SOFTWARE PORTABILITY GAINS REALIZED WITH METAH AND ADA95

Bruce Lewis, US Army Aviation and Missile Command, Redstone Arsenal, Alabama

Abstract

MetaH is an Architecture Description Language (ADL) developed to express and evaluate the software architecture of avionics and flight control systems. It is intended for not only description and analysis, but also for integration of the software components on the specified embedded hardware. This automated composition to specification with glue code generation allows rapid development and evolution of real-time embedded mission and safety critical systems. It also provides very high portability in these complex systems for software application code across various execution environments. This is accomplished by leveraging language and O/S standards, minimizing component dependencies, and by constructing component timing relationships to specification across differing execution platforms.

This paper describes MetaH and provides the results of porting a highly time sensitive application across significantly different embedded hardware/software execution environments. We initially developed a reusable MetaH specification for missile architectures, populated it with software components reengineered from a production missile system, and executed it on single and dual i80960MC target configurations. We then retargeted this application to single and dual Pentium target configurations; and to a single PowerPC configuration. We compare the costs of these exercises with estimated costs to do the same tasks using traditional methods.

In each of these cases the high portability and supported functionality of Ada95 was a significant enabler. Current trends say that future avionics systems and perhaps other safety critical applications will require space and time partitioning. Should Ada95 and the Ravenscar profile support these capabilities?

Problem Statement

Due to significant dependency of software on the execution environment (compilers, operating systems, processors, buses, I/O devices), it is often very expensive to re-host software as execution capacity is exceeded. Embedded real-time software is particularly difficult to re-host because of timing dependencies, performance requirements, complex processor architectures, synchronous concurrent processes and 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 processor resources during the software development process. 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 shoe-horned into the available resources for as long as possible.

In addition, the execution capacity of many systems is not well understood. Avionics systems have long lives with periodic upgrades. The software system control techniques often used provide no quantitative indication of schedulability bounds or the impact of changes on the application. Even small changes can result in unexpected and difficult to resolve failures. Eventually, these changes exceed the capability 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 lifecycle. Capacity on the original processors is soon exhausted if its not already exhausted when fielded. Multiple processors become obsolete within the development phase of these systems with millions of dollars and years of effort spent to upgrade or re-develop the software each time. Many more processors and buses will become obsolete over the system lifetime costing many millions more and significantly delaying system capabilities. A much more evolvable approach that meets system requirements is needed.

History

The technology behind the MetaH ADL was developed over several DARPA programs. The first, Domain Specific Software Architectures (DSSA), was concerned with using domain specific system engineering knowledge to build languages (ADLs) that could specify software architectures and analyze architectural properties to prevent architectural problems from impacting development. MetaH leverages the rapid construction of systems further by adding the automatic integration of hardware and software in accord with modeling used to analyze the system. DSSA emphasized the use of domain specific application generators or reusable component libraries in concert with ADLs to build systems. The second, the Evolutionary Design of Complex Systems (EDCS), was focused on our ability to build systems that could be rapidly evolved and to predict the impact of change. EDCS started with multiple approaches but ended with a strong focus on ADLs as a foundation for building highly evolvable complex systems. The DARPA program Dynamic Assembly for System Adaptability, Dependability and Assurance (DASADA) is extending the impact of MetaH by addressing efficient dynamic and static scheduling required for efficient integration of soft real time dynamic applications (tactical internet) with hard real time (safety critical) applications to build more dynamic systems with retained dependability and assurance qualities. This also includes dependable adaptation through architectural constraint based dynamic reconfiguration and design time automated verification technology.

Dr. Steve Vestal of the Honeywell Technology Center has been the principal investigator. Bruce Lewis has served as technical POC on both DARPA programs and has led the US Army Aviation and Missile Command, Research Development and Engineering Center, Software Engineering Directorate (SED) laboratory demonstrations and technology integration with MetaH since 1993. The US Navy, US Air Force, the Ada Joint Program Office, and the US 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 POSIX API and to impact GOA and SAE OS API standards efforts. OS-JTF is currently supporting the standardization of an Avionics

Architecture Description Language (AADL) based on MetaH. The synergistic integration of advanced DARPA technology for evolvability, SED lab resources and OS-JTF open systems and standardization investigations has resulted in the advanced portability demonstrated in this paper.

An Overview of MetaH

In this DARPA research context, MetaH was developed for building missile and aircraft avionics and flight control systems. It was designed to integrate the multiple domains of application software in avionics on a generated architectural backplane based on formal scheduling and implementation methods. The MetaH language provides a system designer a simple but precise language for specification of architectural requirements from which it extracts the formal modeling parameters for multiple analyses. It comprehends both hardware and software components. It will generate the architecture integrating the hardware and software components into a system complaint with the modeled behavior. Current architectural analyses include schedulability, reliability and safety/security.

The language and toolset were developed to meet the requirements for building state-of-the-art modular multiprocessor systems with multilevel safety and security and fault management. It provides for the specification and generation of dynamic multi-mode behavior across multiple processors under the real-time constraints of flight control. It also can build from specification advanced space and time partitioned systems enabling very significant reductions (estimated at 80%) in re-validation and re-testing costs when changes are made to an avionics or mission critical system. The approach of using a combination of time and space partitioning is an emerging commercial avionics technology first fielded by Honeywell on the Boeing 777 and now part of several fielded commercial avionics systems. Space and time partitioning is also becoming important in military systems since military avionics are now required to be FAA certified if they fly in commercial airspace. With Glass Cockpits, avionics software becomes much more flight critical. Space and time (S&T) partitioning is also desirable for avionics and space systems to reduce the amount of hardware that must fly. These applications are also among those most likely to use Ada and where processing environment upgrades are likely to be extremely expensive.

MetaH was also developed to provide rapid development and evolution of the system. One aspect of evolution is the ability to rapidly reconfigure these complex real-time systems to new hardware environments. The MetaH process for system development offers low risk rapid changes in the execution environment. This fundamentally changes the high risk, tied-to-the-hardware development approach we currently use. Now, with MetaH, from a software/execution environment perspective, programs can evolve the hardware multiple times during system development and field with plenty of capacity using modern processors.

The MetaH Process

The combination of MetaH language and tools that analyze and implement the system to the architecture specified provides a new paradigm for the development of embedded real-time systems. The software architecture becomes a specified, analyzable entity in the design process and a generated layer in the implementation. The system becomes architecture based or architecture driven. Since the architecture of the final system is generated to the specification, it results in a system with known architectural properties and system execution behavior which can be rapidly evolved with predictable impact.

Figure 1 shows the current software development paradigm with its specification of requirements and design on paper. Integrated Project Teams alleviate some of the communication problems in this "Over-The-Wall" approach but the basic approach is specification for human interpretation and system construction. Evaluations of architecture may occur with requirements modeling tools and simulations but 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 and requirements are again specified for human development of a traceable, testable integrated system.

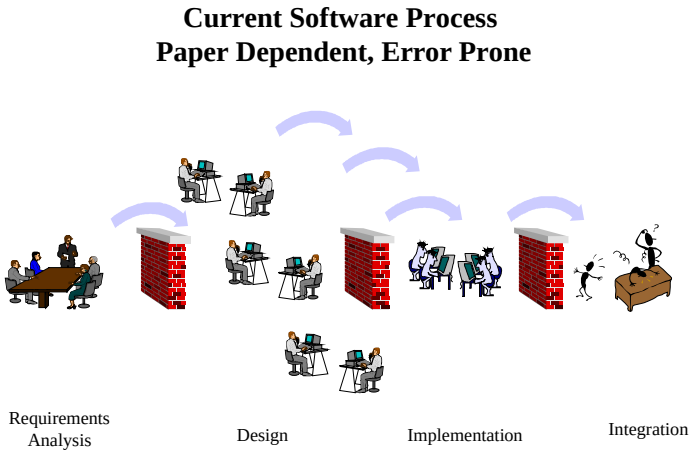

Figure 1

Figure 2 shows the new paradigm based on the ability to specify an architecture (consisting of software and hardware components, their interfaces and the system execution behavior), analyze its properties and then automatically build the system to the specification. First the architecture specification is used to model and analyze schedulability, reliability (fault handling), and safety/security dependencies. These issues need to be understood early in safety and time 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 with MetaH. 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.

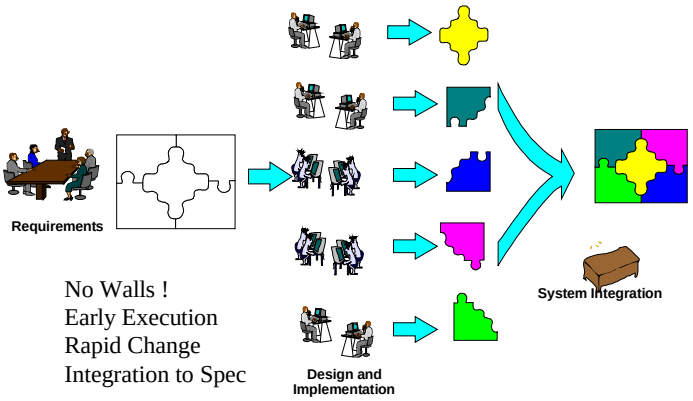

Figure 2

A major benefit is that the architecture and execution behavior specified is captured, not in paper or 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 and 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 they can quickly be changed to any of a predefined set. The defined set is user expandable as demonstrated in this experiment. Execution environment dependencies reside in the toolset rather than the application code allowing rapid ports to new environments based on toolset ports.

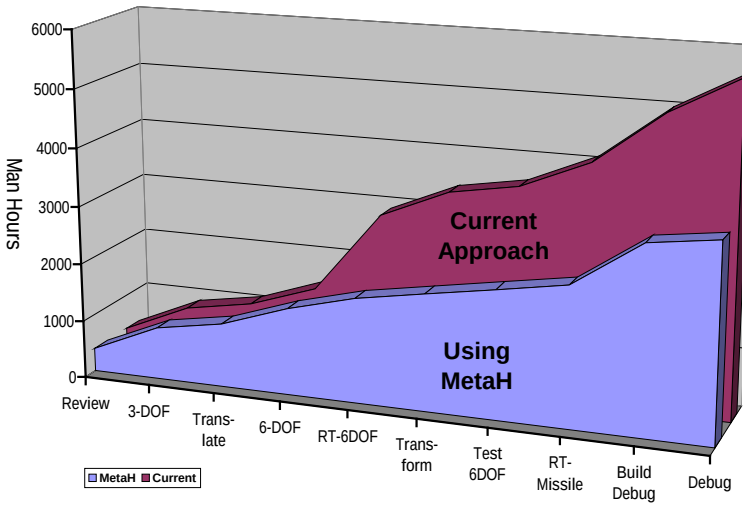

Figure 3. Cost in Manhours

The Demonstration

The MetaH Application Ported

The SED developed a generic missile architecture and used it to re-engineer the on board software of an Army missile system. MetaH was used to specify the architecture interfaces and timing and to integrate re-engineered components and the dual 80960 embedded hardware together to create an executable system. In addition, a 6 Degree of Freedom (6DOF) Software-In-The-Loop missile flight environment simulation was developed to allow us to evaluate flight characteristics. It was also re-engineered into MetaH so it could be executed in real-time. The 6DOF took one and a half processors and the missile on-board code ran on the remaining half processor. During this first development, MetaH reduced the total effort in man-hours by more than 40 % as shown in Figure 3.

This first effort provided a MetaH application that flew correctly on a specific embedded platform. The missile flight control application is quite sensitive to timing errors and delays since it flies at an angle of attack that causes it to overcorrect or tumble easily. It took about a man month with an expert systems engineer to debug and tune the system to fly.

For the onboard missile, we had 12,000 source lines of application code (non comment, non blank) in 11 concurrent processes and 800 lines of textual MetaH generated from a graphical specification. Developing the graphical specification is significantly easier than typing textual code for most people. The 800 lines of MetaH are used to build the MetaH executive which for the missile is 6100 source lines of code. Of those about 3000 are generated based on the MetaH Missile specification and the rest are standard services MetaH always uses. The number of lines generated depends on the complexity of the specification.

While converting the application to the reference architecture we had designed and to MetaH required an investment, it was cost effective. The cost was less that would have been required to port a typical real-time OS solution on our first application of the

technology. Our data for the 6DOF simulation (our first port) shows that MetaH saved us more in embedded system integration time (1200 man hours expected from prior simulation ports to real-time execution) than it took to learn (240 mh for training and development of the first MetaH process) and to specify the remaining MetaH processes (180 mh).

Honeywell experience porting the MetaH toolset

Honeywell developed their initial MetaH toolset port to the i80960MC which took 9 man months to develop but used no standards in the interface layer. It was custom developed for the Tartan multi-program run-time and i80960MC hardware. In contrast, the single processor Pentium retarget using standard Ada95 features took 90 hours including some bug workarounds and performance tuning. Creating the multi-Pentium toolset port took another 75 hours. To add execution monitoring (optional) to the targets took over 300 hours due to problems in the O/S.

Army experience porting the missile application using MetaH

The SED developed a number of missile application configurations on single and dual 80960s using MetaH's specification capability and moved the 6DOF to a PC NT environment. Our Pentium port moved the 6DOF back onto the embedded environment.

The first port to a totally new environment was to the Pentium Processor VMIC-VMIVME-7589 processor card using Aonix Real Time ObjectAda and Pharlap ETS O/S. Honeywell produced the toolset port. It took us 24 hrs to come up to speed with the Aonix tools and get the application code running. Reintegrating the 6DOF into MetaH and porting the MetaH reflective memory interface took another 19 hours, which was a system upgrade rather than porting time. The application flew correctly on the new environment the first time it ran. No time was spent modifying or tuning application code to get the timing right.

We also changed our host environment from the Sun to a PC running NT, another optional upgrade to our environment. MetaH toolset environments existed for both host environments. It took us 31 hours to move the MetaH spec from the Unix file system to the NT

Our next port was to the dual Pentium target using the Aonix compiler with the Pharlap O/S. Here our application, when generated, broke the new dual processor MetaH target. The Tundra chip commonly used on Pentium VME boards had a bug. We developed our own dual processor MetaH Target HW spec and integrated it into MetaH in 48 hours. We developed a workaround for the Tundra chip in 80 hours. After the hardware problems were solved, we re-specified the system for processing on both processors, regenerated it and had it flying in 1 hour. No time was spent modifying application code or twicking application timing.

Our third port was to the PowerPC. In this case we did the toolset port independently and then regenerated the system from the specification and application components on the new execution environment. We did not develop the MetaH execution timing feature on this port since it was not required to build or fly the missile application. Our total time required was 36 hours. This consisted of 10 hours to learn the Green Hills Compiler and VxWorks O/S sufficiently to get execution, 8 hours to add byte swapping between the PC with launch control and the embedded environment, 16 hours to modify the MetaH HW spec for the Pentium to the PowerPC, and 2 hours to change the MetaH Target package for the locations of hardware ports. The missile application built by MetaH flew correctly without changing timing in the specification or changing application code.

Why MetaH Reduces Porting Cost

MetaH improves portability of real-time applications for two reasons.

First, MetaH uses Ada 95 interfaces to provide functional portability between different execution environments (compilers, run-times, operating systems and processors). The use of standard language and OS interfaces is not new, but our work indicates they can be successfully applied in real-time systems. Use of these interfaces makes it significantly easier to port the toolset to a new execution environment. We estimate an 8 to 1 reduction in porting costs using standard Ada95. A standard POSIX interface has also been developed but early indications say that more work is needed in the toolset to demonstrate significant reductions with real-time POSIX. Standard configurations of POSIX, profiles, and conformance testing will also be important.

Second, and more novel, MetaH provides a target-independent tasking and message passing timing semantics. The behavior of a real-time system is as dependent on the timing behavior as on the functional behavior, and in order to achieve portability it is necessary to provide standard timing interfaces as well as standard functional interfaces. MetaH insures that variations in source code execution time have no effect on the values output by a system or the times at which those values are output as long as the overall system remains schedulable (which can be assessed using the MetaH schedulability analysis tool).

What this means is that the X (example 100 th) execution of process A always inputs into the Y execution of process B (say 400th), on any system, whether a workstation or embedded or even embedded multiprocessor systems with A and B on different processors. The result is that the simulation (executive generated by MetaH) gets the same numerical result as the embedded real-time (executive generated by MetaH for a different platform). This capability is provided in a multitasking environment which is auto-constructed by specification and is easy to modify and scale up or down. A second impact is that tactical code in the simulation can be easily ported to the embedded multiprocessor platform by re-specifying the target and allocating the processes to processors. This provides rapid porting from the simulation environment to embedded execution environment with correct scheduling in both environments.

Analyzing the Cost Impact

Retargeting or porting a MetaH application is basically retargeting the MetaH toolset if the target does not already exist. Low cost retargeting means more pre-existing targets and far less risk for targets not yet available. The application itself will be rebuilt by MetaH on the target once it exists. This assumes that the application code will execute properly in the new environment. For instance its important to use portable features of standard languages including precision of data types and functions. MetaH should also make the O/S calls rather than risking potentially non-portable direct O/S calls.

Retargeting the MetaH toolset requires three steps. First, an interface layer that maps MetaH executive calls onto the underlying micro-kernel, run-time or RTOS must be developed. The current toolset comes with standard interface layers that map MetaH executive calls onto either Ada 95 tasking and real-time annex services, or onto real-time POSIX services. Obtaining the interface layer is usually a matter of working around target-specific bugs and doing target-specific performance tuning. Second, a MetaH specification of the new type of processor must

be developed. This specification identifies the interface layer code and also includes overhead timing measurements obtained by running some benchmark programs developed for use with MetaH. Finally, a set of make scripts may need to be modified to invoke the specific compiler, linker and other cross-development tools used for a specific target.

In the best situation of predefined MetaH toolset targets and portable application components, the port is extremely easy. Port time would be the time needed to change the target in the MetaH spec, run MetaH and the compiler, download and execute. It takes about 10 minutes to make a MetaH processor change and rebuild on the new hardware. Most of the time would be spent getting the new execution environment in place. However, if we assume three days to get the execution environment working and 10 minutes for MetaH to change the specification, versus 9 months to port and retune normally, then the ratio is 60 to 1. Since this ratio is dependent on the size and complexity of the system, the savings ratio could be much higher. Also, many a program has run into significant re-tuning risk and cost at the transition between simulation and embedded code execution.

One way to quantify the savings is by expert estimation. Our engineer, Ken Stachelbeck from Coleman Research, has over 18 years in hardware/software integration experience and has developed and ported many hardware-in-the-loop simulation environments (missiles, mortars, smart bombs, RPV's). He personally accomplished both the port from the 80960 to the Pentium, and from the Pentium to the PowerPC and so knows each of the environments involved. His estimate was that to accomplish the port and tune by hand (unassisted by MetaH) for the dual Pentiums would have required 8 + 3 man months. The port and retune to the single PowerPC would have required 5 + 2 man months. In contrast, given that we were porting to a new target based on POSIX or Ada95, we probably could have it running in less than 4 weeks, maybe in 1 week. The benefit for a small application like the missile flight software for the case where a MetaH target did not already exist would still be about 10 to 1 over the conventional approach. The benefit becomes much higher as the application size grows while the MetaH port time remains relatively the same.

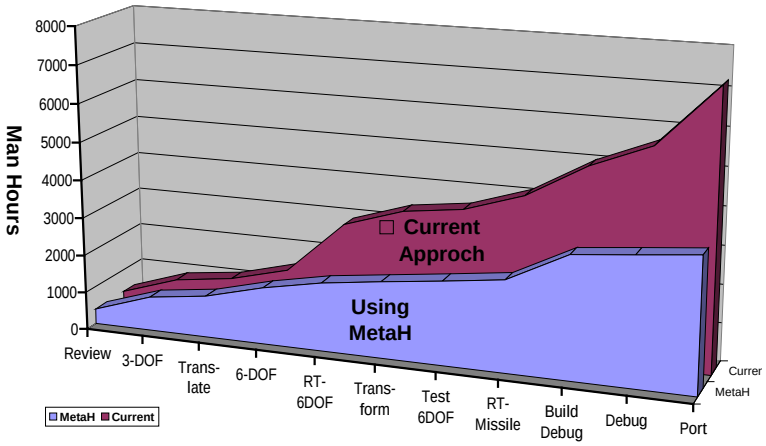

Figure 4. Cost in Man-Hours after Port

Figure 4 above adds man-hours for one port to the missile project.

Additional lessons learned during the ports:

The MetaH implementation on the 80960 interfaced at a low level to the Tartan runtime system. Protected address space targets (i80960MC, real-time Solaris, Lynx) need a trap mechanism. The overheads on these targets can be significantly reduced through the use of an eXtended Operating System (XOS) interface. The SAE GOA subcommittee has evaluated the possibility of a standard XOS but this seems quite difficult. Our current estimate is that the cost of context switches can be reduced by 80% if a low level

XOS interface is used. A standard would benefit all high performance applications with greater portability.

The single address space targets may suffer increased overheads due to multiple software layers, an Ada run-time on top of an RTOS. Performance tuning of the MetaH toolset port can improve this at the expense of stepping outside the standard for a few critical services. For example, the context switching time using the Aonix run-time layered on the Pharlap ETS RTOS on the 233Mhz Pentium is about 10 us after some target-specific performance tuning. The context switching time using the fully standard Ada 95 interface layer was originally about 50us. This was reduced by substituting calls to ETS services for a few of the Ada 95 run-time services.

On the 20Mhz i80960MC with protected address spaces, we were getting 60 microsecond context switches using low level traps (80960 architecture was optimized for fast multi-register set context switches).

One of the benefits of MetaH is that performance tuning of the MetaH toolset port does not affect the portability of the MetaH specified system as demonstrated by the ports between the 80960, Pentium and PowerPC. Execution behavior will be correct if schedulable.

We were very impressed with the portability of the MetaH toolset and missile components across the Ada95 environments. We were also impressed by the (non-standard) performance of the Tartan target port for space and time partitioned targets.

However, to support space and time partitioning and advanced scheduling approaches to provide efficient mixed soft and hard real time requirements for modern avionics architectures with an Ada95 runtime, we also need POSIX like partitions as well as process and thread CPU timers. Accurate task timing requires services from the underlying micro-kernel, run-time or RTOS that are not currently covered in the Ada95. These services must be currently implemented in a target-dependent manner, are optional, but are necessary to support certain MetaH capabilities such as time partitioning, multi-criticality scheduling, and slack scheduling of aperiodic and incremental processes. The Execution Timers under POSIX 1003.1d will provide portable services if they are supported by real time POSIX O/S suppliers.

Ada95 compilers currently use layered operating systems to provide these capabilities and the MetaH middleware generated calls POSIX rather than use Ada runtime features. However, applications that would choose Ravenscar are very performance sensitive as well as safety critical. High performance avionics applications may need go under the POSIX API to achieve performance. Now we are back to a non-portable approach and more expensive retargets. Could a higher performance approach be available through Ada that would also provide the high level of portability Ada95 has demonstrated?

References:

1. Pam Binns, Matt Englehart, Mike Jackson and Steve Vestal, "Domain-Specific Software Architectures for Guidance, Navigation and control," Honeywell Technology Center, Minneapolis, MN, International Journal of Software Engineering and Knowledge Engineering, Vol6, No. 2, 1996, pages 201-227.
2. Mark H. Klein, John P. Lehoczky and Ragunathan Rajkumar, "rate-Monotonic Analysis for Real-Time Industrial Computing," IEEE Computer, January 1994.
3. David J. McConnell, Bruce Lewis and Lisa Gray, "Reengineering a Single Threaded Embedded Missile application onto a Parallel Processing Platform using MetaH," $th Workshop on Parallel and Distributed Real-Time Systems, 1996.
4. Farnam Jahanian and Aloysius K. Mok, "Modechart: A Specification Language fore Real-Time systems," IEEE Transactions on Software Engineering, v20, n12, December 1994.
5. Andrew L. Reibman and Malathi Veeraraghavan, "Reliability Modeling: An Overview for Systems Engineers," IEEE Computer, April 1991.
6. Steve Vestal, "Fixed Priority Sensitivity Analysis for Linear Compute Time Models, " IEEE Transactions on Software Engineering, April 1994
7. Design Guidance for Integrated Modular Avionics, AEEC/ARINC 651, Airlines Electronic Engineering Committee/ Aeronautical Radio Inc. 1991
8. Software Considerations in Airborne Systems and Equipment Certification, RTCA/DO-178B, RTCA, Inc., Washington D.C. , December 1992
9. Steve Vestal, "Mode Changes in a Real-Time Architecture Description Language," Second International Workshop on Configurable Distributed Systems, March 1994.
10. S. Ramos-Thuel and J.P Lehoczky, "Algoithms for Scheduling Hard Aperiodic Tasks in fixed-Priority Systems Using Slack Stealing," Real-Time Systems Symposium, December 1994.
11. Pam Binns, "Scheduling slack in MetaH," Real-Time Systems Symposium session on work in progress, December 1996, http://www.cs.bu.edu/techreports/96-027-ieee-rtss96-wip/Home.html.
12. Steve Vestal, Laurent Guerby, Robert Dewar, David McConnell, Bruce Lewis, "Reimplementing a Multiprocess Distributed Paradigm for Real-Time Systems in Ada 95," International Real-Time Ada Workshop, 1997.
13. Jonathan W. Kruger, Steve Vestal, Bruce Lewis, "Fitting The Pieces Together: System/Software Analysis and Code Integration Using MetaH," Digital Avionics Systems Conference, 1998.