the current state of commercial availability of such technology and efforts to develop standards, such as those put forth by the Society of Automotive Engineers (SAE); Avionics Systems Division (ASD); working group on Avionics Architecture Description Language (AADL); and the Object Management Group (OMG) Unified Modeling Language (UML) working group on real-time and performance support in UML.

1 Introduction

Embedded real-time systems are used today in many mission critical settings. Since they play a crucial role they are developed through careful analysis and stringent tests. These systems are evolving from federated systems to integrated systems and are becoming components of larger and more complex systems. Examples include unmanned air vehicles, autonomous robots with complex missions, and integrated automotive control.

In addition, systems are deployed using technology whose half-life is three years or less, and in operational environments that are constantly changing. Not only do we need to scale the analysis capabilities to handle increasing complexity, but also we need to accommodate increasingly rapid change in capability and technology even in the domain of embedded real-time systems.

In this paper we present a model-based architectural approach retaining the predictability of real-time systems in terms of schedulability and reliability while accommodating an increasing degree of flexibility to support rapid evolution. We first elaborate on the need for predictability, followed by a discussion of the key concept of this engineering approach—the use of architectural models as a key element of software development. We proceed by describing MetaH, an architecture description language and supporting toolset for embedded real-time systems, as an example of technology for this approach. This is followed by a case study of the deployment of this approach and supporting technology in a pilot project at the U.S. Army. The paper closes by summarizing current activities in the software engineering community that offer viable commercial methods and tools and supporting standards.

2 Predictable Real-Time System Evolution

The performance and reliability of time-sensitive systems depends significantly on the execution environment (compilers, operating systems, processors, buses, I/O devices). It is often very expensive to rehost such systems when computing capacity is exceeded or the hardware becomes obsolete. Embedded real-time software is particularly difficult to rehost because of 1) its tailoring and optimization to fit the limited resource footprint of the hardware and 2) the need to support specialized device interfaces. Avionics and flight control software adds to the complexity by requiring multilevel safety, fault tolerance, modular multiprocessor architectures, and very complex multi-mode system behavior.

Because of the complexity of upgrading the software for a new processing environment, one of the most significant risks in system development of large real-time systems, especially avionics and flight control systems, is the problem of exceeding the computational resources during the software development process and during the operational lifetime of the system. Program after program has had to scale back system requirements to fit on the hardware. Integration, maintenance, and upgrade costs are driven up since software must be shoehorned into the available resources for as long as possible.

In addition, the execution capacity of many systems is not well understood. The software system design and analysis techniques often used provide limited quantitative indication of schedulability bounds and performance limitations early in the life cycle. Furthermore, the impact of system changes on available resources, real-time performance, and reliability is often not understood. Even small changes can result in unexpected and difficult-to-resolve failures. Eventually, these changes exceed the capacity of the system.

In this age of commercial off-the-shelf (COTS) processors, and with the very rapid increase in power of those processors, finding a higher performing processor is often not the problem. Again, the greater difficulty is in moving the software onto a new execution platform.

The software portability problem also manifests itself in fielded systems. Military mission critical weapons and aircraft systems typically have very long lives and must be upgraded throughout their life cycle. Capacity on the original processors is soon exhausted, if it's not already exhausted when fielded. Multiple processors become obsolete within the lifetime of these systems. Millions of dollars and years of effort were spent to upgrade or re-develop the software each time.

3 Model-Based Engineering Approach

Many development projects today use computers to develop and maintain their documents. However, the software development process still imitates a manual, paper-intensive process, where developers work on design after reading requirements documentation. Similarly, code is produced manually from design documentation. This introduces opportunity for errors.

Even in projects that deploy tools to support detailed design, architectural design typically is expressed as box-and-arrows charts; accompanying text specifies expected system behavior and system quality attributes such as performance and reliability. As detailed design and implementation approaches, the system is divided into computer software configuration items (CSCI) that are developed independently. Less and less architectural context information is available. When integration time comes, pieces do not always fit. If the development process has poor interface control, they may not fit functionally. If quality attributes such as performance are not well documented and are not analyzed repeatedly, system behavior in terms of these quality attributes may not be satisfactory when the system is integrated for the first time or upgraded. This is illustrated in Figure 1.

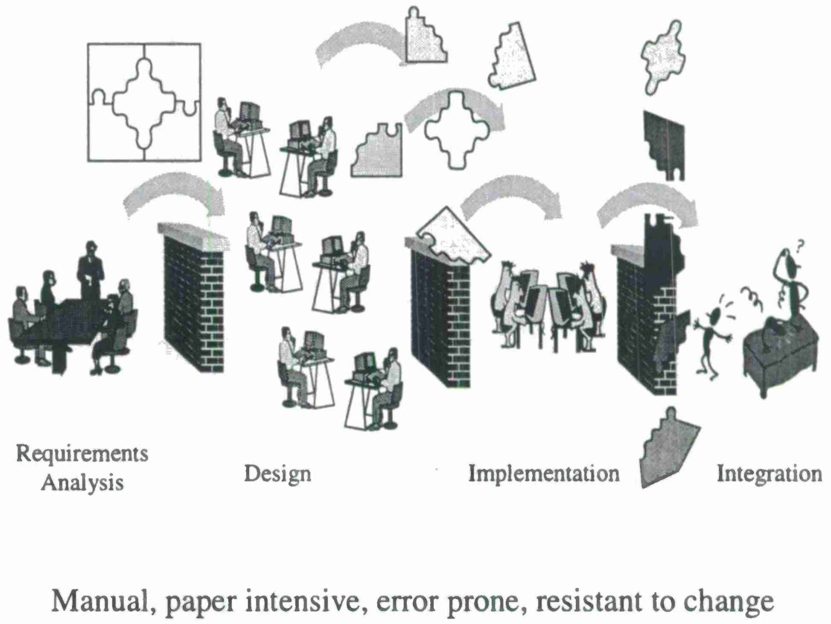


Figure 1: Current Software Process

Integrated Project Teams alleviate some of the communication problems in this "Over-The-Wall" approach, but still retain the problems inherent in human interpretation and translation of documents. Although evaluations of architecture may occur with requirements modeling tools and simulations, the results are reduced again to paper for impact on the final system software. Modeling results tend to be disconnected from the next phase and from each other. Multiple complex modeling languages are required, one for each system analysis area. Integration of components into a system is manual, often difficult, complex, and very expensive. Code generation for system or component analysis is for prototyping; requirements are again specified for human development of a traceable, testable integrated system.

In a model-based engineering process the architecture of a system is made explicit and is visible throughout the development process (see Figure 2). The architecture is the basis for an engineering model that allows for repeated analysis of the system from various perspectives, starting early in the life cycle. The architectural model evolves with the system – being a key element of the system development. As a result, the impact of changes to a system on system-wide quality attributes can be quickly validated through re-analysis, based on the architectural model. System integration is performed more smoothly as interface inconsistencies can be identified early, as well as inconsistencies in various critical quality attributes of the system.

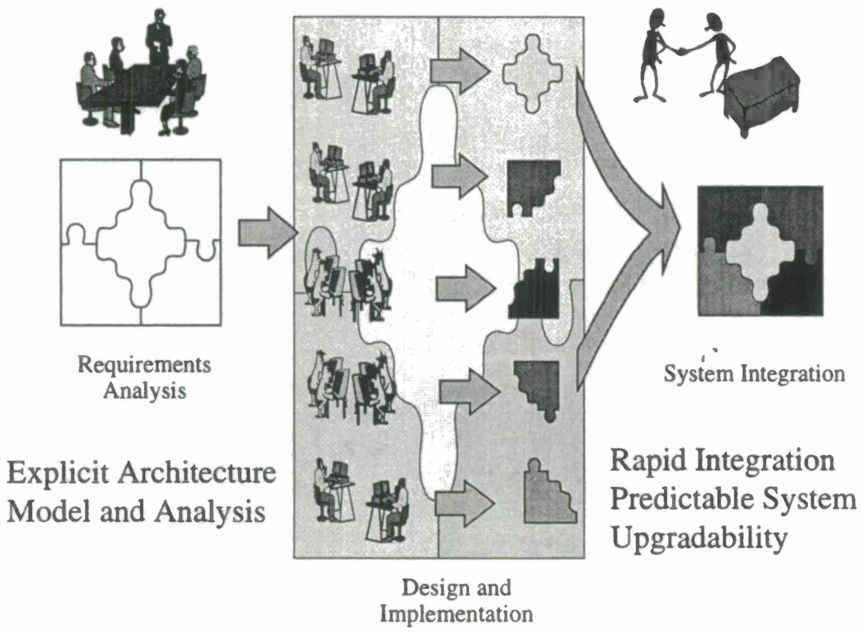


Figure 2: Architectural Engineering Process

This new paradigm is based on the ability to specify a real-time system architecture in terms of software and hardware components and their interfaces, the system execution behavior, and its quality attributes. This architectural model is the basis for analyzing the system's properties and automatically building the system. This is illustrated in Figure 3. First the architecture specification is used to model and analyze schedulability, reliability (fault handling), and safety/security dependencies. These issues must be understood early in time- and safety-critical systems. Once the systems engineer is satisfied with the architecture, the components can be developed, reused from another project, or generated in parallel with incremental automated integration of the system. The system is easily re-integrated through regeneration from the specification. Early integrations may be on a workstation, where behavior and system output can be validated. The final system is automatically integrated from the specification and components, hardware and software, on the target platform where execution behavior and results can again be validated.

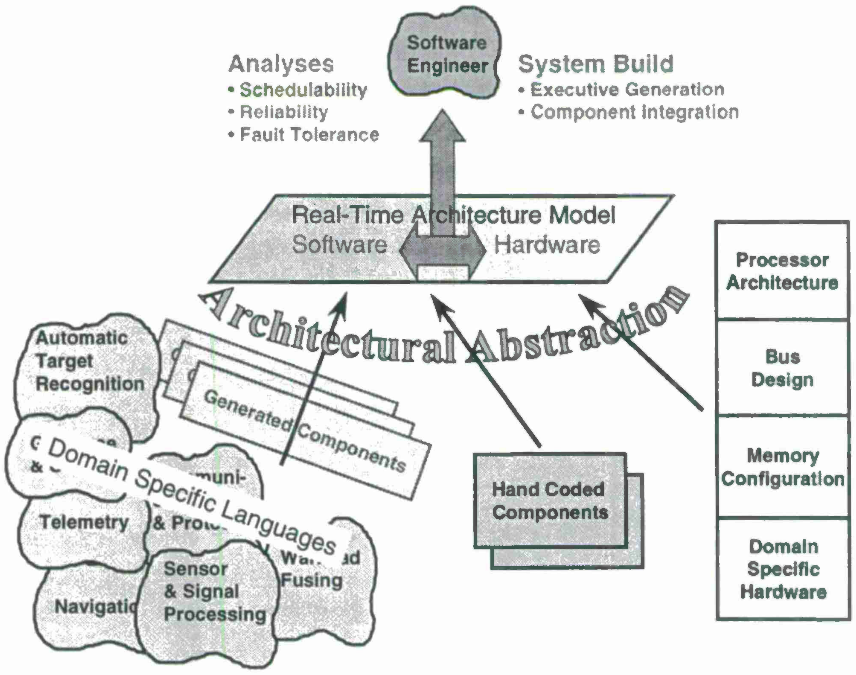


Figure 3: Model-Based Real-Time System Engineering

A major benefit is that the specified architecture and execution behavior are captured, not on paper, in the heads of the designers, or in scattered databases, but in one specification that integrates the final system and generates the executive that drives its execution. Also, a single architectural specification is used for multiple formal analyses; therefore the system is generated compliant, with each of the models used for analysis.

Changes can be quickly made at the specification level for load balancing, scaling, timing, message passing, shared data, new components, adding fault response modes, etc. Since the processor, buses, or other hardware devices are part of the architecture specification, they can quickly be changed to any from a user-expandable library. Hardware dependencies reside in the specification and toolset rather than the application code, allowing rapid ports to new environments.

This model-based engineering approach can not only be used in new system development, but also leveraged in maintenance of existing systems. Architectural modeling and analysis can provide and document system insight. Based on this insight, the legacy system may be reengineered to fit within the architectural model. The investment into such an effort has high payoff, as the architectural description is reused in future maintenance activities. The U.S. Army case study in Section 5 illustrates the use of this engineering approach in an existing system.

4 MetaH: Model-Based Engineering for Real-Time Systems

MetaH is an architecture description language originally intended for use in Avionics applications [Honeywell 98]. Specifically, it supports the description, analysis, and generation of task and communication architectures of embedded real-time system applications. The MetaH notation allows developers to describe an application in terms of tasks, task communication, operational modes, and composition of tasks in terms of software components, hardware, and mapping of the software system onto the hardware [Binns 93]. Software components themselves may have been developed by hand or by domain-specific application generators such as SimuLink. The notation currently emphasizes support for processing of continuous data streams such as continuous control applications, with limited support for discrete event systems.

The MetaH toolset provides

- a graphical editor to create and maintain architectural models
- a suite of analysis tools including a schedulability analysis tool based on Generalized Rate Monotonic Analysis (GRMA); a reliability analysis tool to determine the probability of failure of a system subjected to randomly arriving faults in terms of a stochastic finite state reliability model; and a safety analysis tool to investigate the potential of impact between system components of different safety levels
- a generation and build capability that includes a code generator for all task dispatch and communication code in form of a MetaH executive; a system builder that combines user-supplied components with the generated task and communication calls; and the runtime kernel, i.e., real-time operating system, supporting the execution of the application

One key to successful embedded systems is a layered runtime architecture that supports partitioning—as illustrated in Figure 4. The major driver for partitioning is the dramatic reduction in initial and upgrade validation and verification (V&V) effort that can be achieved. Partitioning methods have been fielded and their use is spreading rapidly for civil aviation. The use of partitioning methods to reduce certification effort is recognized in the Radio Technical Commission for Aeronautics (RTCA) DO-178B standard, in several Aeronautical Radio, Inc. (ARINC) standards, and by the U.S. Federal Aviation Administration (FAA) and European Joint Aviation Authorities (JAA).

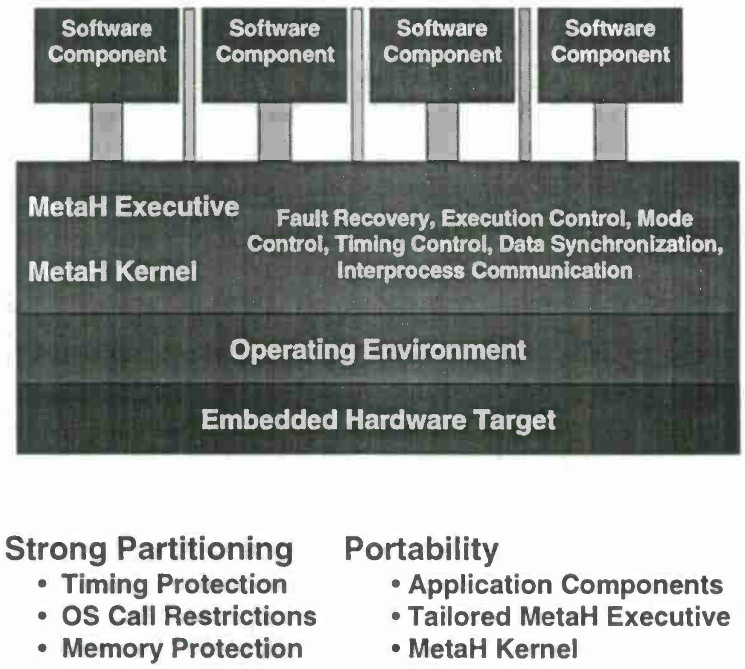


Figure 4: A Partitioned Layered Runtime Architecture

The layered runtime architecture facilitates portability in the following ways. Auto generation allows for tailoring of the MetaH executive. The MetaH kernel is portable through use of Ada95 and IEEE POSIX (portable operating interface standard) application programming interface (API). Timing protection enforces timing constraints at runtime. Their enforcement ensures validity of analysis results; i.e., a misbehaving process cannot encroach on the resources granted to another process. Applications are restricted from use of operating systems functions that are key to maintaining integrity established through the MetaH executive and kernel. Memory protection assures the safety of one component from misbehavior of other components by preventing access to private memory spaces.

The technology behind the MetaH stems from research started at Honeywell Technology Center in the late 1980's. Through funding from Honeywell and two Defense Research Project Agency (DARPA) programs—Domain Specific Software Architectures (DSSA) and Evolutionary Design of Complex Systems (EDCS)—the technology has matured from proof of concept to a notation and toolset that has found its way into actual use in various pilot projects within Honeywell and other organizations. The U.S. Army Aviation and Missile Command (AMCOM), Research Development and Engineering Center, Software Engineering Directorate (SED) has performed laboratory demonstrations and technology integration with MetaH since 1993.

5 U.S. Army Case Study

This case study describes a pilot application of the MetaH technology by the U.S. Army AMCOM SED laboratory to missile guidance systems. An existing missile guidance system, implemented in Jovial, was reengineered to run on a new hardware platform and to fit into a generic missile reference architecture [McConnell 96]. As part of the reengineering effort the system was modularized and translated into Ada95. The task architecture consisting of 12-16 concurrent tasks was represented as a MetaH model and the implementation generated automatically from the MetaH model and the Ada95 coded application components. The resulting system consisted of 12,000 source lines of application component code, 3000 lines of MetaH executive generated from the MetaH model, and 3000 lines of code representing MetaH kernel services. The engineers doing the reengineering work made a conservative estimate of effort required to reengineer the system into a pure Ada95 implementation and validated the estimate with the prime contractor who implemented the missile.

After the initial port into Ada95 and MetaH, the application was ported several more times to new hardware platforms as processor technology evolution continued its fast pace. These ports included multiple ports to single and dual processor implementations of the initial target hardware, as well as new processors, compilers, and O/S. In these successive ports the executables performed correctly on each target environment the first time.

Figure 5 illustrates a comparison of effort in reengineering the application into Ada95 and performing the ports under the traditional approach (i.e., implementation in Ada95 vs. Ada95 components and a MetaH task and communication model). In both the traditional approach and the MetaH approach, a JOVIAL-based application is translated into Ada95, then ported to a new target platform.

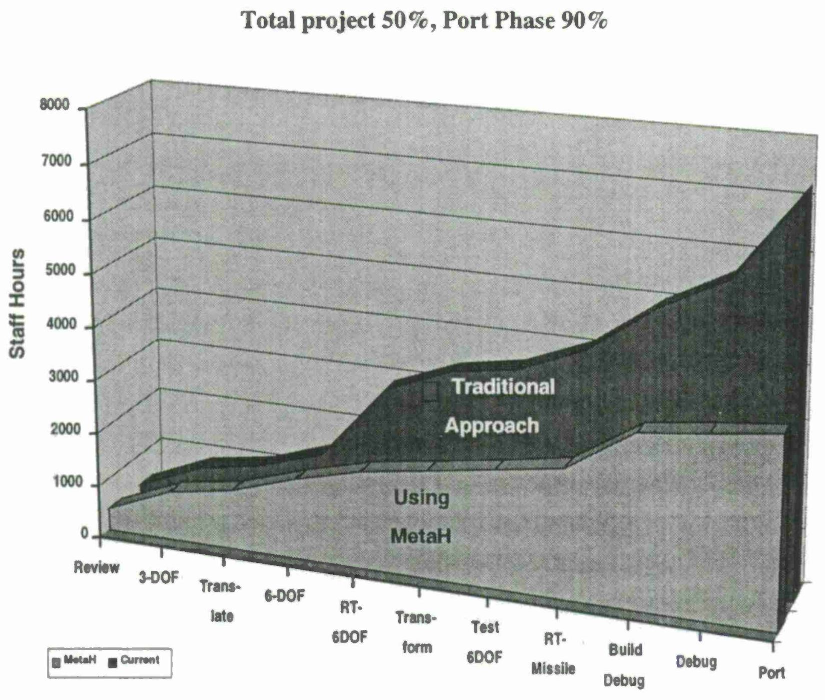


Figure 5: Effort Comparison: Traditional vs. MetaH Based

The cost of the translation process from JOVIAL to Ada95 was similar in both efforts. In the case of the MetaH effort it includes the cost of developing the MetaH specification. In the initial reengineering effort, the MetaH approach shows payoff in the phases that address the real-time behavior of the application (RT-6DOF, Transform, RT-Missile) and in integration test (Debug).

The cost of the port to a new target platform (see the last step in the graph above) demonstrates the benefit of performing an application port to a new platform based on MetaH. In the case of the traditional approach, developers had to be concerned with code throughout the system. In the MetaH approach, time critical behavior was reproduced through regeneration of the system executive (including creation of new messaging across processors for the new processing environment) code relevant to the port was localized, performance analysis early in the port cycle allowed verification of schedulability, and components were automatically re-integrated. Missile application reengineering including a port to a new processor at the end of the project resulted in a 50% cost reduction over the non-MetaH approach.

MetaH-based ports to a new platform resulted in even more impressive cost savings:

- ***Application and MetaH kernel port.*** Using standards based ports, the user should be able to port in four weeks (160 hours) with time to debug the environment and do performance tuning. Some ports require only a week. Use of MetaH resulted in a ***ratio of 10 to 1*** reduction.

- ***Application port to existing MetaH kernel***. Only a few hours were required to rebuild. Estimated time to port these applications without MetaH is nine months. There was a ***ratio of 60 to 1*** reduction.

Latter examples include

- ***Single processor Pentium/Aonix/Pharlap target*** took ***90 hours*** including some bug workarounds and performance tuning. Honeywell developed and supplied the target. It took ***24 hours*** to come up to speed with the tools and get application code running. The missile flew correctly the first time executed.
- ***Multi-processor Pentium/Aonix/Pharlap target*** added an additional ***75 hours*** of Honeywell labor; we discovered bug in Tundra chip impacting multi-processor control. It took ***128 hours*** to find, fix, and implement. The missile flew correctly when executed.
- ***Workstation Pentium/GNAT Ada95/NT target*** was available as part of toolset. It took 45 minutes to build with MetaH, compile, link, and execute. However, NT would not let the support applications (which collected data) run. So we spent two hours creating a clock interface to the missile so data collection could run. We validated correct flight dynamics with flight software-in-the-loop, non-real time. Total time was less than ***three hours*** to port application with predefined target.
- ***PowerPC/Green Hills/VxWorks target*** was developed by the user organization, SED. Total time to install, learn, and port the MetaH toolset to the environment, and then fly, was ***36 hours.*** MetaH correctly constructed the application and it flew correctly the first flight.

These are imprecise but expert estimates provided by the engineer who did the ports in MetaH and has been doing ports for hardware-in-the-loop environments for real-time systems for 18 years.

6 Becoming a Reality

The U.S. Navy, U.S. Air Force, the Ada Joint Program Office, and the U.S. Army Space and Missile Defense Command have also funded MetaH related projects. The Open Systems - Joint Task Force (OS-JTF) has funded projects using MetaH's advanced system building capabilities for modular avionics to evaluate the IEEE POSIX API and to impact the Society of Automotive Engineers (SAE) Avionics Systems Division Embedded Computing Systems committee (AS-5) Generic Open Architecture (GOA) and OS API standards efforts. OS-JTF is currently supporting the standardization of an Avionics Architecture Description Language based on MetaH. The synergistic integration of advanced DARPA technology with industrial standards has resulted in the cost-effective portability of MetaH–based applications as demonstrated in the U.S. Army case study.

The MetaH is currently available from Honeywell under a no-fee license. A modified version of the language and toolset was included on a ground-based testing system supplied by NASA to International Space Station developers worldwide. To date MetaH has only been used for advanced development and demonstration projects. Therefore a discussion of the intended and possible uses for MetaH is somewhat speculative.

- Most of the studies, demonstrations, and technology exchanges that have involved MetaH have been for avionics applications, both civil and military. These include International Space Station ground-based test system, Boeing 777 flight management systems, business jet real-time operating systems, Lockheed-Martin Joint Strike Fighter vehicle control, and C-130 mission management system.
- Medical devices have requirements for high reliability and safety, and control systems for some medical devices may be well suited for MetaH. Initial discussion with Siemens Research Center and Guidant Corporation indicate that this market may require increased discrete-event control to complement continuous control support.
- Automotive control systems, particularly power train and braking systems, seem well suited for MetaH and have been investigated in discussions with Ford and Visteon. They have high performance requirements (control rates of several kilohertz, higher than most avionics systems), and stringent efficiency and size requirements due to high recurring hardware cost. Future drive-by-wire and brake-by-wire systems will have extremely high reliability and safety requirements.
- Robot control systems built using Honeywell MetaH tools have been demonstrated on simulators, though not yet in actual robots. Robot control systems combine a need for real-time execution and, in some markets, high reliability and assurance of safety. To date, work in this area consists of one demonstration program focused on unmanned ground vehicles.
- Engine control systems (jet, turbine, and automotive), seem well suited for MetaH. Like automotive applications, they have high requirements in performance, efficiency, reliabil-

ity, and assurance. Applied Dynamics International is developing an interface between the Beacon/MatLab Computer-Aided Control System Engineering (CACSE) toolset and MetaH.

Commercial toolsets aim to support design and development of real-time applications. They include ObjecTime (ObjecTime Ltd., Ontario, Canada)—now part of Rational Rose Real-Time (www.rational.com), ObjectGEODE (www.verilog.fr)—recently purchased by Telelogic, ControlShell (www.realtimeinnovations.com), Tau (www.telelogic.com), Real-Time Studio (www.artisansw.com), and Rhapsody (www.ilogix.com). Many of these products support design notations that have their roots in detailed design and emphasize discrete event modeling in terms of finite state machines. Primary application domains have been in the telecommunication arena. Capabilities for schedulability analysis are available by interfacing with real-time modeling and analysis toolsets such as TimeWiz (www.timesys.com) or RAPID (www.tripac.com).

This market is going through a consolidation in the form of mergers and by companies aligning their products with the Object Management Group (OMG) standard Unified Modeling Language (UML). In 1999 the OMG initiated an effort to define a UML profile for scheduling, performance, and time with all of the key players in the method and tool community participating [OMG 99]. In that context, an extension to UML itself to better support architectural modeling is being considered.

The Avionics community has also recognized the need for better modeling support for embedded real-time systems. In the Fall of 1998 the Society of Automotive Engineers (SAE) Aerospace Avionics Systems Division (ASD) Embedded Computing Systems committee (AS-5) initiated a working group to investigate the standardization of an Avionics Architecture Description Language (AADL) [www.sae.org/technicalcommittees/aasd.htm]. This community brings a strong avionics systems development perspective to the table. MetaH was chosen as a starting point of discussion. In 2000 this has become a full-fledged subcommittee. A requirements document has been balloted and work is starting on the language standard document itself. An effort is being made to align its evolution with progress in the standardization of real-time UML.

7 Summary

In this paper we have examined an approach for model-based engineering of embedded real-time systems. This approach leverages architectural modeling of real-time aspects of a system by supporting analysis of schedulability, performance, and reliability. The approach also supports automatic generation of runtime executives specific to the application, and system build of the complete system from developer-supplied components and the generated executive.

We have demonstrated the practicality of this approach in the context of MetaH, a real-time system architecture description language and supporting toolset for analysis and generation. A U.S. Army AMCOM case study has demonstrated the benefits of deploying such technology to existing systems. These benefits include system analysis and validation of non-functional properties, such as timing and performance, early in the life cycle; separation of concerns regarding functionality of the application and the real-time behavior in terms of task dispatching and communication; and automatic generation of executive code from the model against commercial and standard runtime environments, such as IEEE POSIX conformant real-time operating systems or language runtime systems such as Ada95. This has resulted in a major reduction in cost for porting embedded applications to new hardware configurations and platforms.

The software engineering community has recognized the need for model-based engineering of real-time systems and has initiated standards efforts through SAE and OMG to put the appropriate technology infrastructure in place.

References

[Binns 93]	Binns, Pam & Vestal, Steve. "Scheduling and Communication in MetaH," *IEEE Real-Time Systems Symposium*. Raleigh-Durham NC, December 1993.
[Honeywell 98]	Honeywell, Inc. *MetaH Product Information.* Available URL:<http://www.htc.honeywell.com/metah/prodinfo.html>
[McConnell 96]	McConnell, David J.; Lewis, Bruce; & Grey, Lisa. "Re-engineering a Single Threaded Embedded Missile Application Onto a Parallel Processing Platform Using MetaH," *Proceedings of 4th International Workshop on Parallel and Distributed Real-Time Systems*. Honolulu, HI, April 1996.
[OMG 99]	The Object Management Group. *RFP: UML Profile for Scheduling, Performance, and Time.* (OMG Document ad/99-03-13) Framing ham, MA: March 1999. Available URL:<http://www.omg.org/cgi-bin/doc?ad/99-03-13.pdf>

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE December 2000	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE Improving Predictability in Embedded Real-Time Systems	5. FUNDING NUMBERS F19628-00-C-0003
6. AUTHOR(S) Peter H. Feiler, Bruce Lewis, Steve Vestal	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2000-SR-011
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES	
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS	12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This paper discusses a model-based architectural approach for improving predictability of performance in embedded real-time systems. This approach utilizes automated analysis of task and communication architectures to provide insight into schedulability and reliability during design. Automatic generation of a runtime executive that performs task dispatching and inter-task communication eliminates manual coding errors and results in a system that satisfies the specified execution behavior. The MetaH language and toolset supports this model-based approach. MetaH has been used by the U.S Army in a pilot project applied to missile guidance systems. Reduced time and cost benefits that have been observed will be discussed as a case study. The paper closes by outlining the current state of commercial availability of such technology and efforts to develop standards, such as those put forth by the Society of Automotive Engineers (SAE); Avionics Systems Division (ASD); working group on Avionics Architecture Description Language (AADL); and the Object Management Group (OMG) Unified Modeling Language (UML) working group on real-time and performance support in UML.

14. SUBJECT TERMS model-based architectural approach, improving predictability, real-time systems, MetaH	15. NUMBER OF PAGES 20
16. PRICE CODE	

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

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102