

QUALITY OF SERVICE MANAGEMENT FOR REAL-TIME EMBEDDED INFORMATION SYSTEMS¹

Christopher D. Gill and David L. Levine, [\[cdgill,levine}@cs.wustl.edu](mailto:{cdgill,levine}@cs.wustl.edu)

Center for Distributed Object Computing, Department of Computer Science

Washington University, St. Louis, MO

Introduction

Next generation real-time embedded information systems, such as avionics mission computing systems, must adapt swiftly to changing environmental conditions. Greater coordination between command functions, groups in the field, and individual systems increases shared awareness of the evolving environment, allowing elements at all levels to leverage information to identify transient opportunities and set in motion activities to exploit those opportunities.

Achieving this level of coordination requires:

(1) systems with an empirically demonstrated ability to accommodate unplanned tasks and changing task priorities in an evolving distributed information and resource availability environment; (2) the ability to share information and control at multiple scales of distribution and timeliness; and (3) supporting infrastructure to manage resources effectively across both distribution and time-scale boundaries.

The ability to manage information and resources dynamically, both locally and globally, requires several enhancements to current systems including: (1) flexible “modeless” execution; (2) support for variable period tasks; (3) remote access to real-time information; (4) dynamic adjustment to network and execution loads over longer time scales; and (5) rapid local adaptation to variable system resource availability. To realize these benefits, systems must be developed and deployed with greater application of real-time adaptive software concepts [1], both in the system framework and in the specific application components themselves.

Our recent work has explored further how dynamic and distributed Quality-of-Service (QoS) management functions can be added to avionics applications built using Commercial-Off-The-Shelf (COTS) standards and components, to provide more powerful adaptive software capabilities. This paper describes contributions in two principal areas. First, it outlines a QoS management architecture that meets the distributed resource management needs of real-time information systems, and describes our recent extensions to that architecture. Second, it presents empirical evidence of the utility and feasibility of dynamic and adaptive system behavior in realistic real-time embedded information systems. The discussion centers on the identification of key architectural features, and describes initial qualitative and quantitative results that are used to assess the benefits and costs of these segments of the overall architecture.

Architecture

We have continued to refine and extend a CORBA-based [2] dynamic and adaptive real-time QoS management architecture whose previous evolutionary stages are described in [3] and [1]. The essential QoS management segments of the architecture are as follows:

- Application components designed to have minimal dependency on variable timing characteristics
- Adaptive rate selection for both critical and non-critical operations
- Real-time deadline feasibility protection for critical operations

¹ This work was supported by The Boeing Company, DARPA contract 9701516, Sprint, and Siemens AG

- Real-time operation deadline failure prediction and detection with possible cancellation of predicted deadline-infeasible dispatches
- Flexible hybrid static/dynamic operation scheduling in realistic real-time embedded information systems.

Our recent research has focused primarily on evolving the last three segments of the architecture, though we have also begun work to improve real-time performance of adaptive rate selection as well. This evolution consists of two main areas: (1) developing new performance measurement and management features, and (2) empirically assessing the performance of these features and the overall framework.

First, we have added key features for both operating and measuring performance of our real-time QoS management architecture in a realistic real-time embedded environment. These features include run-time deadline monitoring and operation cancellation, and supporting infrastructure for real-time capture and distribution of performance data, without interfering with the real-time behavior of the measured system.

Second, we have conducted a set of experiments, described in the next section, to demonstrate the efficacy and suitability of the architecture in a realistic real-time embedded information system environment. These experiments in turn serve to demonstrate both the capabilities of the QoS management architecture in a realistic real-time embedded information system environment, and the capabilities of the experimental monitoring and visualization framework.

Empirical Results

The focus of these experiments was to quantify the benefits and costs of scheduling real-time embedded information systems using hybrid static/dynamic approaches, when compared to statically scheduled systems. Our hypothesis is that hybrid approaches, though they can incur additional runtime overhead, will prove overall to be more flexible, both in terms of application development ease and overall computational throughput.

Ease of application development is facilitated by two adaptive properties of hybrid static/dynamic scheduling: (1) when load exceeds the schedulable bound, non-critical operations are dropped, whereas critical operations are scheduled; (2) dynamic scheduling supports selectively dropping non-critical operations that will miss deadlines, while preserving non-critical operations that might be schedulable later. Encapsulating fine-grain adaptive control over operation dispatching in the middleware layers relieves developers of tedious, error-prone, and often redundant tasks related to developing this aspect of their applications.

Increased computational throughput is achieved through greater processor utilization compared to static systems, which generally require under-utilization of the CPU to be feasibly schedulable. Here too, hybrid static/dynamic scheduling provides fine-grain adaptive control over operation dispatching so that more operations can be scheduled to increase CPU utilization. Moreover, dropping operation dispatch requests that will not meet their QoS requirements has the potential to improve the amount of useful computation that is performed, though it is essential to also take into account the potential overhead associated with operation cancellation.

Below, we describe the experimental configuration used to investigate these issues. We also report the results of qualitative visualizations and quantitative benchmarks that support key aspects of our hypothesis outlined above.

Experimental Configuration

Our experiment used a complete real-time embedded information systems application, with roughly 30 scheduled dispatching entry points. The application ran using the TAO ORB [4], the TAO Scheduling Service [5], and the TAO Real-Time Event Service [6], configured for various scheduling strategies.

We conducted measurements on two key areas of QoS management: (1) stability of operations in meeting their deadlines, and (2) stability of operation execution times. The analysis below features a comparison of two publicly available scheduling algorithms, Maximum Urgency First (MUF) [7] and Rate Monotonic Scheduling (RMS)

[8]. Measurements were conducted on Single CPU 200 MHz Power PC Single Board Computers running the VxWorks 5.3 operating system.

For each scheduling algorithm, the application cycled repeatedly through a sequence of alternating *operating regions*, each region offering a different range of utilization loads to the embedded CPU. The combined utilization for critical hard real-time (HRT) operations in each region was always within the schedulable bound. Varying utilization by shorter duration non-critical soft real-time (SRT) operations were then added to produce a total utilization that fell within the schedulable bound in some operating regions, and exceeded it in others.

Deadline Stability Comparison

These single board computers were connected via a VME back plane, and the master computer also had an Ethernet connection to a visualization computer running the Windows NT Workstation 4.0 operating system. The visualization framework is used here in a way similar to that described previously in [9]. The Distributed Object Visualization Environment (DOVE [10]) framework consists of Java visualization components connected to a CORBA-based framework for capturing and transmitting data from the running experiment to the visualization workstation. The DOVE configuration used in these experiments is illustrated in Figure 1.

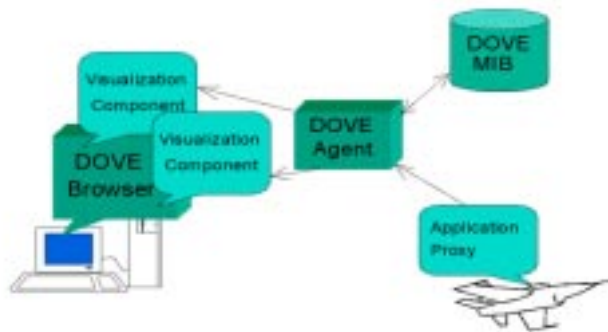


Figure 1. Distributed Object Visualization Environment (DOVE)

Figures 2 through 5 present a series of visualization images captured during the quantitative embedded experiments. The data collected in the experiment were fed in real-time to

the visualization displays. Each image shows a pair of plots over time showing the number of HRT operations and the number of SRT operations that either made or missed their deadlines. Each image reflects a comparable period of time, spanning the same transition from an operation region that already offered a total utilization load in excess of the CPU's schedulable bound, into an operating region that was even more overloaded. The number and utilization of HRT operations remained the same between the operating regions, but the number of SRT operations increased from the earlier region to the later one. No operation cancellation was applied during the experimental phase from which these visualizations were captured.

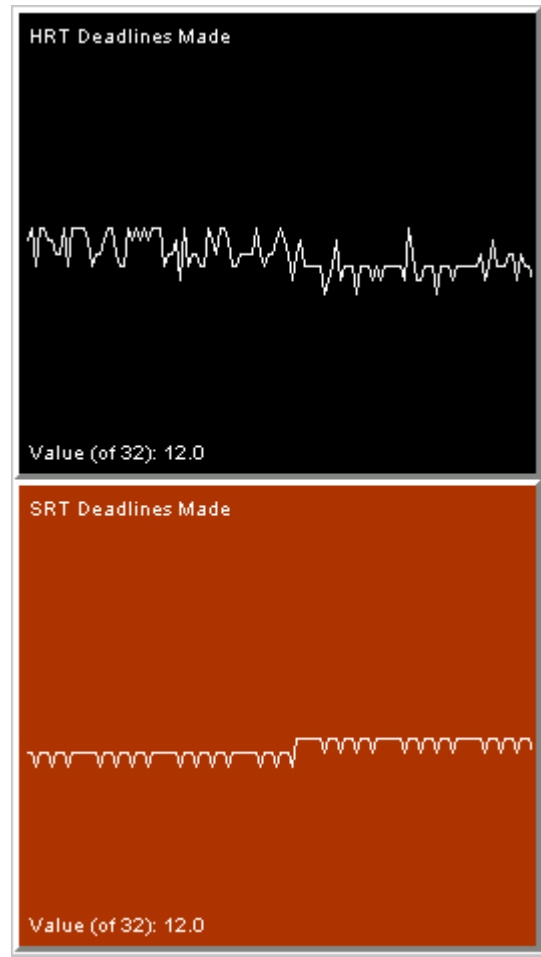


Figure 2. Operation Deadlines Made: RMS

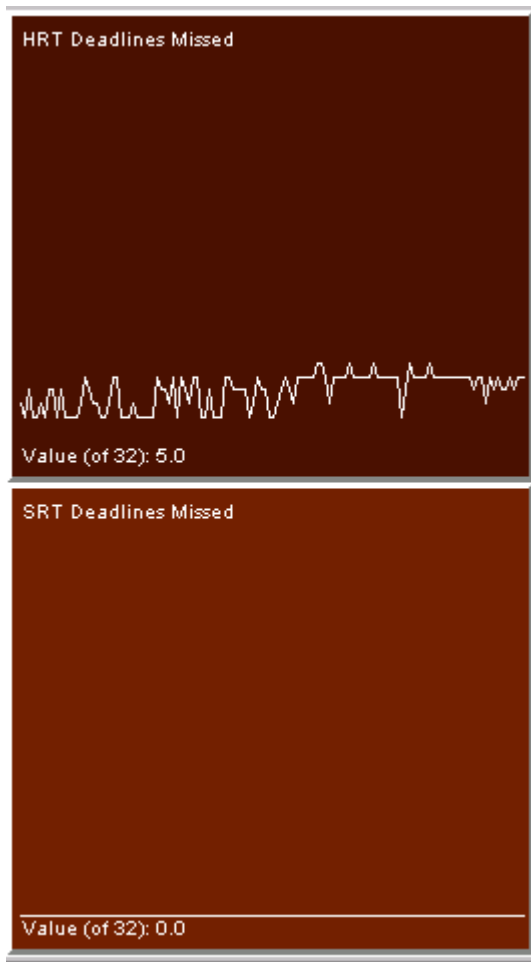


Figure 3. Operation Deadlines Missed: RMS

Figure 2 shows the number of HRT and SRT deadlines made under the classic static RMS scheduling algorithm. The lower panel of the image shows the shorter-running non-critical SRT operations competing more effectively for the CPU, with a clear transition point at the boundary between operating regions. The curve is reasonably flat² and regular, indicating highly successful and deterministic deadline timing behavior for SRT operations in both regions. The HRT operations shown in Figure 2, on the other hand, suffered significant degradation both in the number and in the predictability of deadlines made. Ideally, the curve should be reasonably flat³ and have a

² The small spikes in the curve reflect inter-frame variations in the number of operation dispatches that were requested, due to period differences between the operations.

³ Again, taking into account inter-frame request variations.

sustained maximum value of 18. The diminished amplitude and jagged shape of the HRT curve indicates high jitter in the deadline behavior of HRT operations. Thus, the RMS scheduling strategy failed to protect the longer running critical HRT operations from competition with the shorter running non-critical SRT operations in these experiments.

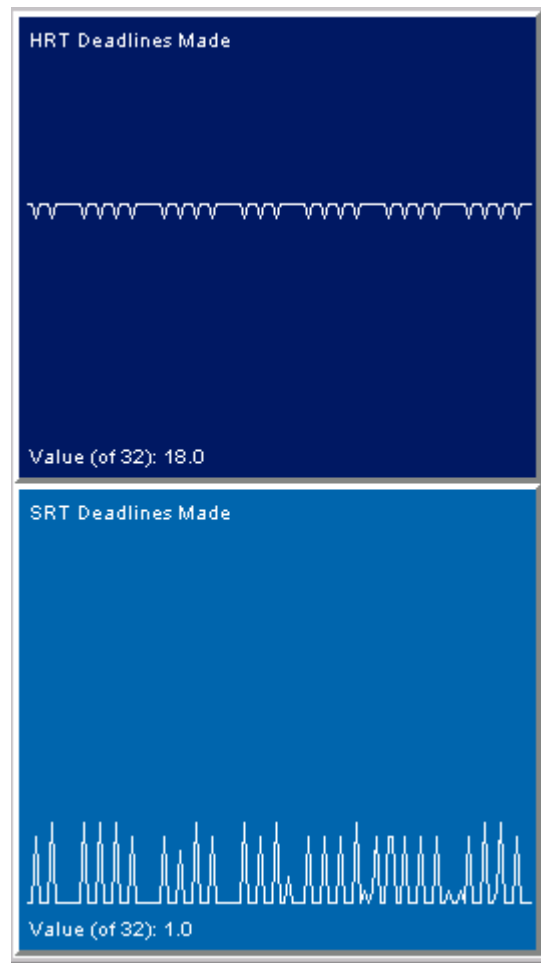


Figure 4. Operation Deadlines Made: MUF

Figure 3 shows the corresponding plots of HRT and SRT operation deadlines missed during this same visualization phase. No SRT operation deadlines were missed, and again the number of HRT deadlines missed was significant for both its increased magnitude⁴ and its jagged shape.

⁴ Ideally, its value should be consistently zero.

Figure 4 shows the number of deadlines made for HRT and SRT operations during the same state transition, but with the MUF scheduling algorithm instead of RMS. The HRT deadline behavior shows no degradation in either operating region. The number and predictability of HRT deadlines made are consistently high. Interestingly, while in Figure 4 the SRT operations show degradation in the number and predictability of operation deadlines made, the pattern is much more regular than that for the HRT operations under RMS in Figure 2.

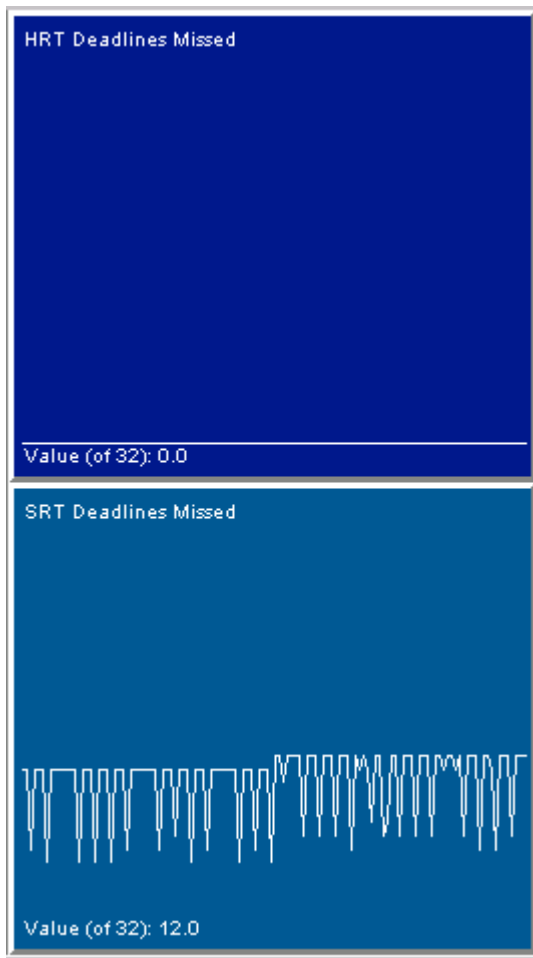


Figure 5. Operation Deadlines Missed: MUF

Finally, Figure 5 shows the corresponding pattern of missed deadlines for the same visualization period under the MUF scheduling algorithm. There were no missed deadlines for HRT operations. Again, SRT operations showed a

more regular pattern of missed deadlines than that for the missed HRT operation deadlines under RMS in Figure 3.

MUF Execution Time Stability

In addition to transporting data to the visualization workstation for qualitative analysis, the experimental configuration also stored a large amount of raw operation timing data for later quantitative analysis. This is useful for addressing questions arising from the experimental results, but for which visualizations were not conducted during the original experiments.

In particular, the success of the MUF scheduling algorithm in protecting the critical HRT operations from competition from non-critical SRT operations begs the question of whether this occurs at a cost to other real-time properties, such as deterministic operation execution times. Figure 6 shows that for MUF, both critical HRT operations and non-critical SRT operations showed reasonable performance in both the value and determinism of their execution times. These results correlate with the regular pattern of made and missed HRT and SRT operation deadlines from the MUF algorithm visualizations, even under conditions of utilization overload.

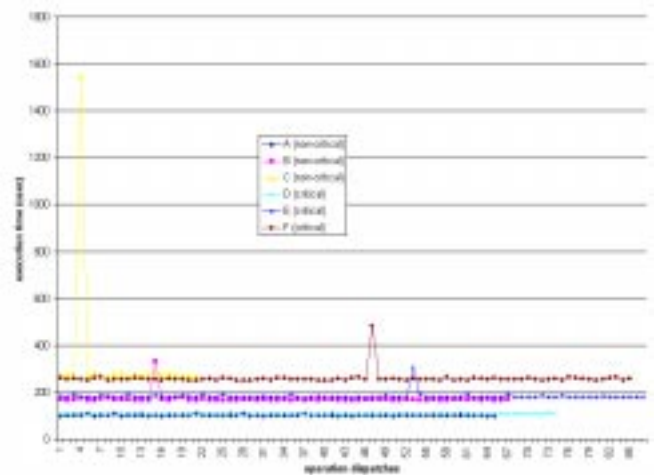


Figure 6. Critical and Non-critical Operation Execution Times: MUF

Concluding Remarks

The results presented in this paper offer empirical evidence of the suitability and benefits of applying adaptive and dynamic QoS management techniques to real-time embedded information systems. Applying these techniques to next generation real-time embedded information systems will offer greater flexibility for appropriate responses to rapidly evolving situational factors, while retaining crucial real-time properties necessary to system integrity.

Acknowledgments

We would like to thank Bryan Doerr and Greg Holtmeyer for their significant contributions to this research.

References

- [1] Doerr, Bryan, Thomas Venturella, Rakesh Jha, Christopher Gill, Douglas Schmidt, 1999, *Adaptive Scheduling for Real-time Embedded Information Systems*, St. Louis, MO, 18th IEEE/AIAA DASC
- [2] 1998, *The Common Object Request Broker: Architecture and Specification*, Object Management Group, Edition 2.2
- [3] Levine, David, Christopher Gill, Douglas Schmidt, 1998, *Dynamic Scheduling for Avionics Applications*, Seattle, WA, 17th IEEE/AIAA DASC.
- [4] Schmidt, Douglas, David Levine, and Sumedh Mungee, 1998, *The Design and Performance of Real-Time Object Request Brokers*, Computer Communications, Elsevier, vol.21, no. 4, pp. 294-324
- [5] Gill, Christopher, David Levine, Douglas Schmidt, 2000 (to appear), *The Design and Performance of a Real-Time CORBA Scheduling Service*, International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware, Kluwer Academic Publishers
- [6] Harrison, Timothy, David Levine, Douglas Schmidt, 1997, *The Design and Performance of a Real-time CORBA Event Service*, OOPSLA '97, ACM
- [7] Stewart, David, Pradeep Khosla, 1992, *Real-Time Scheduling of Sensor-Based Control Systems*, Tarrytown, NY, Real-Time Programming, Pergamon Press
- [8] Liu, C., J. Layland, 1973, *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*, JACM, vol. 20, no. 1
- [9] Gill, Christopher, David Levine, Carlos O’Ryan, Douglas Schmidt, 1999, *Distributed Object Visualization for Sensor-Driven Systems*, St. Louis, MO, 18th IEEE/AIAA DASC
- [10] Kircher, Michael, Douglas Schmidt, 1999, *DOVE: A Distributed Object Visualization Environment*, C++ Report, vol. 11, no. 2