[1]Improving Predictability in Real Time Avionics and Space Systems

Bruce Lewis[(1)], Peter Feiler[(2)], Steve Vestal[(3)], Christophe Guettier[(4)]

[(1)] *U.S. Army Aviation and Missile Com, Attn:AMSAM-RD-BA-AT, Redstone Arsenal, AL., 35898, USA, Email:Bruce.Lewis@sed.redstone.army.mil*
[(2)] *Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA, Email:phf@sei.cmu.edu*
[(3)] *Honeywell Labs, 3660 Technology Drive,* Minneapolis, MN 55418-1006, *USA Email:vestal@htc.honeywell.com*
[(4)] *Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto CA 94304, USA Email:guettier@parc.xerox.com*

ABSTRACT

This abstract 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 in a demo projects applied to missile guidance systems and spacecraft attitude control. Reduced time and cost benefits observed will be discussed as a case study.

The Society of Automotive Engineers (SAE); Avionics Systems Division (ASD); working group on Avionics Architecture Description Language (AADL) is using MetaH as a baseline capability to develop an international standard avionics architecture description language. Space is a domain with similar requirements. A joint research project is being considered in combination with constraint programming technology from Axlog and systems engineering technology from INRIA (Dr. Gerard Le Lann, Proof Based System Engineering) of France which should provide additional system engineering capability of interest in the space domain.

CREDITS

This paper has been adapted from a special report from the Software Engineering Institute published under CMU/SEI-2000-SR-011. The technology has been developed with the support of the US Defense Advanced Research Projects Agency, US Army, and the Open Systems Joint Projects Office.

1. NEED FOR PREDICTABLE REAL-TIME SYSTEMS DEVELOPMENT AND 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.

[1] In the proceedings of the Eurospace conference on Data System in Aerospace, Nice, France, June 2001

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.

2. 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.

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. 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.

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. 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 re-generation 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.

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.

3. METAH, THE MODEL BASED ARCHITECTURE DESCRIPTION LANGUAGE

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. 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).

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.

4. POSSIBLE EXTENSION FOR FEASIBILITY ANALYSIS

The MetaH toolset provides a capability to automatically load balance processors, buses across modes of operations for processes that are not specified to execute on a specific processor. However, a constraint programming approach provides a more flexible approach for feasibility analysis and selection of best alternatives. Instead of experimenting with values and simulating an important search space, the designer needs a more powerful expression and solving approach. This objective requires solving simultaneously several related problems such as mapping the tasks set to physical processors (which is NP-hard), as well as satisfying feasibility conditions of scheduling policies. In the case of the timeliness property, these conditions model the satisfaction of real-time constraints based on Higher Priority First (HPF) or Earliest Deadline First (EDF) policies. Hence, finding a feasible solution satisfying real-time constraints and processing elements resources generally requires problem-solving techniques. Stemming from Logic Programming, Integer and Mathematical Programming, Constraint Programming (CP) [Jaffar & Lassez 87] approaches are recognized to be powerful tools to cope with difficult and large combinatorial problems. The efficiency of the approach to model and solve mapping problems has already exhibited significant results in the Digital Signal Processing area, despite the numerous non-linear constraints [Guettier 97]. Following the same approach, CP provides various ways to model complex, dynamic, real-time systems in order to propose automatically design choices and architectural size. The global problem design can be expressed using several constraint-based models. Composed with mathematical variables and algebraic constraints, models represent invariants of sub-problems like schedulability conditions or processor allocation. Relations between models are conjunctions of constraints that maintain the consistency of the global solution. Using CP techniques, models are derived into concurrent search spaces. Each time the solving progress in one of the search space, the partial solution is propagated to other ones using models relations. Therefore, by maintaining arc-consistency, the CP system cuts other search spaces such that a global solution can be reach faster.

The workload distribution of the tasks set can mathematically be represented using set partitioning constraints:

$$T = \bigcup_{i=1}^{n} S_i, \quad \forall\, i, j, i \neq j, S_i \cap S_j = \{\ \}, i \in \wp$$

Where T (n=card(T)) is the set of tasks, and each S_i is a subset of T associated to the i[th] processor of the system ($\wp$ is the set of available processors). The first constraint states that all the tasks are completely distributed, while the second one states that tasks cannot be replicated.

Preemptive scheduling policies fit very well coarse grain task scheduling, with periodic/sporadic activation periods, satisfying timeliness property. On a practical viewpoint, when deadlines are assimilated to periods, modeling schedulability using CP techniques is fairly simple and is sufficient to illustrate our global solving approach. To tackle more complex assumptions (when deadlines are different from periods), a CP approach can take advantage of a convex schedulable domain for EDF, opposed to the HPF, for which the domain can not be easily expressed. Let us consider a periodic non-concrete traffic T, represented by a set of n periodic and sporadic tasks t_i . An activation period (T_i) (equal to its deadline) and a worst-case execution time (C_i) are assimilated to each task. The well-known Liu &Layland necessary condition for scheduling a worst-case execution of the tasks using EDF or HPF can be given using the workload:

$$T \text{ is feasible with EDF, HPF} \Rightarrow \sum_{j=1}^{n} \frac{C_j}{T_j} \leq 1$$

The associated prototype is developed upon Sicstus Prolog that encapsulates the state-of-the art in constraint propagation algorithms. Those algorithms are equivalent to logical proof methods, but where predicates can be constraints interpreted in a mathematical algebra. Thus, the proof algorithm can interplay between a logical reasoning using Horn clauses and arithmetic reasoning. This leads to more important proof domain, a better management of the combinatory and a higher efficiency, resulting from important search pruning and constraints propagation.

The prototype has been experimented on spatial platform, aircraft avionic and autonomous undersea vehicles and has provided interesting results. Future works will extend this approach to more complex feasibility conditions, related to distributed scheduling problems with a-periodic activation laws, under real-time and reliability constraints.

4. MISSILE 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. Based on the results, we estimated a 40% savings on the total re-engineering effort. Most of the savings came in the building and debugging of the real-time environment simulation and the real-time missile flight code. Because the processing environment, dual 80960 processors, was very tight for both the missile code and the environment simulation code, we used extensively the scheduling analysis to break up the simulation into rates that would meet the flight requirements but also be schedulable across the dual processors. The automated integration of components allowed rapid re-integration as we developed in an iterative fashion with more and more capability in each proven design. Iterations on the architecture were easily expressed and the system auto re-integrated by generation of the middleware and glue code. The capability to get timing data from the executing system and to run on both non-real-time and real-time environments with the same flight behavior was also very valuable for system tuning. Estimates from the missile prime were that we saved 66% of the effort based on their experience in similar activities.

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, timing and ordering preserved, on each target environment the first time we could execute on the new environment. This capability to port to a new target not only the application code but also its time sensitive qualities demonstrates an ability to do software first development and then port or evolve at significantly lower risk to new hardware. Our porting time was 1/10 of the expected time on average for Ada95 ports and increased the savings for the overall project (if a final port had been necessary) from 40% to 50%. POSIX ports would be more complex given the far greater variation of services provided in POSIX compliant O/Ss. Custom ports can also be complex since MetaH middleware generation must be mapped to custom O/S calls. However, once a port is working, rebuilding and tuning on the execution platform is very rapid. Glue code is rebuilt to the timing and architectural requirements in the MetaH specification. The MetaH Architectural Specification Language (ADL) is highly tunable for software and hardware architectural variation. MetaH hardware ports become part of the component library for future use.

SUMMARY

In this paper we have examined a highly predictable, flexible, approach based on model-based engineering for the development and evolution embedded real-time systems. This approach came from the avionics and flight control domain and is useful in the space domain. 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 system development, evolution and for porting embedded applications to new hardware configurations and platforms.

References:

[Binns 93] Binns, Pam & Vestal, Steve. "Scheduling and Communication in MetaH," *IEEE Real-Time Systems Symposium*. Raleigh-Durham

NC, December 1993.

[Guettier 97] C. Guettier, Global Optimization of Digital Signal Processing Application Mapping onto Parallel Architectures using Constraint Logic Programming, Thesis of the Ecole Superieure des Mines de Paris, Paris, December 1997

[Jaffar & Lassez 87] J. Jaffar and J-L. Lassez, Constraint Logic Programming, in Proc of the 14th ACM Symposium on Principles of Programming Languages, Munich, Jan. 1987

[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>