AFRL-IF-WP-TR-1999-1506

AVIONICS SYSTEM PERFORMANCE MANAGEMENT

BHAVESH DAMANIA
STEVE VESTAL
DEVESH BHATT
RASHMI BHATT

HONEYWELL TECHNOLOGY CENTER (HTC)
MN65-2200, 3660 TECHNOLOGY DRIVE
MINNEAPOLIS, MN 55418-1006

MAY 1998

FINAL REPORT FOR 09/23/1993 - 05/30/1998

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE,OH 45433-7334

Kenneth Littlejohn

KENNETH LITTLEJOHN
Project Engineer

[illegible]

JAMES S. WILLIAMSON, Chief
Embedded Info Sys Engineering Branch
Information Technology Division
Information Directorate

Walter B. Hartman

WALTER B. HARTMAN, Acting Chief
Wright Site Coordinator
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** May 1998	**3. REPORT TYPE AND DATES COVERED** Final Report, 09/23/1993 – 05/30/1998

4. TITLE AND SUBTITLE Avionics System Performance Management	**5. FUNDING NUMBERS** C: F33615-93-C-1346 PE: 62204F PR: 2003 TA: 02 WU: A1
6. AUTHOR(S) Bhavesh Damania Steve Vestal Devesh Bhatt Rashmi Bhatt	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Honeywell Technology Center (HTC) MN65-2200, 3660 Technology Drive Minneapolis, MN 55418-1006	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) INFORMATION DIRECTORATE AIR FORCE RESEARCH LABORATORY AIR FORCE MATERIEL COMMAND WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334 POC: Kenneth Littlejohn, AFRL/IFTA, 937-255-6548 x3587	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-WP-TR-1999-1506

11. SUPPLEMENTARY NOTES
Honeywell Technology Center has copyright license.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(Maximum 200 Words)***

Performance management includes the methods and technologies used to develop and assure the timing behavior of real-time systems. The reasons for improved performance management include advances in processor technology and architectures, increasingly integrated systems, and the requirement of reducing costs in developing and deploying the systems. These methods include schedulability modeling and analysis, simulation modeling and analysis, and performance instrumentation and testing. The approach used under this program is an integrated use of technologies and methods that leverage the strengths of individual methods while overcoming their weaknesses.

14. SUBJECT TERMS Performance Management, Real-time Avionics System, Domain Specific Software Architectures	**15. NUMBER OF PAGES** 46
	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** SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Table of Contents

Table of Contents

List of Figures

Avionics Systems Performance Management

1. Executive Summary

This report summarizes the work performed under the Avionics System Performance Management (ASPM program funded by U.S. Air Force Wright Laboratory under the contract No F33615-93-C-1346. Avionics System Performance Management includes the methods and technologies used to develop and assure the timing behavior of real-time avionics systems. The reasons for improved performance management include advances in processor technology and architectures, increasingly integrated systems and the requirement of reducing costs in developing and deploying the systems. These methods include: schedulability modeling and analysis; simulation modeling and analysis; and performance instrumentation and testing. The approach used under this program is an integrated use of technologies and methods that leverages the strengths of individual methods while overcoming their weaknesses. The program resulted in the integration of a suite of tools that enables the developers of avionics systems to analyze the performance characteristics of a real-time avionics application and to perform rapid iterative debugging, testing, and performance enhancements to the application so as to ensure that performance requirements are satisfied.The integrated tools include Scalable Parallel Instrumentation (SPI), MetaH and Domain Modeling Environment. The key benefit of such a toolset is the significant reduction in level of effort and the time required to transform an application from a design phase to a real-time multiprocessor application that meets the performance requirements on the target hardware.

2. Introduction

The earlier claim is realized by integrating an existing set of tools developed under previous government programs and/or Honeywell internally funded program: Scalable Parallel Instrumentation (SPI), MetaH, and Domain Modeling Environment (DoME). SPI provides a flexible and configurable architecture for instrumenting the performance of a multiple and/or parallel processor application. MetaH is an architecture specification language for describing the software and hardware architecture of real-time securely partitioned, fault-tolerant, scalable multiprocessor avionics systems. The MetaH toolset supports, among other things, real-time schedulability analysis and the automatic generation of "glue" code that implements real-time message passing and process dispatching for a class of multitarget architecture. DoME is an extensible collection of integrated model-editing, meta-modeling, and analysis tools that provides the rich, expressive power and abstraction facility required to support complex systems development. Graphical MetaH, one of the notations supported by DoME, supports the graphical specification of MetaH language.Transformation tools are provided that support transforming from the textual representation of MetaH to its corresponding graphical representation, and vice versa. Under this program, these three toolsets were integrated to support the rapid specification and automated code generation of a performance measurement experiment. A performance measurement experiment consists of an instrumented real-time avionics application that runs on the target architecture and for which the performance data is gathered, analyzed, and displayed.

Several applications were integrated during the program to demonstrate the benefits of an integrated approach. The first application integrated was the navigation application generated by using the Adage toolset from Lockheed Martin Federal System. Lockheed Martin Federal System, one of the contractors for the DSSA program, developed the Adage toolset that automates the task of generating Navigation application from a library of modules. The final demonstration demonstrated the capabilities of the testbed for the embedded missile application. The US Army Missile Command (MICOM) Software Engineering Directorate (SED) is developing a Domain-Specific Software Architecture (DSSA) for an embedded missile application. The application had stringent performance requirements. The testbed identified the bottlenecks in the system and the subprograms within the application that could be optimized. This information, which was previously very difficult to obtain, can now be utilized in optimizing the performance of the application. The program also developed a set of performance management guidelines that describes why improved performance management is needed and the alternative approaches for achieving performance management, including the strengths and weaknesses associated with each technique. The techniques described include *Schedulability Modeling and Analysis, Simulation Modeling and Analysis,* and *Performance Instrumenting and Testing.* The performance management approach recommended supports the integrated use of formal schedulability modeling and analysis, simulation, and instrumentation and testing. By using these methods in an integrated fashion, the weaknesses associated with each technique can be overcome while taking advantage of their respective strengths. The applicability of these methods as it relates to the major steps of the development process, including *Analysis and Design, Detailed Design and Component Implementation, Software and Systems Integration,* and *Testing and Verification* is discussed. Also, discussed are the performance management process and the application of methods and techniques to the *Capability Maturity Model for Software* and *Software Considerations in Airborne Systems and Equipment Certification.*

The report is organized in the following manner. Section 3. describes the overall capabilities and the features of the Avionics System Performance Management testbed. It discusses the salient capabilities of the testbed and its applicability in developing, testing and deploying a real-time avionics system application. It also describes the physical and hardware architecture of the testbed. Section 4. describes the capabilities and the features of the SPI toolset. It also provides a high-level description of the steps involved in using the SPI toolset in developing an instrumentation experiment and describes the tasks performed under the ASPM program. Section 5.describes the capabilities and the features of the MetaH toolset. It provides a high-level description of (1) the steps involved in using the MetaH toolset in developing and deploying a real-time avionics system application of the testbed and (2) the steps involved in using the MetaH toolset in developing the avionics application. It also describes the steps performed under the ASPM program. Section 6. describes the details of the final demonstration that was performed under the program. Section 7. describes the results of the investigation that was performed in analyzing the alternate mechanisms for accomplishing "performance management" for real-time avionics system. Section 8. details the conclusions from the program and the lessons that were learned during the course of the program.

3. Avionics System Performance Management Testbed

3.1 Avionics System Performance Management

By "performance management," we mean the methods and technologies used to develop and ensure the timing behavior of real-time systems. Improved performance management is needed for the following reasons. First, processor technology and architectures are changing rapidly in a way that makes performance management more difficult. Compute (execution) times are becoming more variable and more difficult to model and bound due to increased use of complex architectural features such as caches. Increased processor throughput means more functionality, more tasks and events, and larger and more complex scheduling problems. Second, systems are becoming increasingly integrated. Performance management must span multiple heterogeneous processors and buses. Performance management must deal with more complex workloads in which hard real-time tasks share resources with soft real-time event-driven and interacting tasks. Third, cost reduction is essentially a requirement. Improved performance management can reduce hardware resource requirements and increase software portability. This means that improved performance management can reduce recurring production and upgrade costs (which usually dwarf initial software and system development costs). To improve performance management, this program used an approach that integrates the MetaH and SPI toolset.

3.2 ASPM Testbed Features

The Avionics System Performance Management system is a comprehensive testbed for the rapid development, testing, and deployment of a real-time avionics systems application. The testbed consists of a collection of tools that significantly reduces the development and the testing of complex real-time avionics systems developed using a multidisciplinary development approach.

The development of avionics systems requires the talents of many engineers trained in a variety of engineering disciplines, of which software engineering is only one. There are trends toward more concurrent and iterative multi-disciplinary development, toward greater reuse of a variety of development artifacts, and toward greater automation and increased use of specialized tools. Different disciplines are sometimes best served by different domain-specific languages and tools, and the results of their work must somehow be integrated to form a complete system. The Avionics System Performance Management toolset provides the mechanisms for integrating development artifacts from different domain-specific tools and generating the final executable image that can be deployed on a real-time testbed. The testbed supports the toolsets for analyzing the performance characteristics of the deployed application to detect problems, enabling a significant reduction in cycletime associated with the iterative nature of testing and deployment of application. Overall, the testbed significantly reduces time and effort in the development, testing, and deployment of real-time avionics application in a multiprocessor configuration that meets the performance requirements.

The toolset within the Avionics System Performance Management testbed supports the following capabilities.

- Multidisciplinary real-time development and test environment
- Significant reduction in the development, testing and deployment of real-time avionics systems
- A toolset for performing what-if analysis on the alternate rates on the schedulability of the application
- Flexible display architecture for viewing and analyzing performance data from the application
- A configurable instrumentation environment
- A configurable environment for building applications with or without the instrumentation system
- Performing post-analysis of the experiment
- Performing both fine-grained and coarse-grained time analysis of the experiment
- Library of performance management actions for supporting the instrumentation needs of typical avionics appli-

cations

- Graphical specification of the experiment
- Ease of configuring experiments to execute on a multiprocessor system

3.2.1 Multidisciplinary Real-time Development and Test Environment

Different disciplines are sometimes best served by different domain-specific languages and tools, and the results of their work must be integrated to form a complete system. A common problem with doing multidisciplinary development is the level of effort required to integrate and test the artifacts generated from multiple toolsets and ensuring that the real-time schedulability and performance requirements of the application are satisfied. The integration and the testing of these artifacts account for a majority of the costs and lead to significant program overruns and delays. The ASPM testbed addresses this limitation by providing a collection of tools for rapidly integrating the artifacts from multiple domains/disciplines and supports instrumentation of the resultant application for verifying the performance requirements and computational correctness of the application. The MetaH environment provides the software and systems analysis and integration toolset. This environment enables the user to perform static analysis of the real-time characteristics of the application and generation of the executable image. The SPI toolset provides the real-time dynamic analysis capability that enables the user to validate the performance and correctness requirements of the application.

3.2.2 Significant Reduction in the Development, Testing, and Deployment of Real-time Avionics Systems

The ASPM testbed significantly reduces the level of effort and time required to move the application from a design phase to being a fully qualified, deployable real-time multiprocessor application. The tools automate the tasks, including performing static analysis of the application, automated glue-code generation for the real-time systems, and dynamic performance analysis of the application. A key issue confronting the developers of an avionics application is the integration and testing of the application. The development, integration and testing of the application are typically performed in an iterative manner wherein the errors and the performance problems detected in the testing phase are fed into the development phase. The ASPM testbed addresses the following classes of problems associated with a real-time avionics application: performance related, application correctness, and schedulability related. Existing systems provide little support for reducing the cycle-time and accelerating the iterative nature of development and testing activities. The ASPM toolset addresses each of these requirements by providing a testbed that automates the tasks of transforming design specifications performed in MetaH (Architecture Specification Language) to an instrumented application executing on a real-time testbed. Once the application is specified, the steps for iterating between the testing and development phases is trivially accomplished by the collection of tools that automate the tasks associated with development and testing.

3.2.3 A Toolset for Performing What-if Analysis on the Alternate rates on the Schedulability of the Application

One problem that confronts the developers of a real-time avionics application is the lack of support for performing rapid what-if analysis on the schedulability of the application under different rate structures. This feature is important to ensure that the available processing power is utilized optimally and that the computational accuracy of the application can be achieved within the constraints of processing power. This requires the capability to analyze the impact of changing the rates on the processes and on the schedulability of the application. Although existing tools support the static analysis of the schedulability analysis, there is little support for environments that support real-time dynamic analysis. The ASPM toolset provides both the static and the dynamic schedulability analysis. The MetaH schedulability analysis tool enables the user to perform static schedulability analysis of the tool whereas the SPI system provides a visual aid to determine and dynamically analyze the schedulability of the application. This feature enables the developer of the avionics application to utilize the processing power optimally and maximize the computational accuracy of the application by selecting the appropriate rate structures for the application.

3.2.4 Flexible Display Architecture for Viewing and Analyzing Performance Data from the Application

The complexity of the avionics system and the real-time requirements make specifying and developing general-purpose diagnostic and error-detection mechanisms very difficult. There is need a for a powerful and flexible mechanism for visualizing and analyzing the data from the underlying application. The ASPM testbed provides a flexible display architecture that enables the users to visualize, analyze, and detect problems in the underlying system and perform the necessary corrective actions. The following are the features of all the SPI displays:

- The ability to store the results of the experiment and replay it, thereby enabling the users to perform post-analysis of the experiment and comparisons across multiple configurations.
- The ability to control the time-granularity of the display, thereby enabling the user to analyze states and transitions at a microscopic level to detect problems in the system.
- The ability to control the speed of the display thereby enabling the users to view the data in slow motion so as to detect problems.

3.2.5 Provide a Configurable Instrumentation Environment

The complexity of the avionics system and the real-time requirements make specifying and developing general-purpose diagnostic and error-detection mechanisms very difficult. A testbed and a development environment are needed that enable the user to perform custom instrumentation programming. A problem encountered with existing instrumentation environments is the difficulty in reconfiguring the system in collecting and analyzing performance analysis data beyond that included with the instrumentation system and the level of effort required to tailor the system. An instrumentation system is needed with an open-ended architecture that simplifies the task of incorporating special-purpose performance gathering and analysis modules while allowing existing display mechanisms and analysis modules to be easily incorporated in an experiment. The SPI system provides such an environment where the user has complete control and flexibility in specifying the performance measurement experiment. The SPI architecture is based on the event-action paradigm wherein the user is provided a development environment for specifying the actions that are performed in response to events in the system and specifying how the events get routed to the actions. This architecture enables the users to develop actions such as analysis, data reduction, and display that can be easily integrated in a performance management experiment.

3.2.6 A Configurable Environment for Building Applications with or without the Instrumentation System

A problem with existing systems that provide the ability to flexibly specify a performance management experiment is the tight coupling that exists between the application system and the instrumentation system. For such a system, it is difficult to sever the instrumentation system link once the application has been fully debugged and tested. The ASPM toolset automates the task of constructing an application with or without the instrumentation system. During the typical development cycle, the instrumentation capability is enabled to allow debugging and problem detection in the application. However, once the user is satisfied with the performance characteristics and accuracy of the application, the user can disenable the functionality. The capability enables the user to deactivate the instrumentation functionality during actual deployment. The experiment specifier merely specifies whether or not the application is configured with the instrumentation system and the task of whether or not to include the instrumentation code is automated by the toolset.

3.2.7 Performing Post-analysis of the Experiment

Existing systems lack a general-purpose capability to perform post-analysis of the data and require rerunning the application to analyze performance measurement data. In many avionics systems, the task of recreating and rerunning experiments is formidable and expensive. The architecture of the instrumentation system enables the user to replay and rerun the results of the experiment even after the experiment is completed. The instrumentation system supports the ability to store the results of an experiment that can be redisplayed and replayed at a later time. This eliminates the need for rerunning the experiment to perform post-mortem analysis of the results. An added benefit of this capability is the ability to perform comparative analysis across multiconfiguration application (e.g., two different experiments run with different rates, single/multiprocessor configuration).

3.2.8 Perform both Fine-grained and Coarse-grained Time Analysis

One of the requirements for a real-time system is the ability to analyze the results of an experiment at varying levels of time granularity (e.g., view the task timeline data at a fine time granularity to verify and validate the schedulability analysis of the application, or observe the trend of the variable during the course of the experiment at a coarser level of time granularity). The ability to see the same set of performance data at varying granularity of time is a vital requirement for analyzing the performance data of an avionics application. The instrumentation system supports this capability as a generic set of features allowing the user to control the display of information at varying levels of time granularity.

3.2.9 Library of Performance Management Actions for supporting the needs of typical Avionics Applications

One requirement for the underlying system supporting the performance measurement of an avionics system is the need to quickly develop an experiment that is instantiated to the specific needs of that application. This is necessary because, for a complex system such as an avionics application, the specification of the performance management application is difficult to ascertain a priori. The instrumentation system delivered as part of the ASPM program addresses these needs by providing a library of displays and the actions for rapidly developing a performance management application. The following are the categories of display action included with the instrumentation system:

- Task Timeline Actions: The task timeline display and its associated set of data collection and analysis actions enable the user to verify and validate the schedulability characteristics of the application. The user can observe the occurrences of kernel-level events such as semaphore locking/unlocking, scheduling of the kernel, etc.
- Strip Chart Display: The strip chart display and its associated data analysis actions enable the user to trend the real-time values of variables under study. The real-time values of output variables can be used to check for the computational correctness of the application. The display can also be used for performing comparative analysis of multiple variables in the system.
- Histogram Display: The histogram display can be used to display the results of those variables that are cumulative over the course of the experiment. The subprogram profiling display can display the relative execution time spent in different subprograms within a process to determine the compute intensive subprograms and processes. This information can then be used to optimize the performance of the system by optimizing compute-intensive subprograms and processes.

3.2.10 Graphical Specification of the Experiment

The complexity associated with assembling artifacts generated from multiple disciplines and domains that is typically required in constructing an avionics system application results in major resource requirements and introduction of errors to the system. One of the means for alleviating such a problem is providing graphical mechanisms for specifying the experiment and the automated generation of a real-time run time executive that is built from the artifacts created using domain-specific tools. The Graphical MetaH addresses that requirement by providing graphical specification and automatic code generation that runs on the target hardware for the performance management experiment.

3.2.11 Ease of Configuring Experiments to Execute on a Multiprocessor System

The complexity of the application, hardware configurations, and processing power requirements typically necessitate executing the application in a cooperatively executing multiprocessor environment. The requirement is for a testbed environment that supports execution of the avionics application in a multiprocessor environment and the ability to rapidly configure the experiment for alternate software process to hardware processor mapping. The MetaH runtime provides the necessary constructs for executing the application in real time in a multiprocessor configuration. The run time provides the necessary mechanisms for real-time clock scheduling and coordination in the executive services in a multiprocessor environment to make the underlying hardware topology seamless to the avionics application.

3.3 ASPM Testbed Hardware Architecture

This section describes the hardware architecture for the ASPM Instrumented Avionics testbed (seeFigure 1). The processors used for the testbed include the Cyclone CVME962 board and a Motorola MV 147 board. The MetaH runtime and the avionics application are hosted on the CVME962 processor. The Cyclone CVME962 is physically connected to the SUN Workstation using the RS-232 cable. The RS-232 cable serves as the interface for downloading executable images developed on the SUN Workstation and the communication mechanisms used by the cross-compiler and the cross-debugger toolset provided by Tartan. The RS-232 based communication mechanism severely limits the amount and rate at which performance data can be sent to the SUN Workstation from the CVME962 processor. So a Motorola MV 147 board is installed on the backplane of CVME962 board, and these two boards are connected via a fast VME bus and have common shared memory. The Motorola MV 147 board is connected to the SUN Sparc station via ethernet that enables data transfer between these processors at a much higher rate than the RS-232 connection. The Motorola MV147 processor board hosts the instrumentation run time thereby creating a configuration comprising a dedicated processing unit for hosting the instrumentation services. This significantly reduces the invasiveness that may be introduced by having the instrumentation run time execute on the same processor as the MetaH kernel and sharing the processing resources with the application software The invasiveness is limited to the event-signaling probes that are inserted in the MetaH run time and the application code. The Motorola MV147 also supports a TCP/IP link to the SUN workstation, thereby providing transfer rate from the instrumentation run time to the display system at ethernet speed.

The Motorola MV 147 board runs the VxWorks operating system, whereas the CVME962 board runs the MetaH kernel. Since these two boards are connected via fast VME bus and the Motorola MV 147 board itself is connected to

SUN workstation via ethernet, multiple VxWorks application processes can do real-time I/O with the 80960MC processor.

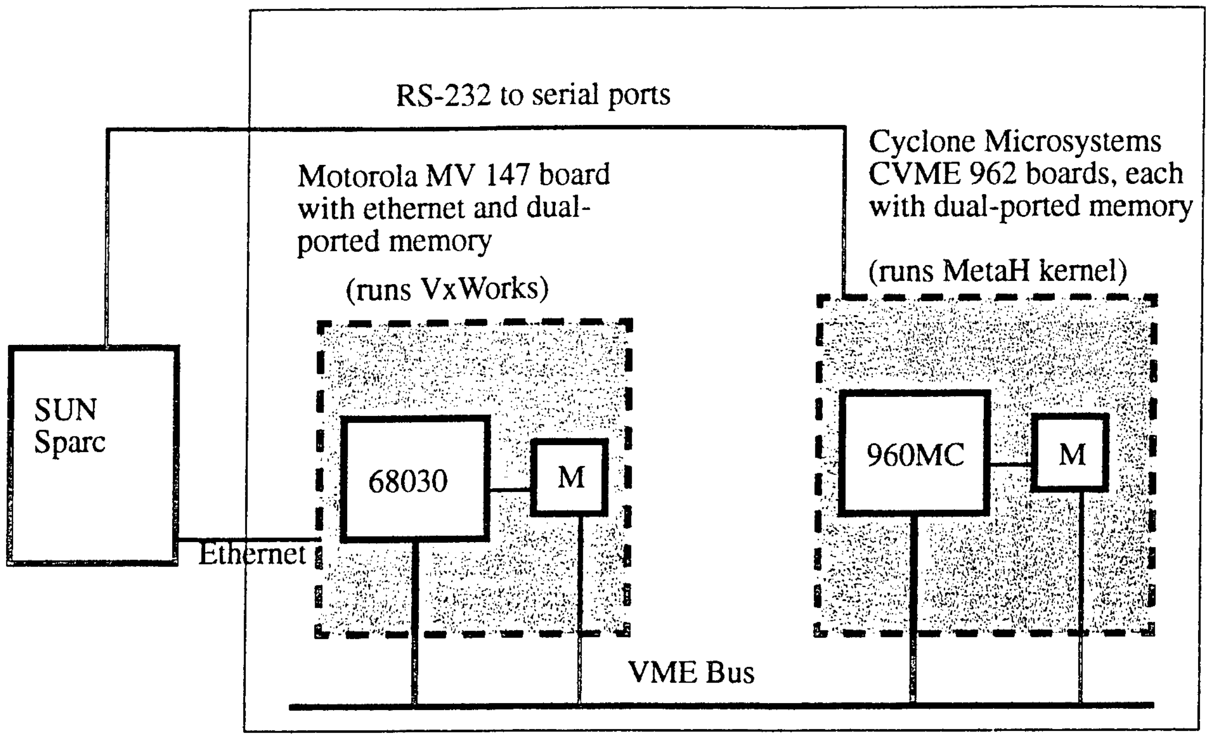

RS-232 to serial ports
Motorola MV 147 board with ethernet and dual-ported memory
(runs VxWorks)
Cyclone Microsystems CVME 962 boards, each with dual-ported memory
(runs MetaH kernel)
SUN Sparc
Ethernet
68030
M
960MC
M
VME Bus

FIGURE 1 Instrumented MetaH Testbed Hardware Architecture.

4. SPI: A Toolset for Instrumenting Real-Time Avionics Systems

Scalable Parallel Instrumentation (SPI) provides a complete development and execution environment for real-time instrumentation functions for heterogeneous parallel/distributed systems. The SPI system addresses the following two critical issues for real-time systems:

- **Design Correctness:** Design correctness must be ensured in the presence of non-deterministic behavior and non-repeatable execution characteristics present in a parallel and distributed system. Ensuring correctness would typically require checking for the absence of race conditions, testing the system under various possible execution interleaving and external stimuli, and checking that the consistency of certain system parameters is maintained by the parallel components of application functions.
- **Requirements Validation:** Reactive embedded systems must respond to external events/inputs and exert real-time control on their environment in the form of actuator control, displays, and data/control interaction with other subsystems. For these systems, the developer must ensure that, in addition to design correctness, the application meet their real-time deadlines, the scheduling mechanisms and policies meet their properties/guarantees, and correct values of outputs are produced.

4.1 SPI Features

SPI provides an environment that allows application developers to build instrumentation that helps obtain the objectives of correctness and requirements validation.

4.1.1 Performance and Behavior Monitoring

This includes real-time continuous monitoring of the performance and behavior of selected system components and activities in both an aggregate and event-based manner:

- Hardware resource utilization, contention/congestion, and OS overheads
- Inter-process communication activity, with message selection by type, source, destination, etc.
- Process execution/scheduling and operating system activity
- Application-level monitoring: flexible probe mechanisms and real-time data-reduction/correlation and display services
- Adaptive monitoring: e.g, turn on monitoring functions based upon results of previous data-reduction.

4.1.2 Testing and Validation

This includes capabilities for testing the functional correctness, parallel performance, and fault-tolerance aspects of the system and validating them against the system functional and non-functional requirements.

- Stimulation capability to inject synthetic loads, test inputs, and various fault modes in the system.
- Experiment control capability that includes execution perturbation and replay and repetitive testing control.
- Dynamic assertion checking capability that allows application developers to specify and verify assertions about local and global properties (e.g., checking for specified casual order of application events).

4.1.3 Instrumentation Architecture and Development Environment

The architecture of the instrumentation system is perhaps as important as the instrumentation function itself; especially in an embedded parallel system. This includes:

- Ability to collect, reduce, and analyze the data on the fly (in real time) with minimal or accountable invasiveness.
- Scalable instrumentation architecture that can itself execute in parallel and can be configured for a heterogeneous target system under study.
- Uniform notation (e.g., a language) to specify instrumentation function selection, customization, and for building application-specific instrumentation functions.

4.2 SPI Architecture

SPI provides a comprehensive approach to instrumentation based on the concepts of *events, actions,* and *event-action virtual machines (ea-machines).* Figure 2 shows the logical view of SPI's distributed event-action model. This model postulates that all the instrumentation functions (such as performance analysis, specific behavior monitoring, stimulation, and display) can be uniformly implemented as *actions* executed in response to *events.* The actions are partitioned (mapped) over multiple communicating *ea-machines*, each providing a run-time environment to its resident actions.

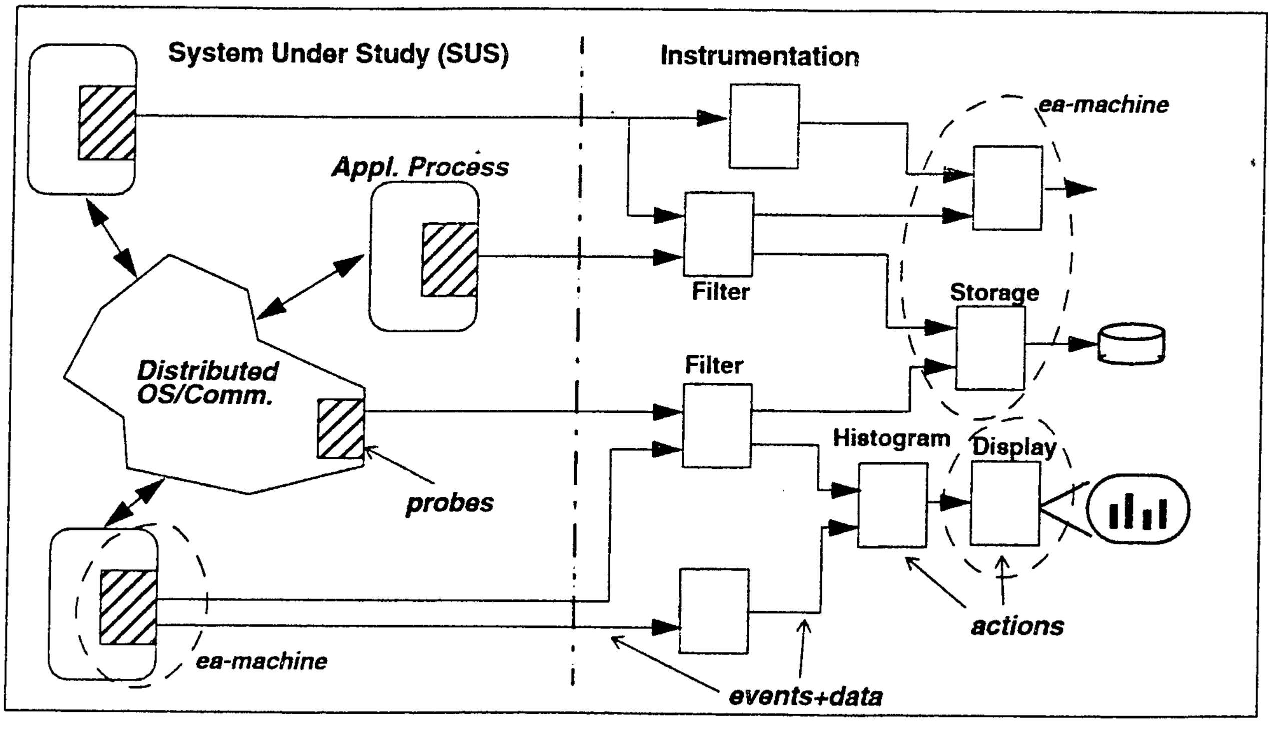


FIGURE 2 Distributed Event-Action Model of Instrumentation

The Experiment Specification Language (ESL) allows the flexible specification of desired instrumentation functions in an experiment as graphs of events and actions, using hierarchical composition. The actions in a typical subgraph act as monitors, filters, event correlaters, data-reducers, or fault-injectors. Multiple ea-machines, distributed across the heterogeneous system, cooperate in real-time to route the events and execute the actions.

SPI allows flexible mapping of instrumentation components over the hardware resources of the distributed system, including special hardware resources dedicated to instrumentation. In an experiment, an ea-machine can share a processor with application software, or it can be executed on a dedicated processor, according to a user-specifiable configuration. This enables continuous, real-time operation of instrumentation and the minimization of instrumentation interference with the application.

The SPI architecture insulates the distributed architecture of the instrumentation experiment at the ESL level. The user specifies instrumentation in terms of abstract graphs of events and actions and mapping directives. The ESL run-time provides complete location transparency to actions by automatically routing events from one action's output port to another action's input port over multiple stages and types of communication.

The SPI architecture is flexible and extensible that permits the users integration of new hardware and operating systems, integration of new action libraries and integration of new display systems.

- Integration of new hardware and operating system: This capability enables new hardware and operating systems to be integrated in the SPI system. Under the ASPM program the SPI runtime was rehosted and retargeted to the VxWorks operating system and the 68030 embedded processor. The rehosting to a new system typically involves porting the communication sub-system and the E-A machine onto the new hardware/operating system. The task of adding a new hardware and operating system typically requires intimate knowledge of the SPI internals and is typically not performed by the users.
- Integration of action libraries. The SPI system is based on the event-action paradigm. Events can range from application signaled event to a clock event. The action is the information processing that occurs in response to actions. The SPI system provides the core set of data collection and data reduction actions needed for performing the performance management for a typical avionics application. However, these libraries can be extended and/or enhanced to support the specific performance management requirements of the application
- Integration of display systems. Under the ASPM program several display systems were integrated in the SPI system. These display systems included the task timeline, strip chart, and multisegment/multibar. Integration of display systems involves developing the data analysis, reduction and display actions.

4.3 Experiment Specification Language (ESL)

ESL is used for specifying an instrumentation experiment. An instrumentation experiment consists of processes representing the system-under-study (SUS) and the instrumentation operations on the SUS. ESL supports the following kinds of program composition, parallelism expression, and synchronization:

ESL supports the following kinds of program composition, parallelism expression, and synchronization:

- Connection of output events of an action to the input events of other actions
- Hierarchical composition by nesting actions to arbitrary levels
- Declaration of arrays of events and actions over subsets and shapes of target system node space
- Full concurrency among actions as per the event-action model
- Mapping directives for mapping actions to target system processors
- Several types of synchronization among multiple input-event streams coming into an action

The above composition constructs make it possible to flexibly specify an ESL program consisting of distributed/parallel components that match the structure of the instrumented system and that can perform the desired data collection, analysis, and display tasks in real-time.

As a simple example, Figure 3 is a graphical depiction of an experiment that monitors the time spent by user processes in certain activities of interest. The experiment is defined as a top-level action named *My_experiment*. It contains two

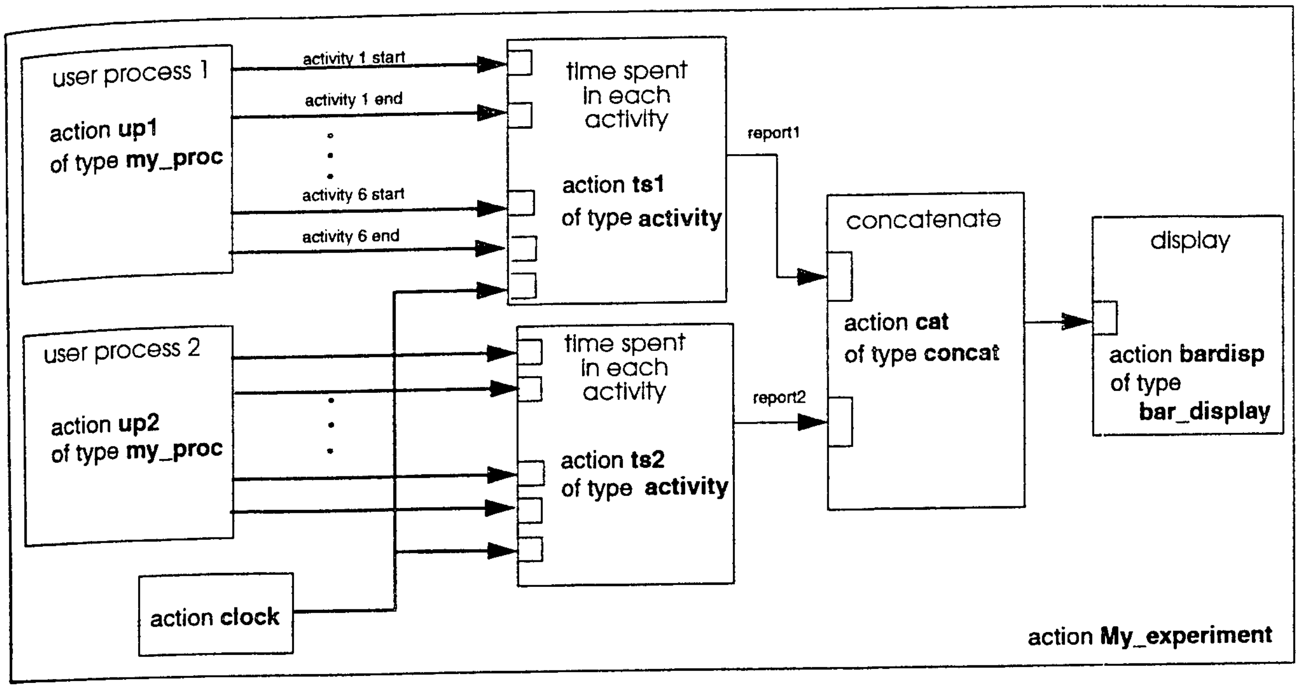


FIGURE 3 ESL Structure of Top-Level Action

user processes represented in ESL as actions *up1* and *up2* (instances of the same action type *my_proc*), each signaling out information about the start and finish time of six different activities. Each of the actions *ts1* and *ts2*, of type *activity*, calculates the time spent by a user process in each activity and builds a 1-D array of six elements containing this data, then signals it on its output port. The action *cat* receives the 2 1-D arrays and builds a concatenated 2-D array (number of processes X number of activities). It fills it with the data coming from the actions *ts1* and *ts2* and signals it on its output port to the action *bardisp*. The action *bardisp* is an instance of type *bar_display*, a special kind of action that executes as a separate X-window client process. It displays the 2-D array in a display window in the form of multi-segment bars. The action *clock* signals on its output port periodically to actions of type *activity* to allow them to calculate and output the time spent by the user process in various activities.

4.4 ESL Development Environment

SPI's ESL development environment consists of a set of tools with which users can use to build customized instrumentation of their applications. The tools provide a high degree of automation and allow users to rapidly construct instrumentation experiments.

The ESL translator compiles user experiment specification to code in the target language. A set of commands (spi.ls, spi.import, spi.rm) is provided to browse and modify the action library. A single command (*spi.make)* is provided to construct the experiment from the ESL specification. It automatically invokes the *mapper* and *builder* tools. The mapper maps actions to ea-machines, allowing for the distribution of instrumentation functions across processors. The builder uses the mapping tables and builds the executable images for the specific ea-machines needed for the experiment. An experiment-specific loader is automatically generated that uses the mapping information, platform configuration, and the information created by builder etc., to load and start the various processes comprising an experiment. To start the remote machines, the loader requests services of a SPI server resident on each remote host.

4.5 ESL Run-Time Environment

The actions written in ESL are supported by a powerful run-time system and a set of library routines. The run-time system provides the abstraction of a *global event space* to the actions. The following is the summary of the capabilities provided by the run-time environment:

- Event buffering and representation optimizations, with run-time procedures for an action to control the degree of buffering and priority for action input and output events.
- Multiple mechanisms of event-probes in target systems, controlled by platform configuration.
- Transparent, multistage routing of events over multiple types of communication and gateways.
- Efficient communication among ea-machines to meet real-time response requirements and minimize the overhead.
- Maintaining a virtual input event queue for each event handler and procedures to manipulate it. Optimizations to share event queues in the same ea-machines.
- Synchronization among an action's input-event streams.
- Procedures for signaling time-based events, including periodic events. This allows for flexible implementation for both event-based and time-based instrumentation functions.

Figure 4 provides a user's view of the SPI environment for the development and execution of experiments. In this figure, any UNIX commands that the user executes are enclosed in rectangular boxes and shown in large bold italics (e.g.: ***spi.make***); certain commands that are automatically invoked by SPI are shown in parentheses (e.g. ***(elab)***).

The following describes the high-level steps involved in using the ESL

- Constructing an experiment which involves developing the *esl* specification and using the tools for generating the executable image for the underlying instrumentation architecture.
- Executing an experiment which involves using the toolset in starting and controlling the execution of the instrumentation system.

4.6 SPI Displays

The most important characteristics shared by all the SPI displays is that they provide the user with the visualization of the system behavior, operation, and performance. Visualization of collected data is a critical element of providing developers with the needed insight into the system under study. The user is able to locate and isolate selected events and activities when presented in graphical form in both real-time and post-mortem analysis mode. A flexible visualization architecture enables the user to display the results and perform analysis of the data.

SPI provides standard display types that are associated with instrumentation such as strip charts, bar graphs, multivariable plots, and task timeline. Unlike many other display and plotting tools, SPI displays must be real-time and be able to operate at fast refresh rates.

All the SPI displays share a common and consistent look and feel and the mechanism for interacting with the displays especially suited for displaying real-time data. The following characteristics shared by all displays make them extremely powerful for analyzing real-time monitoring and diagnostic information for avionics application. The features provided by the display include:

- **Performing post-analysis of the results:** This capability enables the user to store the results from a given experiment and perform post-analysis of the data. It provides the flexibility of performing post-mortem analysis of the application run as well as the mechanisms for performing comparison across multiple experiments. This is an important feature for performing what-if analysis for different experiment configurations. The store/redisplay capability also eliminates the need to rerun experiments that may require a nontrivial level of effort and cost for avionics application.
- **Controlling the time window for display:** This capability enables the user to control the time granularity of the displays, ranging from seeing the activities at fine-grained to coarse-grained time-intervals. In the case of

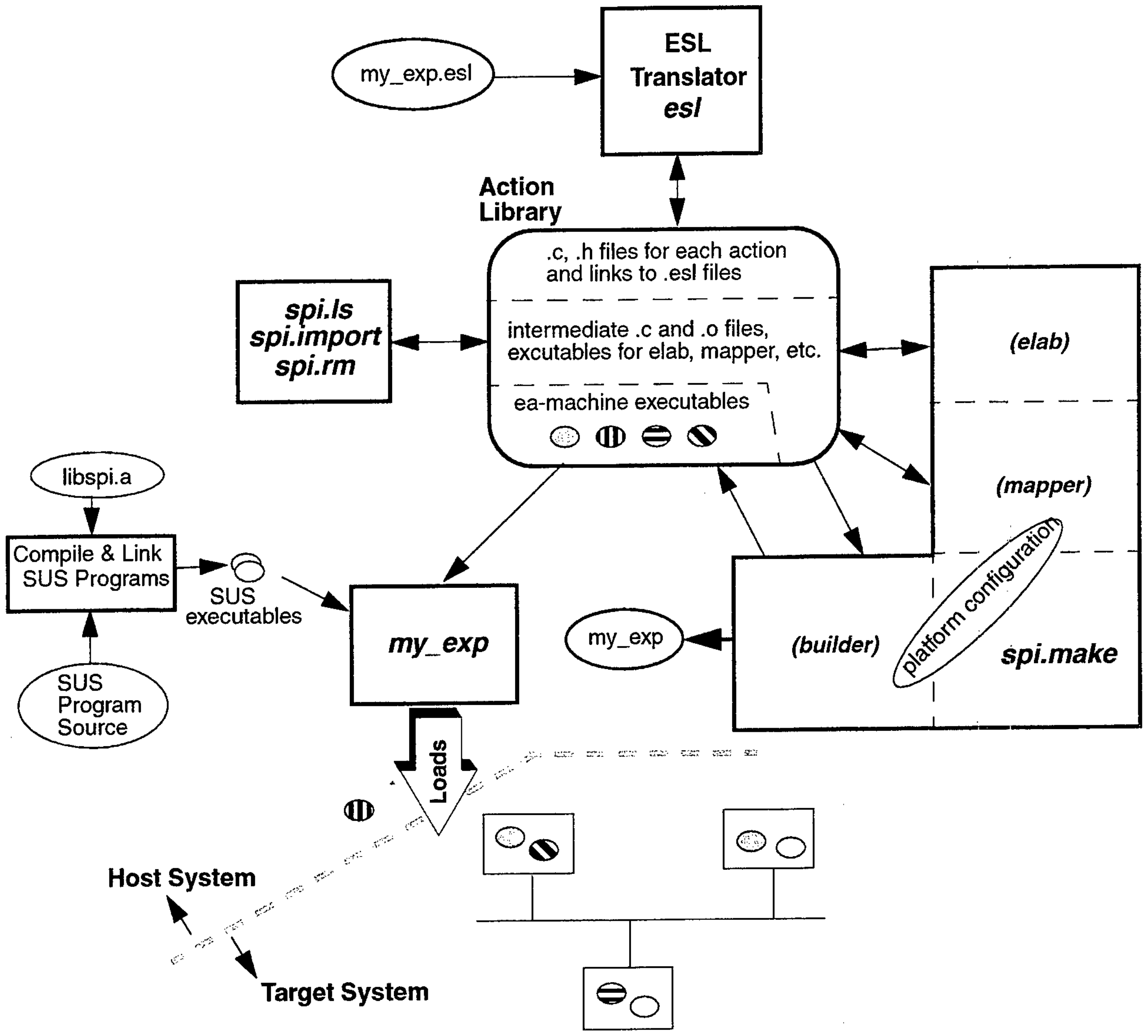


FIGURE 4 Experiment Development and Execution Environment

task timeline display, the fine-grained time-interval enables state transitions or fine-grained process scheduling activities to be viewed. A coarse-grained time-interval displays the overall trend of the variable over the run of the experiment to be viewed.

- **Controlling the speed of the display:** Controlling the speed of the display enables the user to vary the speed of updates, thereby enabling display of the results at varying rates. This is an important characteristic for real-time displays that are used as a mechanism for detecting and diagnosing problems. In a typical experiment, the display can be shown at regular speeds, with slower speeds used to display the data within a time interval where problems are detected and fast mode is used when no problems are detected. Playing the results at slower speeds enables the user to detect abnormal patterns or problems that may be difficult to detect if the experiment is run at normal speed. In Figure 5 the speed control buttons can be used to either single-step a single frame, or control the speed of display.
- **Playing back experiment data during application execution:** The ability to play back display data during application execution provides the control for replaying the data to detect anomalies, patterns, and missed information. In Figure 5, the 2 back arrow keys can be used to play back the display, alternatively the time slider can be used to control the time period of the experiment run that is displayed.

- **Controlling the granularity of the display** This capability enables the user to control the granularity of the display. Zooming out from the display enables the user to view the finer details of the display, whereas zooming in compresses the display to enable the user to perform. In Figure 5 the Zoom In/Zoom Out button in the Display Options area can be used for zooming in or zooming out.

The following provides a high-level description of the displays developed as part of the ASPM program.

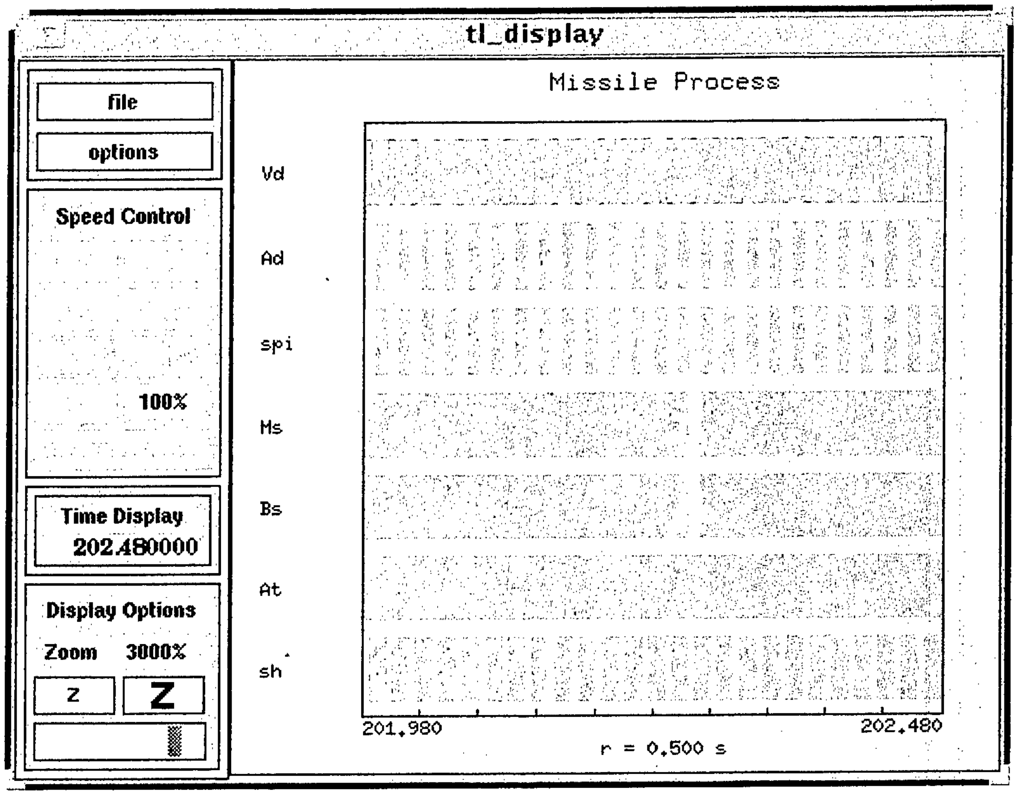


FIGURE 5 Task Timeline Display for the Missile Application

- **Task Timeline Display:** The task timeline display provides a general-purpose mechanism for observing state values and state transitions as a function of time. It can be used for displaying information that typically holds discrete values and where the transitions and/or changes in these values over the time period is important in performing diagnosis of the application. A typical instantiation of this display can be used for displaying the scheduling each process/task of the applications to ensure that the schedulability requirements are met and that the scheduling policies of the kernel are being applied to the application. Thus the results derived from the mathematical schedulability analysis theory can be verified with the actual results. The task timeline display can also be used to display kernel-level activities such as semaphore locking/unlocking, scheduling of the kernel, etc. Upto eight task timelines can be displayed on a single display enabling intertask comparisons (e.g., display the task timeline for each process in the application, the scheduler and the idle process). A given task-line can use a combination of task color and task height to display the state of that task e.g., for displaying the process scheduling information the colors of the task can be used to display the current state (running, ready to run, blocked on semaphore) of the task. The zoom out capability available with the task timeline display can be used to perform micro-time analysis when problems are detected in the task timeline. Figure 5 shows a typical time display in which the task timeline associated with each process in the application is displayed, including the kernel scheduler process.

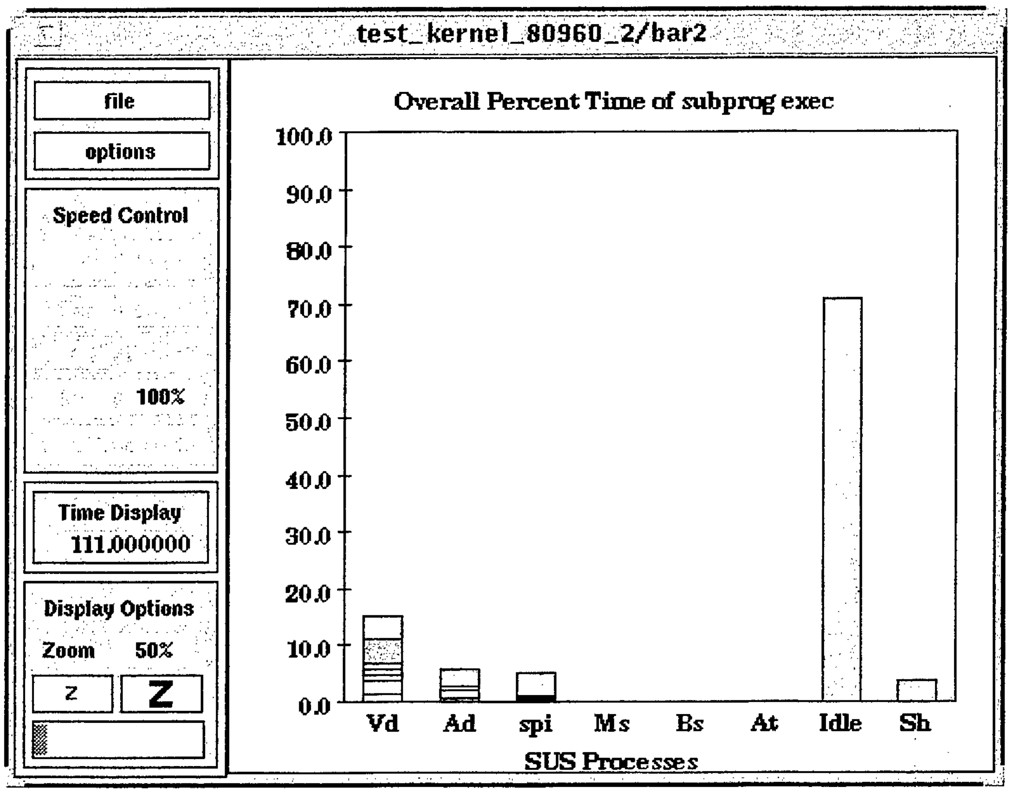


FIGURE 6 Multisegment Multibar Display

- **Multisegment, Multibar Display**: The multisegment, multi bar display can be used to display cumulative values of occurrences in the system. A typical instantiation of the display can be used to display the subprogram execution profiling information. The subprogram profiling display can be used to detect processes and subprogram that are compute intensive. This information can be used to optimize the performance of the process and correct any bottlenecks that may exist in the system. Figure 6 displays the interprocess subprogram execution profile with the height of each bar indicating the percentage of time spent performing that activity (e.g., the display indicates that approximately 70% of the time is spent in the idle process, thereby providing the user with vital information on processor availability). The multiple segments within each bar indicate the relative amount of time spent in each subprogram. A color legend associated with each display lets the user bind the color in the multiple segments to the name of the subprogram. This information can be used in determining the compute-intensive subprograms.

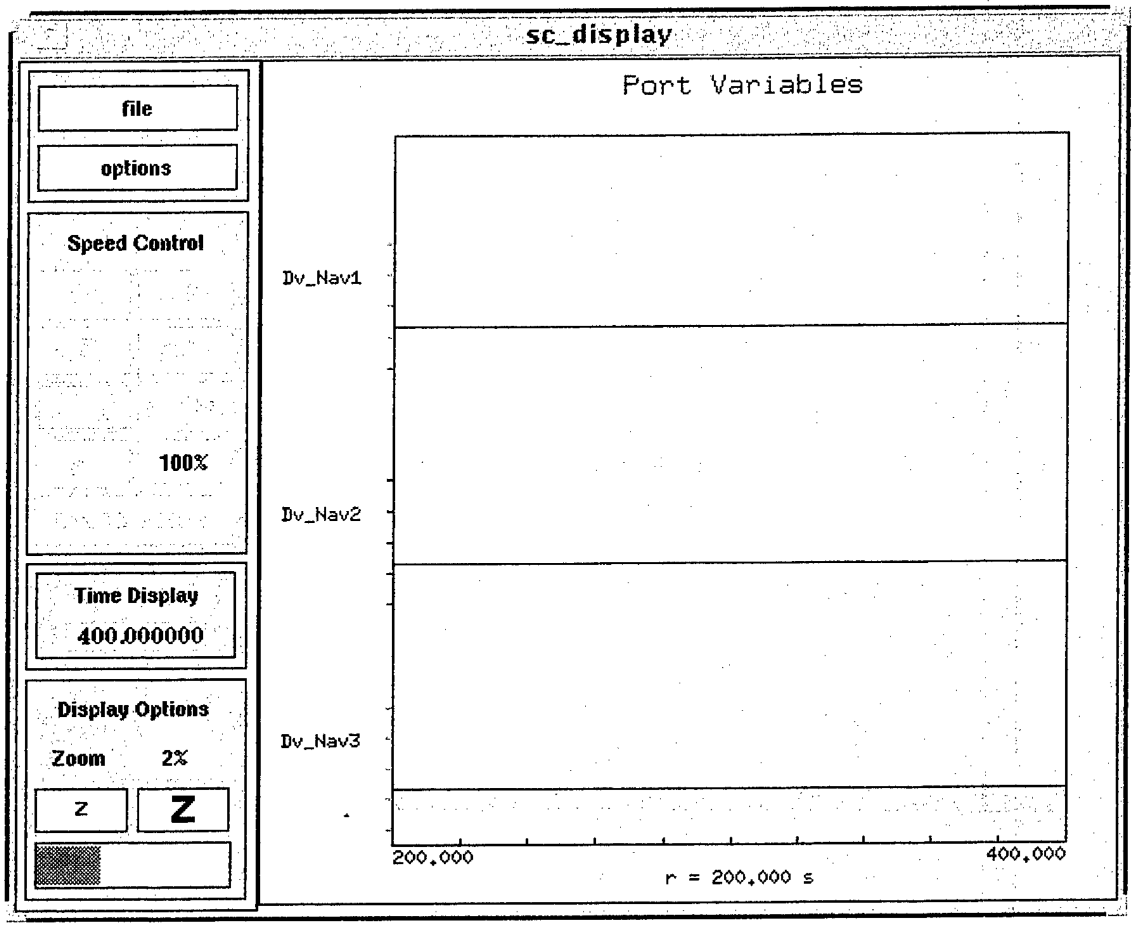


FIGURE 7 Strip-Chart Display of Port Variables

- **Strip-Chart Display:** The strip-chart display provides a general-purpose mechanism for displaying the value of variables from the underlying system under study as a function of time. The display is typically used for plotting the variables. The strip-chart display provides the mechanisms for displaying values of multiple variable from the system, thereby enabling the user to perform comparative analysis between them. The user specifies the bounds of the display. During the experiment the display bounds for a variable can be used to constrain the values of the monitored variables, with a distinctive coloring scheme used to indicate when the variable is out of the display bounds. Figure 7 displays the coarse-grained range of port variables Dv_Nav2 and Dv_Nav3. The strip chart enables the user to monitor the trend of application variables which can then be used to determine the computational correctness of the application. The following are example usage of strip-chart variables:
  - Multiple-variable display: Since in a typical avionics application variables computed by the application tend to be interrelated, the ability to visually see multiple variables on a single display enables the user of the system to determine the computational correctness of the application.
  - Displaying the error variables in the system: The difference between the actual value of the variable vs. the expected value of the variable is the application error. A strip-chart display used to indicate the error values can be used in determining the accuracy of the application.
- Histogram Display: The histogram display can be used to display the results of those variables that are cumulative over the course of the experiment. The subprogram profiling display can be used to determine the relative execution time spent in different subprograms within a process to determine compute-intensive subprograms within a process and the interprocess execution times to determine the compute-intensive processes. This information can then be used to optimize the performance of the system

Incorporating one or more displays in an instrument experiment merely requires the user to create an instance of display action type and to connect another action's output event to the input of the instantiated display action. The SPI mapper and loader tools automatically fork the X-client process for the display, and the other ea-machines transparently route the events to this process.

4.6.1 SPI Displays Design

All the SPI displays consist of two distinct components: the display driver and the custom Xt widget. Combined, they provide SPI with a real-time data visualization and recording capability. The selection of the display and control of major attributes are all controlled directly from ESL. The display actions (task timeline, strip chart. bar graph etc.) are completely integrated into the SPI build process. They are automatically built, linked, loaded, and executed just as any registered SPI action. The experimenter will select or create actions that provide the events in the predefined formats of the particular display. Initialization values for the displays (titles, x, y axis titles, etc.) are also generally set up in the action that directly feeds the display. The experimenter will define the event flows from standard SPI or custom probes, filter and reduce the data, format it appropriately for the chosen display, and connect the display action to the resulting event output.

Based on selections in the user menus and buttons, one can monitor the real-time display or view the captured data while continuing to record incoming events. They display data can be stored to file for later examination and analysis.

The driver component has been designed to provide the functionality of an EA machine and supports the SPI event protocol. The driver accepts SPI-compliant events, buffers them, and based on user selections sends the appropriate information to the widget. The drivers for all the different display types utilize many of the same procedures and utilities. In particular, the log buffer, ea machine utilities, and display control library are used consistently across all display types. The procedure names shown in the driver are the specific ones that are most likely to differ between the display types. The Xt widgets are controlled through the methods and resources of the Xt widget interface.

All displays are implemented with XLIB and X Toolkit Intrinsics. The widgets have been designed to be completely compliant with the resource and method protocol for X Toolkit. The typical SPI user should not be concerned with the actual widget interface. The driver handles all aspects of widget control. The widgets have been designed to provide a very generic graphics capability and it is possible that new display actions could be created by driver modification using the same widget. This is significantly more difficult than configuring an ESL display action.

The high-level structure of a complete display action including both the driver and widget, is shown in Figure 8. The major components are the widgets, the utilities, and the driver. Specific instances of the driver are named in accordance with the ESL action name. For example.ESL action bar_display has a driver of the name bar_display.c. Display action can be modified by developing an ESL wrapper that accepts different data representations and converts them to the standard ESL display action format.

4.7 Enhancements and Extensions to the SPI System

Under the ASPM program several enhancements were performed to the SPI run time and the SPI display systems.

4.7.1 Enhancements and Extensions to the SPI Run time

Several configurations were analyzed and evaluated in providing the instrumentation functionality for the MetaH application and kernel including:

- Rehosting the SPI run time on the same processor as the avionics application and the MetaH kernel: This alternative would have entailed rehosting the SPI run time on the same processor as the MetaH kernel. This would impact the schedulability analysis of the MetaH kernel, since, the scheduling of the SPI run time would need to accounted by the MetaH run time. The static schedulability analysis will need to account for the overhead of the

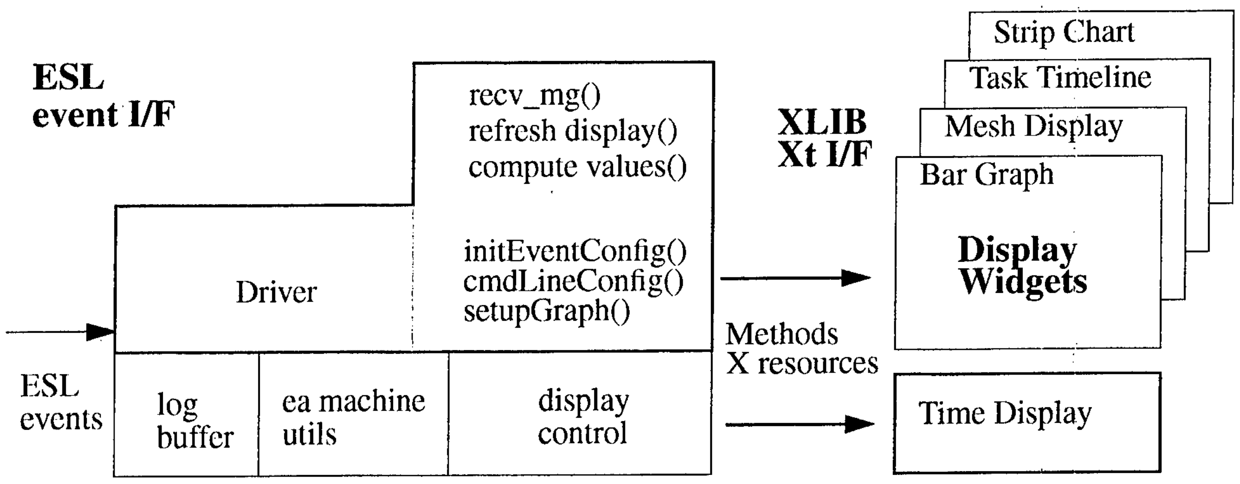


FIGURE 8 SPI Graphical Display Structure

SPI run time in performing the schedulability analysis of the application. Additionally, it would require the development or integration of real-time operating system and networking services on the I80960MC processor. These services are required by the SPI run time in executing the event-action machine and communicating with other SPI run times.

- Rehosting the toolset on a separate processor from the avionics application. This configuration would involve retargeting and rehosting the SPI run time on a separate dedicated processor.

Under this program, it was decided to rehost the SPI software on a separate dedicated processor with TCP/IP connectivity to the host system. The 68030 processor board running the VxWorks operating system from Wind River was selected as the dedicated processor for hosting the SPI software. The TCP/IP connectivity with the SUN host was used as the communication mechanism for communicating the display results on the SUN. The VME memory provides the mechanisms for maintaining the event buffer that consists of events signaled by the MetaH application and MetaH kernel. The following were the benefits for the choice of dedicated hardware

- Minimally invasive instrumentation system: Since the instrumentation system runs on its own dedicated hardware the execution performance of the instrumentation system has no impact on the performance on the MetaH run time and the application ensuring that the dynamic behavior of the system is not impacted.
- Increasing the throughput between the SPI system and host system. The only communication mechanism available for the I80960MC processor with the host system was via the use of serial link. This link severely limits the rate at which data can be transferred between the target system and the host system. Using the ethernet link between 68030 and SUN host would significantly increase the bandwidth of data that can be sent to the host thereby permitting larger collection and display of performance data and providing the scalability needed to address the performance measurement needs for a complex avionics application.

The following tasks were performed in retargeting the SPI run time to 68030 processor

- Rehosting the SPI run time from SUN Unix OS to VxWorks: This task involved rehosting the SPI run time that was hosted on the SUN Unix operating system to the VxWorks operating system.
- Development of event management module between 68030 and I80960MC: The event management module provides the set of services for maintaining and managing the events signaled by processes on I80960MC processor in a shared area of memory on the VME bus.
- Development of event buffering scheme: The event buffering scheme provides the mechanisms for buffering events consisting of the data for the display system from the 68030 to I80960MC to optimize the usage of the ethernet and to ensure that the data transfer mechanisms are scalable to the requirements of signaling perfor-

mance data for an avionics system.

4.7.2 Enhancements and Extensions to the Display System

The displays provided by the existing instrumentation system were not ideally suited for displaying the performance management data for an avionics system. The following displays were developed during the course of the program.

- Task timeline display system: The task timeline display as described in Section 4.6 provides a general-purpose mechanism for observing state values and state transitions as a function of time. The display system comprising of data analysis, data reduction and data display actions were developed under the program to observe in real-time the schedulability of the MetaH application and the MetaH kernel. Extensions were also performed on the MetaH kernel to support the signaling of raw scheduling data that is analyzed and displayed on the task timeline.
- Strip-chart display system: The strip-chart display as described in Section 4.6 provides a general-purpose mechanism for displaying the value of variables from the underlying system under study as a function of time. The display system comprising of data analysis, data reduction and data display actions were developed under the program to observe in real-time the values of variables as a function of time.
- Multisegment, multibar display system: The multisegment, multibar display as described in Section 4.6 provides the mechanisms for displaying the cumulative occurrences of events in the system. The multisegment display system comprising of the data analysis, data reduction and data display actions, was developed under the program to support subprogram profiling. Extensions were also performed on the MetaH kernel to support the signaling of information necessary for constructing the subprogram profiling information.

4.8 Architecture for the Instrumented Experiment

Figure 9 displays the architecture of the processing that occurs on the testbed to support instrumentation of the MetaH application. The MetaH application executes on the i80960MC processor. The instrumentation event-action engine executes on both the 68030 processor and the SUN workstation. The data filtering and the data analysis actions execute on the 68030, while the display actions execute on the SUN workstation. A shared area of memory on the VME bus is used by the MetaH application and the instrumentation event-action machine for signaling events from the MetaH application.

The MetaH kernel and the MetaH application signal three types of events to support the instrumentation functionality. The kernel events are automatically included with the MetaH kernel when an instrumented target of the MetaH kernel is built. The kernel events include events from the scheduler that signals process scheduling information and events from the subprogram profiler that, based on a random number timer, signals the program counter of the executing process. The communication events signaling the state or value of the port variable are signaled by the communication signaler process. The code for the communication event signaler is automatically generated by the DoME toolset. The signaler is automatically integrated by the MetaH toolset when a version of MetaH kernel is built that supports displaying the strip chart for the port variables. The instrumentation probes for signaling application level will have to be manually inserted by the application developers. Signaling an application event requires calling a subprogram with the appropriate parameters. The code for managing the event queue is included for both the instrumentation system and the MetaH system.

The preliminary set of data analysis, data reduction and data filtering actions is performed by the event-action machine executing on the VxWorks. For the kernel events, the data analysis actions construct the process scheduling information and signal to the display action the state of each element displayed in the timeline and the transitions occurring for each element. This information is used by the display system in displaying the task timeline display. The data reduction actions executing on the VxWorks include the creation of the multisegment, multibar display from raw event signaled by the MetaH kernel that indicates the program counter. The data reduction action generates the data for both the interprocess and the intraprocess histograms. This data is then routed to the multisegment, multibar display action on the SUN workstation. The data filtering actions include the actions that take as input the events from the port variables, performing filtering and transformation of that data and signaling the resultant data to strip chart display.

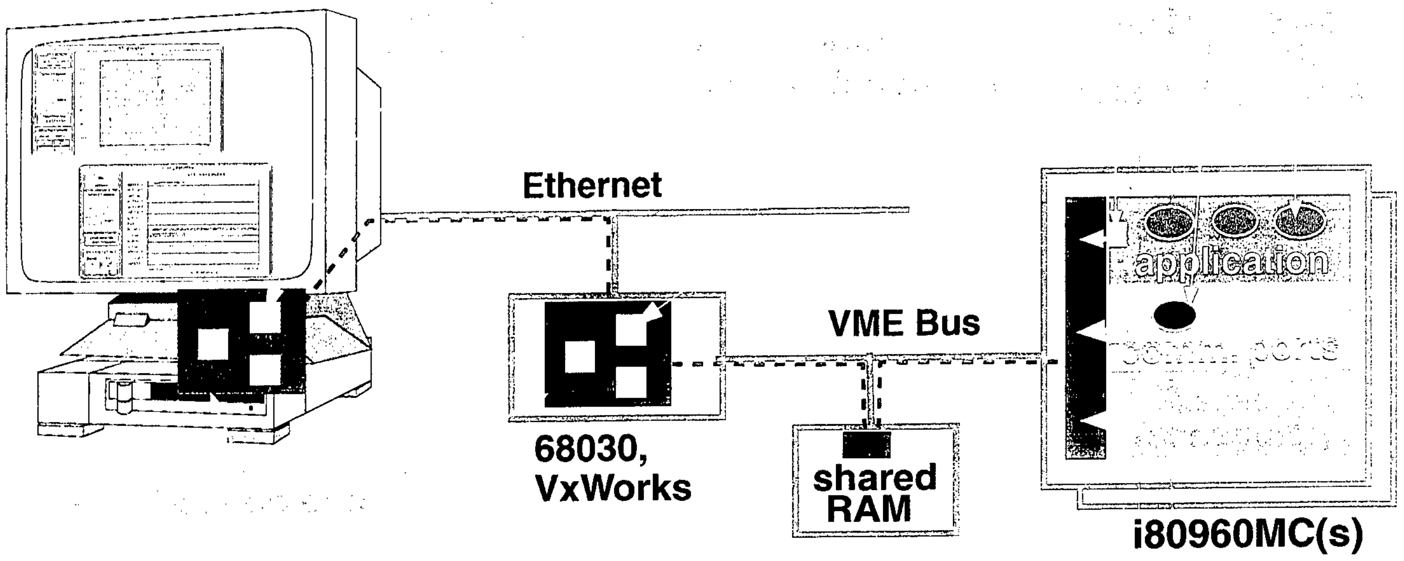


FIGURE 9 Instrumented Avionics Testbed Architecture.

5. MetaH

5.1 MetaH Features

MetaH is a language for describing the software and the hardware architecture of real-time multiprocessor avionics systems. The language supports the definition of securely partitioned, fault-tolerant, scalable systems. The toolset supports, among other things, real-time schedulability analysis and the automatic generation of "glue" code that implements real-time message passing and process dispatching for a class of multi target architecture. MetaH allows developers to specify how a system is composed from software components such as processes and packages and hardware components such as processors and memories.

Low-level software constructs of the MetaH language describe source components written in some traditional programming languages such as Ada. MetaH subprogram, package, and monitor specifications describe important attributes of source modules such as the file containing the source code, nominal and maximum compute times on various kinds of processors, stack and heap requirements, real-time semaphore protocol to be used, etc. Events (user-specified enumeration literals used in certain service calls) and ports (buffer variables used to hold message values) can appear within source modules and must be described in the MetaH specification.

The higher-level constructs of the MetaH language are processes, macros, and modes. Processes group together source modules that are to be scheduled as either periodic or aperiodic processes. A process is also the basic unit of security and fault containment, and memory protection and compute time enforcement are provided. Macros and modes group processes, define connections between events and ports, and define equivalences between packages and modules that are to be shared between processes. The difference is that macros run in parallel with each other, whereas modes are mutually exclusive. Event connections between modes are used to define the hierarchical mode transition diagrams, whereas mode changes at run time can stop or start processes or change connections.

MetaH also allows hardware architectures to be specified using memory, processor, channel, and device components grouped into systems. Hardware objects may have ports, events, or monitors in their interfaces. Software and hardware ports and events can be connected to each other, and software can access hardware monitors (hardware monitors provide hardware-dependent service calls). Hardware descriptions identify (among other things) hardware-dependent source code modules for device drivers, as well as code to provide a standard interface between automatically composed applications and the underlying real-time operating system.

Both graphical and textual specification are supported. The two can be mixed (part of the specification can be maintained textually and part graphically), and the toolset can translate graphical to textual and vice versa. This is convenient in a software and systems integration tool, since different parts of a specification may be produced by different groups or automatically generated by different domain-specific tools.

A simple software/hardware binding tool assigns to hardware those software objects in a specification that are not explicitly assigned, possibly subject to user-specified constraints.

An executive generator tool automatically produces the "glue" code needed to compose various source modules to form the overall application. This glue code resembles an application-specific executive or supervisor, with code to dispatch processes and pass messages, synchronize access to shared resources, vector events, perform mode changes, etc.

The design schema for the generated executives is based on preemptive fixed-priority scheduling theory. Using attributes in MetaH specifications of process period, deadline, and criticality, the executive generator derives priority, period transformation, and dispatch and time slice refill information used in data tables and dispatching code. Port connection specifications and process timing information are used to schedule and generate code to move data between processes' port buffer variables. Monitor specifications are used to generate semaphore protocol and timing data tables. Code to vector events to dispatch aperiodics or to trigger mode changes, and code to manage mode changes, is also generated.

Using information contained in the MetaH specification and produced by the executive generator, the real-time modeler generates a detailed preemptive fixed-priority schedulability model of the application. The model contains elements common to all applications (e.g., a dispatcher process) and elements that are application-dependent (e.g., process periods and compute times, monitor blocking times, communication times). The model includes all scheduling and communication overheads. The schedulability model is a conservative and accurate representation of the final load image structure and behavior (from the real-time scheduling standpoint).

The schedulability analysis algorithm used is an extension of the exact characterization algorithm. The analysis tool produces sensitivity analysis information describing how compute times may be changed while preserving (or in order to achieve) schedule feasibility; and it allows processes to be decomposed into component source modules and provides timing analysis data for individual source modules. The report generated by the analyzer contains a listing of various application source components and also various executive overheads.

The MetaH language includes a construct called an error model, which allows users to specify sets of fault events and error states. An error model also includes specifications of transition functions to define how the error states of objects change due to fault, error propagation, and recovery events. An individual object within a specification can then be annotated to specify the error transition function and fault arrival rates for that object.

A prototype reliability modeling tool generates a stochastic concurrent process reliability model. Error propagations between objects are modeled as synchronizations or rendezvous between stochastic concurrent processes. Each such propagation synchronization in the model can be controlled using an associated consensus expression, which can conditionally mask propagations depending on the current error states of selected objects. In the MetaH specification, user-supplied consensus expressions describe the error detection protocols that are implemented by the underlying source modules for a particular application. The reliability modeler uses the MetaH error model specifications and annotations to generate the proper set of object error state machines and uses the consensus expressions and design structure to generate the proper set of propagation synchronizations between these object error state machines. A subset of reachable state space of this stochastic concurrent process is a Markov chain that can be analyzed using existing tools and techniques. The MetaH kernel implementation is based on the stochastic concurrent process model rather than one of the popular stochastic Petri Net models because it allows the generation of a reliability model whose structure can be easily traced back to the MetaH specification and vice versa. One of the goals of the toolset is maintaining good mappings between specifications, formal models, and code.

The executive code generated from a MetaH specification includes a number of security mechanisms (protected process address spaces, process criticalities, enforcement of execution time limits, capability lists for executive services). A prototype safety/security modeling and analysis tool checks that the mechanisms properly enforce a particular safety or security policy.

The source objects can be annotated with a safety level to indicate the degree to which the code has been assured correct. The safety policy is that proper operation of an object cannot be affected by an error in any other object having a lower safety level. For example, a higher-criticality object should not receive data from a lower-criticality object (unless the connection is explicitly annotated in the MetaH specification to allow this). A higher-criticality process must have a higher scheduling criticality specified than all lower-criticality processes on the same processor, or else execution times must be enforced on all lower-criticality processes on the same processor.

Objects might also be annotated with a list of security rights. However, the security checks are not currently implemented. The security policy is that an object cannot receive or share data unless it has at least the security rights of the object that supplies the data. The connections or data accesses of an object with many security rights can be annotated in the MetaH specification to show that only some subset of these rights are needed to access information provided through that particular connection or shared data area, however. Security (and safety) properties can also be specified for hardware objects, e.g., an inter-processor channel can be flagged as low-security (or low-criticality), making it erroneous to route a high-security (or high-criticality) message over that channel.

The MetaH system is similar to a number of others such as Durra, Maruti, SARTOR, ABE, LileAnna, CAPS and Onika/Chimera in its use of an architectural specification language for composing source modules written in traditional programming language. It is similar to Real-Time Euclid, Maruti, SARTOR, CAPS and Onika/Chimera in its integration of real-time schedulability theory and analysis with automated code generation or composition. Modes and macros provide capabilities to specify hierarchical system-level state transitions that are comparable to aspects of Durra, Statemate and Modechart. MetaH represents a selection, integration and extension of a variety of basic technologies in an attempt to meet the requirements of a range of real-time fault-tolerant securely partitioned multi processor applications.

5.2 Developing Applications Using the MetaH Toolset

This section describes the high-level steps involved in developing an experiment using the MetaH toolset and executing it on the testbed. The details for specifying an application using the MetaH language can be found in the MetaH Programmer's Manual. The distribution software includes the MetaH specification for the applications that were developed under the ASPM program, including the guidance and navigation application and the missile application.

5.2.1 MetaH Specification of the Application

A typical MetaH specification for an application consists of defining the various modes of the application, the process specification (including the specification of periodicity for each process) within each mode, the in/out port (data variables that are exchanged between multiple tasks), specification for each process, and the connections between the processes. The MetaH specification enables the user to specify the overall architecture of the application and the specification of how components are combined in making the overall application. Both graphical and textual specification are supported. The graphical specification is done using the DOME toolset while the textual specification can be done using a standard text editor and by generating a specification file that conforms to the syntax and the semantics of the MetaH language. The two can be mixed (part of the specification can be maintained textually and part graphically), and the toolset can translate graphical to textual and vice versa.

5.2.2 Instrumentation Specification

Once the standard application specification is performed by the user, the next step is deciding on whether the application will be instrumented during application execution or not. The MetaH language provides an instrumented target option. Selecting that option automatically generates all the glue code necessary for gathering and signaling performance data to the instrumentation system. This provides flexibility and control to the experimenter, who can control whether or not the instrumentation code is integrated with the application. For an instrumented application, the user can view the task timeline and the subprogram profiling displays. Additionally, the user can graphically specify the port (application variables that are exchanged between multiple tasks) variables that will be instrumented. An instrumented port variable will be displayed in a strip-chart display during program execution. The graphical MetaH generates the set of required files to create the MetaH specification and the support files needed to construct the instrumentation subsystem.

5.2.3 Compilation of the Application MetaH specification

The next step is compiling the MetaH specification using MetaH compiler. If the compilation is successful without any errors, it will still give some warning (which can be ignored). If the compilation is successful it will create a directory called *I80960MC* and inside that directory it creates subdirectories for each process specified in the experiment. All the source files defined for each process in the specification and other standard files are placed by the *metah* compiler in the corresponding process sub directories. The MetaH compiler also creates a static time analysis file that contains the schedulability data of the application.

5.2.4 Building an Executable Image of our Application

The final step is to create an executable image from the Ada source files generated by the MetaH compiler. The MetaH toolset provides a build tool that performs the task of compiling all the files for the respective processes and the generation of the final executable image for the application. The build tool uses the Tartan Ada compiler and linker in generating the final build image.

5.2.5 Run of MetaH Application (run of application load image)

The next step is to download the build image onto the I80960 processor and execute the application. A script file is provided for loading the build image from the Tartan Ada compiler to the target processor and executing the application.

6. Final Demonstration

The application selected for the final demonstration was the embedded missile application that was rearchitected and reengineered by the U.S. Army Missile Command (MICOM) Software Engineering Directorate (SED) by using the Domain Specific Software Architecture (DSSA) techniques. DSSA tools developed by the Honeywell Technology Center, consisting of ControlH, MetaH, and DoME, were used. ControlH generates tactical Ada code for guidance, navigation and control systems, and also supports analysis of these systems in a module testing context. MetaH as described in the previous sections creates a real-time executive, based on rate monotonic theory, that binds with hand generated Ada code and code generated from ControlH. The MetaH toolset provides an easy mechanism for modifying process execution rates and a schedulability analysis of the processes in the system. DoMe as described in the previous sections provides the graphical programming interface for ControlH and MetaH. The representations created using DoME can be outputted into the Architecture Description Languages, which are interpreted by the ControlH and the MetaH tools.

The selection of this application for the final demonstration was driven by the following considerations:

- A MetaH specification for this application already existed as part of the work performed by MICOM. This eliminated the need for reengineering an application in MetaH, which could have consumed significant resources from the ASPM program and would have limited the functionality that could be demonstrated.
- The missile application with the rate structure exhibited significant performance measurement characteristics that made it ideally suited for the testbed. The rate structure and the processing requirements of each process in the application posed a major challenge on the testbed. Providing an ability to identify the performance bottlenecks in the system and performing optimizations on it could result in significant benefit to the application.
- Previously demonstrated the development and the reverse engineering of an application for the JAST demonstration

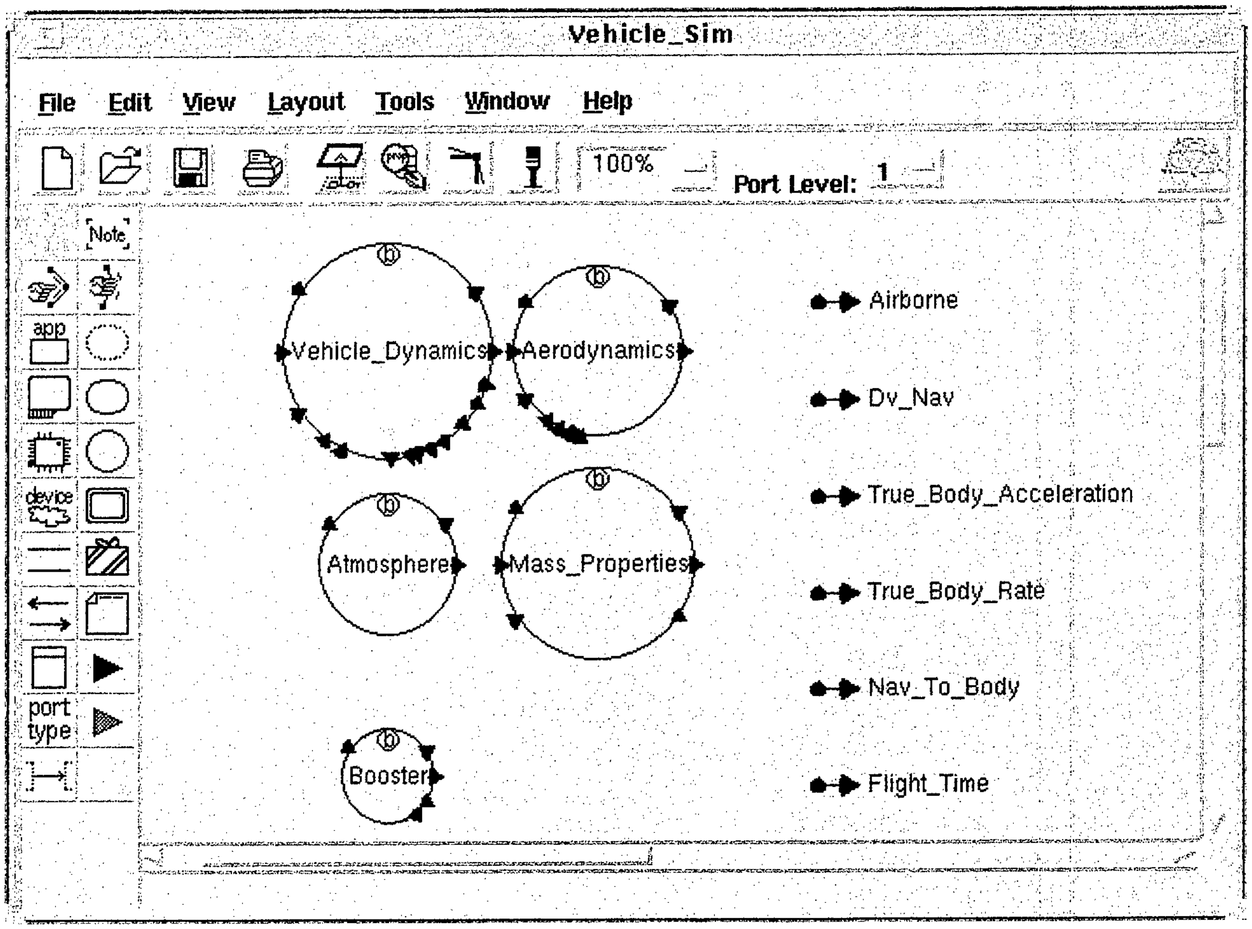
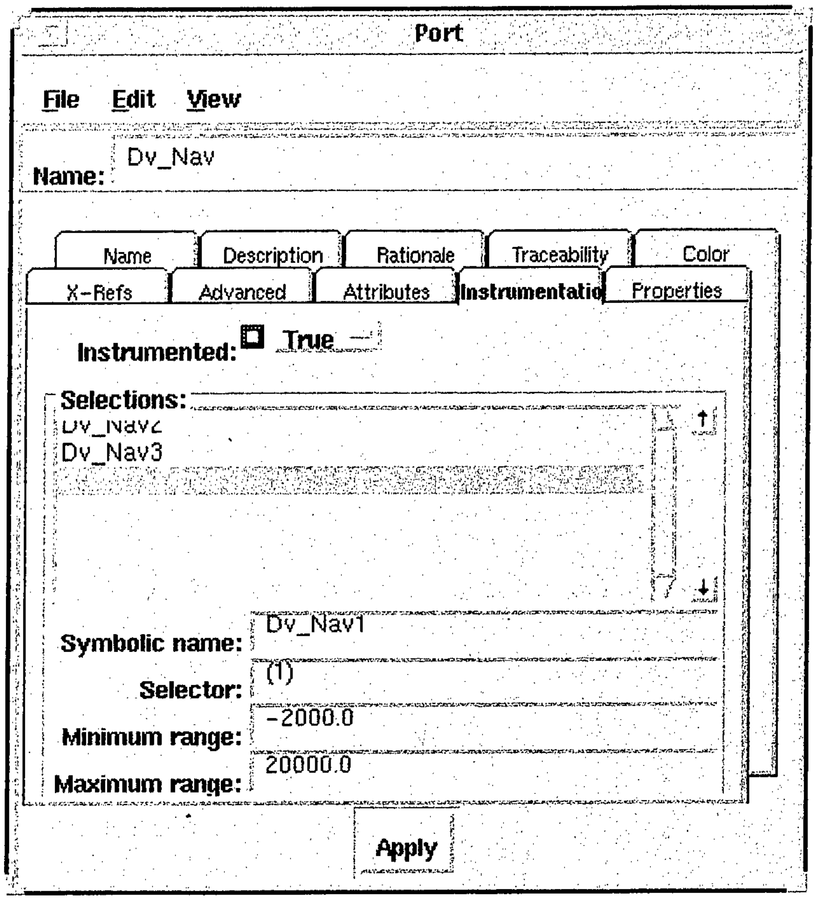


FIGURE 10 MetaH Specification of the Missile Application

Figure 10 is the MetaH specification of the missile application for the vehicle simulation mode. The figure illustrates the processes and the port variable associated with the specification. The process specification and the port specification can be performed using the DoME toolset.



Name: Dv_Nav

Instrumented: True

Selections:
Dv_Nav2
Dv_Nav3

Symbolic name: Dv_Nav1
Selector: (1)
Minimum range: -2000.0
Maximum range: 20000.0

FIGURE 11 The Instrumentation Specification of Port Variable

The DoME tool provides the graphical specification of port variables that will be instrumented during the course of the experiment. Figure 11 illustrates the instrumentation specification of the port variable. The symbolic name maps to the name that will be displayed in the strip-chart display while the minimum range and the maximum range values enable the user to appropriately scale the y-axis of the strip-chart display for that variable. DoME supports automatic code-generation that gets linked both into the MetaH and SPI system that supports the monitoring of the port-variable during program execution.

The following are captured from running the missile application.

Task Timeline Displays: Figure 5 and Figure 12 show the task timeline for the missile application. Figure 5 displays a coarse-grained task timeline for a 0.5 second range of time interval, whereas Figure 12 displays a fine-grained task timeline. In each of these figures the color legend red indicates that multiple activities occurred within that time frame, the green color indicates that the process is executing, and the yellow color indicates that the process is ready to run. Observing the patterns in the figure confirms the correct workings of the real-time scheduler.

Subprogram Profiling Displays: Figures 6 and 13 display the results for interprocess and intra-process subprogram profiling data, respectively. The inter process display provides the information on the relative time spent in each process. After adjusting for the periodicity of the processes this information can be used to determine the compute intensive processes. This intra-process display provides the information of the relative time spent in each subprogram within a process. This information can be used in optimizing the performance of the process.

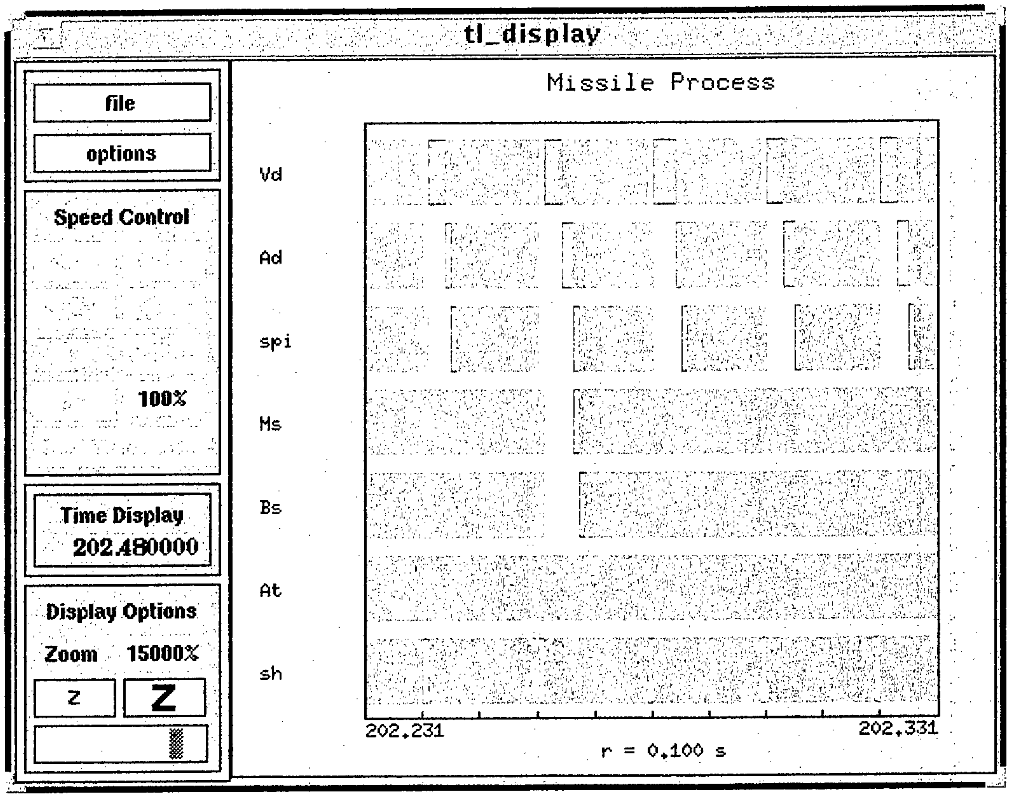


FIGURE 12 Fine-grained Task Timeline for Missile Application

6.1 Results from the Final Demonstration

The approach used for performance management as described in Section 7.1 is one that integrates the methods and the technologies for performance management in an integrated environment. The following is the summary of results:

- Detect the performance bottlenecks in the system: The performance requirements for the application were not being met by the target hardware, thus it was very essential to determine where the performance bottlenecks in the system occurred. The results displayed in Figure 6 identified that the Vehicle dynamics process is the most compute-intensive process. While this information was previously known from static analysis the assertion was verified with actual results by running the application. The ability to actually view the results was not previously available because of the lack of integration of performance measurement application.
- Optimize the workings of the application: The results in Figure 6 and Figure 13 illustrate the most compute-intensive subprograms that occur in the Vehicle dynamics process. The two most compute-intensive subprogram were associated with mathematical functions for matrix manipulations. This information is crucial in determining the appropriate optimizations techniques that can be applied. For this application performing inlining of the matrix manipulation function can significantly improve the performance of the application
- Enable performing what-if analysis under alternate rates: Without an integrated toolset for performance management it is very difficult to perform the what-if analysis on alternate rates for a performance stringent application. This analysis enables the user to determine whether the application meets its performance management objectives, such as schedulability of the application. It also enables the application developer to determine the extent to which the rates of various processes can be adjusted to improve the quality of results. The task-time line display illustrated in Figure 5 enables the user to verify that the schedulability requirements of the application are being satisfied.
- The ability to visually see the results from the actual application as illustrated in Figure 7 provided the ability

to the user to ensure correctness of the application. This capability was previously not available because of the lack of an integrated environment for debugging a real-time application.

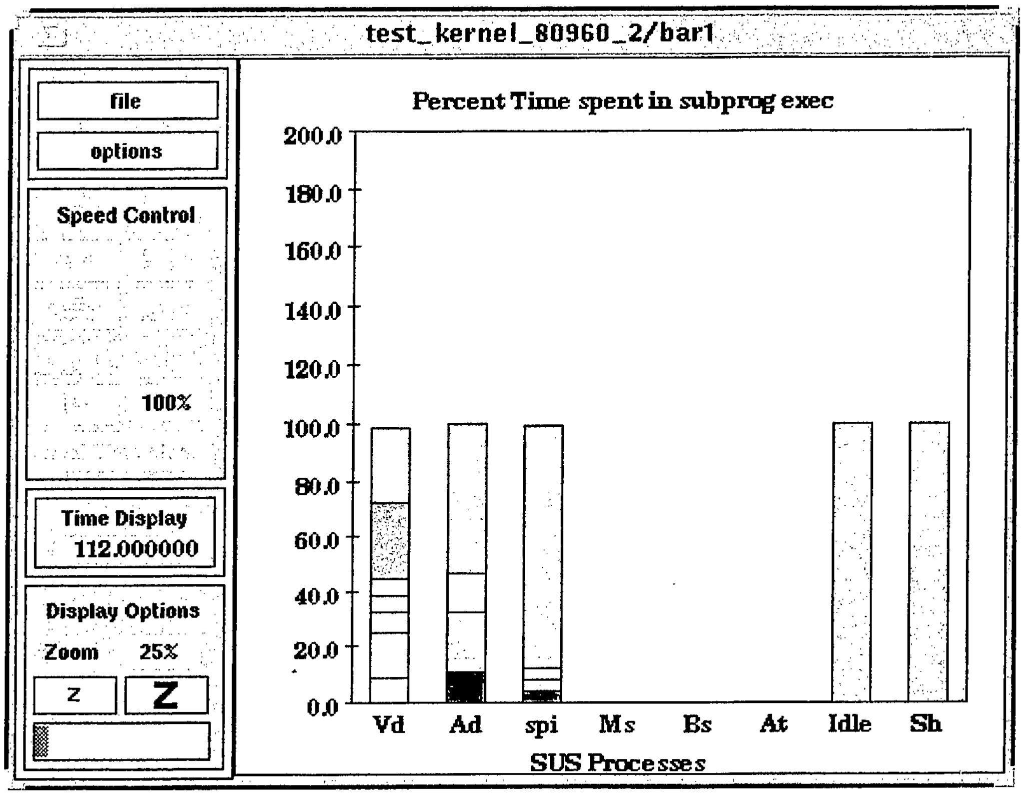


FIGURE 13 Intra-Process Subprogram Profiling

7. Performance Management for Avionics Software and Systems Development

During the ASPM program, alternative mechanisms for "performance management" were investigated and a report describing the methods and technologies used to develop and ensure the timing behavior of real-time systems was developed. The report [1] also describes a process-oriented perspective of performance modeling and the performance management methods that can be applied to each major step of the process. The following summarizes the results of the findings.

7.1 Performance Management Technologies

The basic technologies for performance management include performance modeling, both formal and simulation; technologies to estimate model parameters such as compute time; instrumentation and testing; performance verification; and the integrated use of these technologies. The following sections describe these technologies, the mechanisms available for applying the technologies, the strengths and benefits of each, and the results that can be obtained by applying them.

7.1.1 Performance Modeling and Analysis

The two major classes of performance models include the schedulability and the simulation models.

Schedulability Modeling and Analysis

A schedulability model is a formal model for which mathematical analyses exist to determine timing properties. Schedulability modeling and analysis methods are closely associated with specific design and implementation techniques. The strengths of schedulability modeling and analysis are:

- Mathematical assurance is provided that the analysis results will be true under all possible executions of a system, subject to the assumptions and parameters in the model.
- Schedulability models are typically relatively easy to develop.
- Schedulability analysis can typically be performed very rapidly, essentially in an interactive fashion.
- Schedulability analysis can produce sensitivity or parametric analysis results that give insight into system scalability and relationships between design elements.

The weakness of schedulability modeling and analysis is:

- Schedulability analysis is limited by the power of the underlying scheduling theory. It is not possible to model important aspects of many real systems.

The detailed overviews of schedulability models can be found elsewhere[2][6][7][20] . Preemptive fixed priority schedulability models describe the workload on a single processor as a collection of periodic and aperiodic (event-driven) tasks. Analysis can determine whether a specified system will always meet its deadline[9][8] , can determine the worst-case completion or response times of the individual tasks [5][3], and can determine the amount by which individual task compute times and blocking times could increase without causing the deadlines to be missed[10]. The MetaH toolset generates a preemptive fixed priority schedulability model from the input architectural specification (a type of design specification). Analysis is performed for each possible operating mode or configuration (MetaH supports dynamic reconfiguration), for each processor and inter processor channel.

Simulation Modeling and Analysis

The simulation models is concerned with performance simulation of real-time embedded computer systems. A performance simulation model is an executable high-level description of a system that captures those aspects that are felt to be significant in estimating timing and resource allocation and utilization characteristics. The model is executed in simulated time using several input and parameter values, yielding several execution traces with accompanying timing information.

The strengths of simulation are:

- Simulation modeling is powerful and flexible and can deal with behaviors that cannot be formally modeled and analyzed.
- Simulations are executable models that allow developers to experiment with, observe and debug system behavior in an intuitive way.

The weaknesses of simulation are:

- Simulation models may become quite intricate and require significant effort to develop and verify.
- Simulation can be computationally intensive with long turn-around times, especially for parametric and design trade-off studies, or studies requiring a high degree of confidence.
- It can be difficult to gain insight into how various parameters and modeling decisions interact, to understand the limits of system scalability, or to achieve high levels of confidence.

Simulation modeling is essentially a programming activity, with the associated characteristics of flexibility, power, difficulty, proneness to errors, and uncertainty. The most effective use of simulation probably relies on systematic, repeated use of modeling structures, methods and artifacts that have demonstrated effectiveness and accuracy for a particular product domain. SES/Workbench[12], Foresight[13] and Cosmos[14] are few commercial products that have been used for performance modeling of embedded real-time computer systems.

Estimating Model Parameters

The parameters of interest for performance models, including the sequences of scheduling events and their times (e.g., dispatch events, semaphore lock and unlock events); and the compute (execution) times of source code components on processors. In multiprocessor systems, message release and communication times are also important.

Dispatch times for periodic tasks are often derived during system design based on other engineering considerations (e.g., vehicle dynamics).

The compute time of a sequential, single-threaded source code component is the accumulated CPU time (number of processor cycles times cycle time) required to execute instructions from the entry to the exit of that source code component. Estimating the compute time parameter can be accomplished by using one of the five basic ways: conjecture based on experience and judgement, derived from algorithm operation counts, derived from compute times of similar components, derived from analysis of source code, and derived from test executions (usually called benchmarking).

7.1.2 Performance Instrumentation and Testing

Instrumentation and testing is the injection of selected test stimuli into an implementation and the measurement of selected behaviors. Two levels of performance instrumentation and testing is supported: source code component measurement (usually called benchmarking) and system instrumentation and testing.

The strengths of instrumentation and testing are:

- Data is obtained form the actual implementation, not from an approximate and possibly erroneous model.
- Direct observation, understanding, debugging and testing of the implementation are provided.

The weaknesses of instrumentation and testing are:

- Instrumentation can be intrusive and can perturb the behavior of the system.
- Implementations are not as controllable or observable as a model.
- Instrumentation data is available relatively late in the development process.
- Instrumentation alone provides little insight into why a system behaves the way it does.
- Instrumentation may require special relatively expensive equipment.

Most real-time operating system development environments include some form of performance instrumentation such as Stethoscope[16] and TimeScan[17]. Several benchmark suites are widely-available[18][11][19].

Component Benchmarking

Component benchmarking is used to measure the compute time of individual source code components. Component benchmarking requires the code to be callable from a benchmarking main program, and a choice of input data that gives the desired information about the distribution of compute times. The dual-loop benchmarking is a simple and widely-used method [4] for component benchmarking.

System Instrumentation and Testing

Integrated systems or subsystems can be instrumented and tested. The level of measurement corresponds to system structure and behavior in a performance model. The integration of MetaH and SPI toolset supports the gathering of two specific types of performance instrumentation. First, a scheduling time line for application processes and the executive/kernel can be displayed. This is useful in understanding system scheduling behavior, e.g., to compare instrumentation data with model behavior. Second, an execution profile can be displayed that shows the relative utilizations spent in each library-level unit in each process. This is useful in understanding the compute time behavior of source code components, e.g., to compare measured utilizations against model predictions at the source code component level.

7.1.3 Performance Verification

Performance can be tested directly using instrumentation; however, it is impossible to cover all possible system executions with any finite set of tests. Confidence in testing results depends on the degree of "test coverage" of requirements and design and implementation. Confidence also depends on assurance that the instrumentation and testing tools work properly and accurately. Performance modeling and analysis can significantly increase assurance that the timing behavior of a system is correct. The following are the three basic approaches for performance verification. First, performance testing can be performed. Second, the parameters used in, and analysis results obtained from, a performance model can be compared with values obtained by testing. Third, the mapping or traceability between the performance model and the design and implementation can be verified.

The primary consideration in performance testing is the determination of a good set of performance tests that can be carried out on the system. Performance testing may be limited by the controllability and observability of the system under test.

In a model-based testing approach, performance verification compares the results of performance model analysis with the results of performance testing. This approach directly compares the analysis results with implementation behavior and simultaneously validates the model and verifies the analysis results.

In the performance model comparative analysis approach the performance model is validated to explicitly describe the mapping between performance model, design and implementation and verify the correctness of mapping.

Performance tool verification is necessary since instrumentation and testing tools can have defects. In the case of performance instrumentation and testing, there are also inherent inaccuracies in time measurements. The advantage provided by model-based testing is a consistency check for each product and test, above and beyond any general and previous verification of the tools themselves.

7.1.4 Integrated Performance Management

Integrated performance management is the integrated production and use of designs, models, analyses, implementations and tests. Integration of design and modeling processes leads to models that are easily and accurately derived from the designs or vice versa. Integrated modeling and design can be done in a non-automated way simply by adapting the development process to tie design and performance model development together. Integration of the schedulability and simulation modeling and analyses, which have complementary strengths and weaknesses, leads to a system that leverages the strengths of each approach while overcoming weaknesses of individual method. The integration of these two methods depends on the actual design structure, which in turn affects the structure of the models. The MetaH/SPI integrated toolset provides both schedulability modeling and analysis and performance instrumentation in support of model-based performance testing.

7.2 Performance Management Process

The performance management methods can be applied to each major step of the process. The performance management process is described in the context of two widely-used software process guidelines, the *Capability Maturity Model for Software* and *Software Considerations in Airborne Systems and Equipment Certification.* For each phase of the software development, *Analysis and Design, Detailed Design and Component Implementation, Software and Systems Integration* and *Testing and Verification,* the performance management considerations and methods to be applied are reported.

Performance is usually a critical constraint for avionics systems, and performance modeling should be used to make complex trade-offs between functionality, quality, cost, size/weight/power, etc. The preliminary performance should be developed as part of the requirements analysis and high-level design activity. The results of parametric and sensitivity analysis generated from the analysis and design phase should be used to determine the order in which detailed designs and code components should be developed. In general, subsystems and components that analysis reveals are possible bottlenecks, or where relatively small variations in model details or parameter values produce relatively large variations in system performance, should be developed. The performance model should be refined and reverified (by comparative analysis) as the detailed design proceeds. During the software and systems integration phase, the software is typically deployed on a target hardware, such as a non-real-time workstation or a real-time testbed. Incremental refinement and verification should be performed during this stage.

8. Conclusion

During the course of the program various methods and technologies were investigated to ensure performance management of avionics application. Under this program the emphasis was placed on integration technology that leverages the strengths of the individual methods and technologies while trying to overcome the weaknesses of the methods. Under this program we integrated the schedulability modeling techniques with the system instrumentation and testing methods. The result is a suite of tools that simplifies and automates the tasks associated with code generation for the target hardware for an avionics system, the MetaH/SPI integrated toolset

The performance management data that is collected from the experiments are from actual implementation and not from an approximate mathematical or simulation. This data serves the means for verifying and checking the correctness of both the schedulability model and the instrumentation toolset. The toolset provides direct observation, understanding, debugging and testing of the underlying implementation. For a multi processor real-time avionics system having such a capability enables the user to debug and test a complex system comprising of interacting components.

The implementation architecture also was minimally invasive by having a dedicated processor for performing the instrumentation functionality and using a low overhead shared memory area for managing and signaling events from the application. The toolset also provided performance management support from the detailed design to the implementation resulting in a system that provides these data at early stages of the application development enabling the developers to make trade-off decisions at early stage of the lifecycle. It is our conclusion that the integrated toolset and the demonstration of application with stringent performance management requirements result in a powerful environment for conducting performance management experiments.

8.1 Lessons Learned

We can now summarize what we believe to be the most important lessons learned:

- The ability to have an environment comprising of integration of the MetaH toolset and SPI toolset provided considerable help in developing and testing the environment. The ability to cross-verify the results generated by static analysis tools supported by MetaH with the actual data collected from executing the experiment significantly improved the quality of results. This approach was used extensively during the course of the program to verify the correctness of the system.
- The ability to have a configurable environment where the experiments could be easily configured for different performance measurement experiment served very well. This provided the ability to overcome the significant bandwidth limitation that was experienced during the course of conducting the final demonstration of relaying the results from the SPI run-time to the SPI display systems. Different experiments, each configured with the ability to collect and display a subset of performance management data, provided the ability to overcome the network bandwidth limitation while still providing a comprehensive view of the performance management data.
- During the course of the program it became very clear that developing, testing and integrating new functionality in the integrated environment was a very challenging task because of the real-time nature of the application and the lack of sophisticated toolset for debugging an embedded application. Detecting problems with the underlying system was typically accomplished using brute-force methods and techniques. However, the ability to perform consistency check between results obtained from static analysis vs. the actual results provided helpful hints in identifying the source of the problem.
- The real-time nature of the experiment and the requirement for viewing the process-level scheduling information introduced significant loads on the network. For very fast rate processes the socket level communication over the ethernet between the VxWorks on 68030 and SUN workstation was the bottleneck in the system. Thus it became clear that utilizing technology with higher bandwidth of data transfer and faster processor was essential for using the toolset for applications with more stringent performance requirements.

9. Bibliography

1. Steve Vestal, "Real-Time Performance Management for Avionics Software and Systems Development," *ASPM Program,* 1997.

2. Neil C. Audsley, Alan Burns, Robert I. Davis, Ken W. Tendell and Andy J. Wellings "Fixed Priority Pre-emptive Scheduling: An Historical Perspective," *Real-Time Systems,* 8, 1995.

3. N.C. Audsley, A. Burns, M.F. Richardson and A.J. Wellings, "Hard Real-Time Scheduling: The Deadline Monotonic Approach," *Proceedings IEEE Workshop on Real-Time Operating Systems and Software,* 1991.

4. Russel M. Clapp, Louis Duchesneau, Richard A. Volz, Trevor M. Mudge and Timothy Schultze, "Toward Real-Time Performance Benchmarks for Ada," *Communications of the ACM,* August 1986.

5. J. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal,* 29, 5, 1986.

6. M.H.Klein, T. Ralya, B. Pollak, , R. Obenza and G.H. harbour, "A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems," *Kluwer Academic Publishers, Boston, MA* 1993.

7. Mark H. Klein, John P. Lehoczky and Ragunathan Rajkumar, "Rate-Monotonic Analysis for Real-Time Industrial Computing," *IEEE Computer,* January 1994.

8. J. P. Lehoczky, L.Sha and Y. Ding, "The Rate-Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *Proceedings IEEE Real-Time Systems Symposium,* 1989.

9. C. L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *JACM* (20)1, 1993.

10. Steve Vestal, "Fixed Priority Sensitivity Analysis for Linear Compute Time Model," *IEEE Transactions on Software Engineering,* April 1994.

11. Nelson Weiderman, Patrick Donohoe and Ruth Shapiro, "Benchmarking for Deadline-Driven Computing," *Tri-Ad'90,* Baltimore, MD, December 1990.

12. SES/Workbench, Scientific and Engineering Software, Inc., http://www.ses.com/

13. Foresight, Nu Thena Systems, Inc. http://mozart.nuthena.com/products/fsOverview/index.html

14. Cosmos, Omniview, Inc. http://www.omnivw.com/products/cosmos.html

15. *MetaH Programmer's Manual,* Honeywell Technology Center, Minneapolis, MN 1996

16. Stethoscope, Wind River, Inc., http://www.wrs.com/html/stethosc.com

17. TimeScan, Lynx, Inc., http://www.lynx.com/products/timescan.html

18. *Ada Letters, Special Edition from the SIGAda Performance Issues Working Group on Ada Performance Issues,* X(3), Winter 1990, see also http://info.acm.org/sigada/wg/piwg/piwg.html

19. Performance Database Server Web, http://performance.netlib.org/performance/html/PDStop.html

20. "Rate Monotonic Analysis," SEI on-line Software Technology Review, http://www.sei.cmu.edu/technology/str/descriptions/rma.html