DYNAMIC SCHEDULING STRATEGIES FOR AVIONICS MISSION COMPUTING

David L. Levine, Christopher D. Gill, and Douglas C. Schmidt
{levine,cdgill,schmidt}@cs.wustl.edu
Department of Computer Science, Washington University, St. Louis, MO 63130, USA [*]

To be presented at the 17th IEEE/AIAA Digital Avionics Systems Conference, Seattle, Washington, 31 October – 6 November, 1998.

Abstract

Avionics mission computing systems have traditionally been scheduled statically. Static scheduling provides assurance of schedulability prior to run-time and can be implemented with low run-time overhead. However, static scheduling handles non-periodic processing inefficiently, and treats invocation-to-invocation variations in resource requirements inflexibly. As a consequence, processing resources are underutilized and the resulting systems are hard to adapt to meet worst-case processing requirements.

Dynamic scheduling has the potential to offer relief from some of the restrictions imposed by strict static scheduling approaches. Potential benefits of dynamic scheduling include better tolerance for variations in activities, more flexible prioritization, and better CPU utilization in the presence of non-periodic activities. However, the cost of these benefits is expected to be higher run-time scheduling overhead and additional application development complexity. This report reviews the implications of these tradeoffs for avionics mission computing systems and presents experimental results obtained using the Maximum Urgency First dynamic scheduling algorithm.

1 Introduction

1.1 Motivation

Supporting the quality of service (QoS) demands of next-generation real-time applications requires object-oriented (OO) middleware that is flexible, efficient, predictable, and convenient to program. Applications with deterministic real-time requirements, such as avionics mission computing systems, impose severe constraints on the design and implementation of real-time OO middleware. Avionics mission computing applications manage sensors and operator displays, navigate the aircraft's course, and control weapon release.

Middleware for avionics mission computing must support applications with both deterministic and statistical real-time QoS requirements. Support for deterministic real-time requirements is necessary for mission computing tasks that must meet all their deadlines, *e.g.*, weapon solutions and navigation. Support for statistical real-time requirements is desirable for tasks such as built-in-test and low-priority display queues, which can tolerate minor fluctuations in scheduling and reliability guarantees, but nonetheless require QoS support.

[*]This work was supported in part by Boeing, DARPA contract 9701516, and NSF grant NCR-9628218.

1.2 Design and Implementation Challenges

Figure 1 illustrates the architecture of an avionics mission computing application developed at Boeing [1] using OO middleware components and services based on the Object Management Group's Common Object Request Broker Architecture (CORBA) [2]. CORBA Object Request Brokers

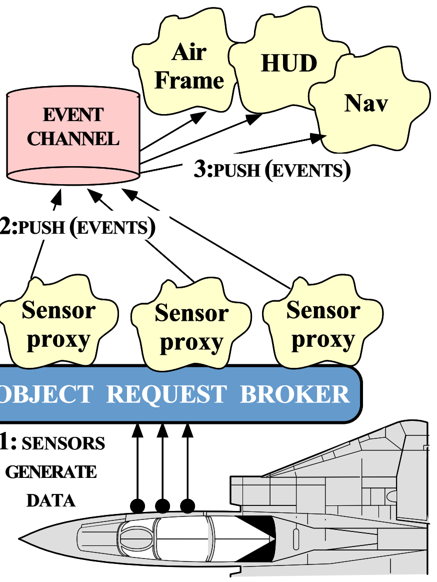

Figure 1: Example Avionics Mission Computing Application

(ORBs) allow clients to invoke operations on target object implementations without concern for where the object resides, what language the object is written in, the OS/hardware platform, or the type of communication protocols and networks used to interconnect distributed objects [3]. To achieve these benefits for avionics applications, however, requires the resolution of the following design and implementation challenges:

Scheduling assurance prior to run-time: In avionics applications, the consequences of missing a critical deadline at run-time can be catastrophic. For example, failure to process an input from the pilot within the necessary time frame could be disastrous, especially in critical situations such as air-to-air engagement or weapons release. Therefore, it is essential to validate that all critical processing deadlines will be met *prior* to run-time.

Severe resource limitations: Processing must be minimized due to limited resource availability, such as weight and power consumption restrictions. A consequence of using static, off-line scheduling is that worst-case processing requirements drive the schedule. Therefore, resource allocation and scheduling must always accommodate the worst case, even in non-worst case scenarios.

Distributed Processing: In complex avionics systems, mission processing must be distributed over several physical processors and computations on separate processors must communicate effectively. Clients running on one processor must be able to invoke operations on servants in other processors. Likewise, the allocation of operations to processors should be flexible, *e.g.*, it should be transparent whether a given operation resides on the same processor as the client that invokes it.

Testability: Avionics software is complex, critical, and long-lived. Maintenance is particularly problematic and expensive [4]. A large percentage of software maintenance involves testing. Current scheduling approaches are validated by extensive testing, which is tedious and non-comprehensive. Thus, analytical assurance is essential to help reduce validation costs by focusing the requisite testing on the most strategic system components.

Adaptability across product families: Current avionics applications are custom-built for a specific product family. Development and testing costs can be reduced if large, common portions can be factored out. In addition, validation and certification of components can be shared across product families, potentially reducing development time and effort.

The remainder of this paper is organized as follows: Section 2 reviews the drawbacks of off-line, static scheduling and introduces the dynamic scheduling strategy we are evaluating, *Maximum Urgency First* (MUF) [5]. Section 3 presents experimental results showing the cost of dynamic scheduling. Section 4 presents concluding remarks.

2 Dynamic Scheduling Strategies

This section describes the limitations of purely static scheduling and outlines the potential benefits of applying dynamic scheduling. We also evaluate the limitations of purely dynamic scheduling strategies. This evaluation motivates the hybrid static/dynamic MUF scheduling approach for CORBA operations used by TAO's real-time scheduling service (described in [6]).

2.1 Limitations of Static Scheduling

Many hard real-time systems have traditionally been scheduled statically using rate monotonic scheduling (RMS). Static scheduling provides schedulability assurance prior to run-time and can be implemented with low run-time overhead [6]. However, static scheduling has these disadvantages:

Inefficient handling of non-periodic processing: Static scheduling treats aperiodic processing as if it was periodic, *i.e.*, occurring at its maximum possible rate. Resources are allocated to aperiodic operations either directly or through a sporadic server[1] to reduce latency. In typical operation, however, aperiodic processing may not occur at its maximum possible rate. One example is interrupts, which potentially may occur very frequently, but often do not.

Unfortunately, with static scheduling, resources must be allocated pessimistically and scheduled under the assumption that interrupts occur at the maximum rate. When they do not, utilization is effectively reduced because unused resources cannot be reallocated.

Utilization phasing penalty for non-harmonic periods: In statically scheduled systems, achievable utilization can be reduced if the periods of all operations are not harmonically related. Operations are harmonically related if their periods are integer multiples of one another. When periods are not harmonic, the phasing of the operations produces unscheduled gaps of time. This reduces the maximum schedulable percentage of the CPU, *i.e.*, the *schedulable bound*, to below unity.

Inflexible handling of invocation-to-invocation variation in resource requirements: Because priorities cannot be changed easily[2] at run-time, allocations must be based on worst-case conditions. Thus, if an operation usually requires 5 msec of CPU time, but under certain conditions requires 8 msec, static scheduling analysis must assume that 8 msec will be required for every invocation. Again, utilization is effectively penalized because the resource will be idle for 3 msec in the usual case.

In general, static scheduling limits the adaptability of systems to changing conditions and changing configurations. In addition, static scheduling compromises resource utilization in order to guarantee access to resources at run-time. To overcome the limitations of static scheduling, therefore, we are investigating the use of dynamic strategies to schedule CORBA operations for applications with real-time QoS requirements.

[1]A sporadic server [7] reserves a portion of the schedule to allocate to aperiodic events when they arrive.

[2]Priorities can be changed via *mode changes*, but that is too coarse to capture invocation-to-invocation variations in the resource requirements of complex applications.

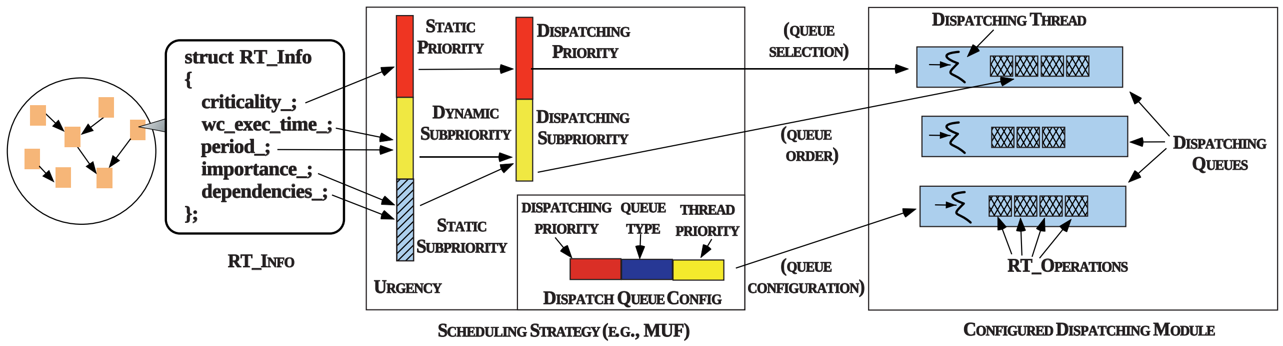

Figure 2: Relationships Between Operation, Scheduling, and Dispatching Terms

2.2 Synopsis of Scheduling Terminology

Precise terminology is necessary to discuss and evaluate static, dynamic, and hybrid scheduling strategies in terms of their priority assignment and dispatching characteristics. Figure 2 shows the relationships between key terms, defined below.

RT_Operation and RT_Info: In TAO, an `RT_Operation` is a scheduled CORBA operation [6]. In this paper, we use *operation* interchangeably with `RT_Operation`. An `RT_Info struct` is associated with each operation and contains its QoS parameters. The `RT_Info` structure contains the following operation characteristics described in [6]:

- **Criticality:** Criticality is an application-supplied value that indicates the significance of a CORBA operation's completion prior to its deadline. Higher criticality should be assigned to operations that incur greater cost to the application if they fail to complete execution before their deadlines. Some scheduling strategies, such as MUF, take criticality into consideration, so that more critical operations are given priority over less critical ones.

- **Worst Case Execution Time:** This is the longest time it can take to execute a single dispatch of the operation.

- **Period:** Period is the interval between dispatches of an operation.

- **Importance:** Importance is a lesser indication of a CORBA operation's significance. Like its criticality, an operation's importance value is supplied by the application. Importance is used as a tie-breaker to distinguish between operations that otherwise would have identical priority.

- **Dependencies:** An operation depends on another operation if it is invoked only via a flow of control from the other operation.

Scheduling Strategy: A scheduling strategy (1) takes the information provided by an operation's `RT_Info`, (2) assigns urgency (and thus its components, static priority, dynamic subpriority and static subpriority) to the operation, (3) maps urgency into dispatching priority and dispatching subpriority values for the operation, and (4) provides dispatching queue configuration information so that each operation can be dispatched according to its assigned dispatching priority and dispatching subpriority. The key elements of this transformation performed by the scheduling strategy, which is shown in Figure 2, are as follows:

- **Urgency:** Urgency [8] is an ordered tuple consisting of (1) static priority, (2) dynamic subpriority, and (3) static subpriority. Static priority is the highest ranking priority component in the urgency tuple, then dynamic subpriority, and last static subpriority. Figure 2 illustrates these relationships.

- **Static priority:** Static priority assignment establishes a fixed number of priority partitions into which all operations must fall. The number of static priority partitions is established off-line. An operation's static priority value is often determined off-line. However, the value assigned a particular dispatch of the operation could vary at run-time, depending on the scheduling strategy.

- **Dynamic subpriority:** Dynamic subpriority is a value generated and used at run-time to order operations *within* a static priority level, according to the run-time and static characteristics of each operation. For example, a subpriority based on nearest deadline must be computed dynamically.

- **Static subpriority:** Static subpriority values are determined prior to run-time. Static subpriority acts as a tie-breaker when both static priority and dynamic subpriority are equal.

- **Dispatching priority:** An operation's dispatching priority corresponds to the real-time priority of the thread in which it will be dispatched. Operations with higher dispatching priorities are dispatched in threads with higher real-time priorities.

• **Dispatching subpriority:** Dispatching subpriority is used to order operations within a dispatching priority level. Operations with higher dispatching subpriority are dispatched ahead of operations with the same dispatching priority but lower dispatching subpriority.

• **Queue Configuration:** A separate queue must be configured for each distinct dispatching priority. The scheduling strategy assigns each queue a dispatching type (static, deadline, or laxity[3]), a dispatching priority, and a thread priority.

Dispatching Module: The dispatching module (1) constructs the appropriate type of queue for each dispatching priority, and (2) sets each dispatching thread's priority to the value provided by the scheduling strategy. A TAO ORB endsystem can be configured with dispatching modules in the I/O subsystem, ORB Core, and/or the Event Channel.

2.3 Survey of Dynamic Scheduling Strategies

Several other forms of scheduling exist beyond RMS. For instance, Earliest Deadline First (EDF) scheduling assigns higher priorities to operations with closer deadlines. EDF is commonly used for dynamic scheduling because it permits run-time modification of rates and priorities. In contrast, static techniques like RMS require fixed rates and priorities.

Dynamic scheduling does not suffer from the drawbacks described in Section 2.1. If these drawbacks can be alleviated without incurring too much overhead or non-determinism, dynamic scheduling can be beneficial for avionics applications. However, many dynamic scheduling strategies do not offer the *a priori* guarantees of static scheduling. For instance, purely dynamically scheduled systems can behave non-deterministically under heavy loads. Therefore, operations that are critical to an application may miss their deadlines because they were (1) delayed by non-critical operations or (2) delayed by an excessive number of critical operations, *e.g.*, if admission control of dynamically generated operations is not performed.

The remainder of this section reviews several strategies for dynamic and hybrid static/dynamic scheduling. These include purely dynamic strategies such as EDF and MLF, and hybrid approachs such as MUF and two-level scheduling.

2.3.1 Purely Dynamic Scheduling Strategies

Earliest Deadline First (EDF): EDF [9, 10] is a dynamic scheduling algorithm that orders dispatches[4] of operations based on time-to-deadline. Operation executions with closer deadlines are dispatched before those with more distant deadlines. The EDF scheduling algorithm is invoked whenever a dispatch of an operation is requested. The new dispatch may or may not preempt the currently executing operation, depending on the implementation strategy.

[3]An operation's laxity is the time until its deadline minus its remaining execution time.

[4]A *dispatch* is a particular execution of an *operation*.

A key limitation of EDF is that an operation with the earliest deadline is dispatched whether or not there is sufficient time remaining to complete its execution prior to the deadline. Therefore, the fact that an operation cannot meet its deadline will not be detected until *after* the deadline has passed.

If the operation is dispatched even though it cannot complete its execution prior to the deadline, the operation consumes CPU time that could otherwise be allocated to other operations. If the result of the operation is only useful to the application prior to the deadline, then the entire time consumed by the operation is essentially wasted.

Minimum Laxity First (MLF): MLF [8] refines the EDF strategy by taking into account operation execution time. It dispatches the operation whose *laxity* is least. Laxity is defined as the time-to-deadline minus the remaining execution time.

Using MLF, it is possible to detect that an operation will not meet its deadline prior to the deadline itself. If this occurs, a scheduler can reevaluate the operation prior to allocating the CPU for the remaining computation time. For example, one strategy is to simply drop the operation whose laxity is not sufficient to meet its deadline. This strategy could decrease the chance that subsequent operations will miss their deadlines, especially if the system is transiently overloaded.

Evaluation of EDF and MLF:

• **Advantages:** From a scheduling perspective, the main advantage of EDF and MLF is that they overcome the utilization limitations of RMS. In particular, the utilization phasing penalty described in Section 2.1 that can occur in RMS is not a factor. This is because EDF and MLF prioritize operations according to their dynamic run-time characteristics. They handle harmonic and non-harmonic periods comparably, and respond flexibly to invocation-to-invocation variations in resource requirements, allowing CPU time one operation does not use to be reallocated to other operations. Thus, they can produce schedules that are optimal in terms of CPU utilization [9]. Moreover, both EDF and MLF can dispatch operations within a single static priority level and do not prioritize operations by rate [9, 8].

• **Disadvantages:** Purely dynamic scheduling approaches like MLF and EDF potentially relieve the utilization limitations of the static RMS approach. However, they have a higher cost to evaluate the scheduling algorithm at run-time. In addition, these purely dynamic scheduling strategies offer no control over *which* operations will miss their deadlines if the schedulable bound is exceeded. As operations are added to the schedule to achieve higher utilization, the margin of safety for *all* operations decreases. Therefore, the risk of missing a deadline increases for every operation as the system become overloaded.

2.3.2 Maximum Urgency First

The Maximum Urgency First (MUF) [8] scheduling algorithm supports both the deterministic rigor of the static RMS scheduling approach and the flexibility of dynamic scheduling approaches such as EDF and MLF. RMS assigns all priority components statically and EDF/MLF assign all priority components dynamically. In contrast, MUF can assign both static *and* dynamic priority components. The hybrid priority assignment in MUF overcomes the drawbacks of the individual scheduling algorithms by combining techniques from each. The remainder of this section describes how each characteristic of the MUF scheduling algorithm helps to achieve this.

Criticality: In MUF, operations with higher *criticality* are assigned to higher static priority levels. Assigning static priorities according to criticality prevents operations critical to the application from being preempted by non-critical operations.

Ordering operations by application-defined criticality reflects a subtle and fundamental shift in the notion of priority assignment. RMS, EDF, and MLF (1) exhibit a rigid mapping from empirical operation characteristics to a single priority value, and (2) offer scheduling guarantees only for all operations or none. In contrast, MUF (1) gives applications the ability to distinguish operations arbitrarily, (2) can offer scheduling guarantees for a critical *subset* of the entire set of operations.

Dynamic Subpriority: At the instant of evaluation, dynamic subpriority in MUF is a function of the the laxity of an operation.

Assigning dynamic subpriorities according to laxity (1) offers higher utilization of the CPU than static approaches, and (2) allows deadline failures can be detected *before* they actually occur, except when an operation that would otherwise meet its deadline is preempted by a higher criticality operation. Moreover, MUF can apply various types of error handling policies when deadlines are missed [8]. For example, if an operation has negative laxity prior to being dispatched, it can be demoted in the priority queue, so an operation that can still meet its deadline can be dispatched instead.

Static Subpriority: In MUF, *static subpriority* is a static, application-specific, optional priority. Static subpriority has lower precedence than either criticality or dynamic subpriority. It is used to order the dispatches of operations that have the same criticality and the same dynamic subpriority. Assigning a unique static subpriority to each operation that has the same criticality ensures a total dispatching ordering of operations at run-time, for any operation laxity values having the same criticality. A total dispatching ordering ensures that for a given arrival pattern of operation requests, the dispatching order will always be the same. This in turn improves the reliability and testability of the system.

The variant of MUF used in TAO's strategized scheduling service enforces a complete dispatching ordering by providing an `importance` field in the TAO `RT_Info` CORBA operation QoS description `struct` [6]. TAO's scheduling service uses importance, as well as a topological ordering of operations, to assign a unique static subpriority for each operation within a given criticality level.

2.3.3 Hybrid Approaches

Hybrid static and dynamic approaches may be used to combine the benefits of both. Multi-level scheduling integrates different approaches at different scheduling *levels*. One example is two-level hierarchical scheduling, which allows real-time applications to coexist with non-real-time applications in an open OS environment [11]. Another is standardized in the ARINC Avionics Application Software Standard Interface (APEX) for Integrated Modular Avionics (IMA) [12]. One level consists of *partitions*, which are executed cyclically and scheduled statically and off-line. Within each partition, application *processes* are scheduled using potentially more flexible approaches.

Each task in a partition is characterized statically by period (for periodic tasks), deadline within the period, and worst-case execution time. Aperiodic tasks are supported; Audsley and Wellings offer an analysis approach assuming minimum arrival time for aperiodic task periods [13]. TAO used this same approach initially to handle aperiodic tasks with rate monotonic scheduling and analysis.

APEX Partitions are scheduled cyclically. Each partition is characterized statically by parameters including criticality level, period, and duration. Therefore, a straightforward static scheduling approach can be used.

The APEX approach provides static schedulability analysis and fault tolerance across partitions. However, it suffers from the drawbacks of static scheduling described in Section 2.1. In particular, it is not clear how APEX can appreciably improve resource utilization when compared to conventional static scheduling approaches. For instance, jitter may be high when the the period of a task is not a multiple of its partition's period [13]. In that case, the task could become ready to run at a time when another partition was executing, and therefore would have to wait for its partition's activation.

3 Dynamic Scheduling Overhead

To assess the run-time cost of dynamic scheduling, we used an experimental setup based on TAO's Event Channel [14]. It consisted of a single high-priority supplier/consumer pair, and a varied number of low-priority event suppliers and consumers. We measured the latency in event delivery between the high-priority supplier and consumer. This latency included the time required for the TAO run-time scheduler to satisfy the Event Channel dispatch module scheduling request.

The test was run on a Sun Ultra 30 in the RealTime scheduling class, with a single 300 MHz UltraSPARC CPU, for two different scheduling strategies. The static scheduling strategy used off-line RMS and table lookup at runtime. The dynamic strategy used MUF, and therefore required an additional laxity calculation at runtime.

As shown in Figure 3, there appears to be a small (up to 10 percent) overhead for dynamic scheduling.

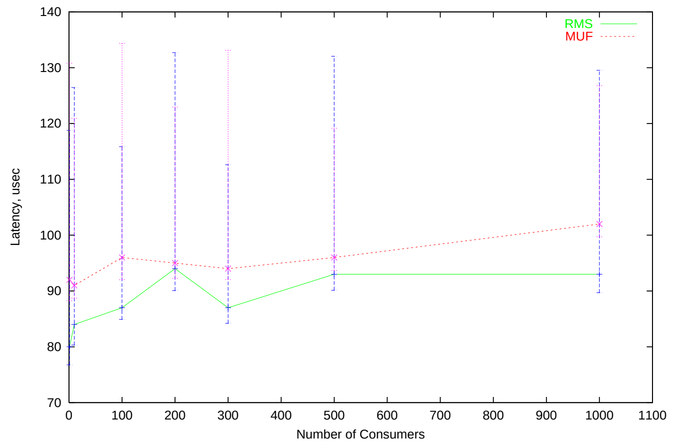

Figure 3: The Cost of Dynamic Scheduling

4 Concluding Remarks

As mission system computing evolves to address increasingly complex user requirements, run-time variation and non-periodic activities will become increasingly common. In addition, required system flexibility, in support of changing mission conditions, will no longer be achievable with static modal behavior. Conventional static approaches to scheduling mission computer activities will not support these increasingly complex behaviors.

Dynamic scheduling approaches offer potential solutions to these increasingly complex requirements, but at some run-time cost. The contribution of this work has been the implementation and analysis of selected dynamic scheduling algorithms within a strategized scheduling framework. From this work, we have concluded that the Maximum Urgency First scheduling algorithm has desirable properties for use within a mission computing application and acceptable run-time overhead.

In future work, we plan to measure the run-time cost of dynamic scheduling in TAO's scheduling service under a variety of of operational conditions. Further work will also explore scheduling in distributed systems. Dynamic scheduling appears to be a prerequisite for distributed system scheduling, due to the loose coupling between operations on separate processors.

5 Acknowledgments

This work was funded in part by Boeing. We gratefully acknowledge the support and direction of the Boeing Principal Investigator, Bryan Doerr.

References

[1] T. H. Harrison, D. L. Levine, and D. C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," in *Proceedings of OOPSLA '97*, (Atlanta, GA), ACM, October 1997.

[2] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.2 ed., Feb. 1998.

[3] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments," *IEEE Communications Magazine*, vol. 14, February 1997.

[4] J. R. Newport, *Avionics Systems Design*. Boca Raton, Florida: CRC Press, 1994.

[5] D. B. Stewart, D. E. Schmitz, and P. K. Khosla, "Implementing Real-Time Robotic Systems using CHIMERA II," in *Proceedings of 1990 IEEE International Conference on Robotics and Automation*, (Cincinnatti, OH), 1992.

[6] D. C. Schmidt, D. L. Levine, and S. Mungee, "The Design and Performance of Real-Time Object Request Brokers," *Computer Communications*, vol. 21, pp. 294–324, Apr. 1998.

[7] J. P. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced Aperiodic Scheduling in Hard Real-Time Environments," in *Proceedings of the IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, 1987.

[8] D. B. Stewart and P. K. Khosla, "Real-Time Scheduling of Sensor-Based Control Systems," in *Real-Time Programming* (W. Halang and K. Ramamritham, eds.), Tarrytown, NY: Pergamon Press, 1992.

[9] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *JACM*, vol. 20, pp. 46–61, January 1973.

[10] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour, *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Norwell, Massachusetts: Kluwer Academic Publishers, 1993.

[11] Z. Deng and J. W.-S. Liu, "Scheduling Real-Time Applications in an Open Environment," in *Proceedings of the 18th IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, Dec. 1997.

[12] ARINC Incorporated, Annapolis, Maryland, USA, *Document No. 653: Avionics Application Software Standard Inteface (Draft 15)*, Jan. 1997.

[13] N. Audsley and A. Wellings, "Analysing APEX Applications," in *Proceedings of the 16th Real-Time Systems Symposium*, pp. 39–44, Dec. 1996.

[14] T. H. Harrison, C. O'Ryan, D. Levine, and D. C. Schmidt, "The Design and Performance of a Real-time CORBA Event Service," *submitted to the Journal on Selected Areas in Communications special issue on Service Enabling Platforms for Networked Multimedia Systems*, 1998.