

AN EVOLUTION OF QOS CONTEXT PROPAGATION IN EVENT-MEDIATED AVIONICS SOFTWARE ARCHITECTURES

Christopher D. Gill and Joseph W. Hoffert,

Department of Computer Science, Washington University, St. Louis, MO

David C. Sharp and Patrick H. Goertzen, The Boeing Company, St. Louis, MO

1. Introduction

Situation awareness, an operational tempo that exceeds the enemy's ability to react, and quality execution of battle plans all contribute to achieving battlefield dominance. Greater coordination between command echelons, combat groups, and individual warfighters, increases shared awareness of the evolving battlefield, allowing elements at all levels to leverage information to identify enemy vulnerabilities and set in motion maneuvers to exploit that vulnerability.

Achieving this level of coordination requires (1) mission 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 adaptive software concepts; both in the application framework and in the specific application components used in avionics mission computers.

Earlier work on static [1] and dynamic and hybrid static/dynamic [2][3][4] resource

management techniques in event-mediated architectures relied primarily on locally available information to perform dynamic dispatching and adaptive scheduling on each node of a distributed system, and limited the amount of quality of service (QoS) information that was propagated *in-band* with the events, or *out-of-band* through other channels¹. However, further work on adaptive resource management [5] and integrated multi-layer resource management [6] required an increasingly distributed and interactive view of resource management and an associated increase in the amount of QoS information propagated.

Distributed threading models, such as that supplied by the emerging Dynamic Scheduling Real-Time CORBA 2.0 Joint Final Submission [7] offer a model in which QoS context is propagated in-band with the schedulable entity, *i.e.*, the thread. We discuss how we have evolved a related QoS context propagation model for several generations of event-mediated architectures, utilizing both in-band and out-of-band context propagation.

This paper makes contributions in two principal areas. First, it describes the evolution of a context propagation model for a broad class of event-mediated architectures, and defines semantics for adaptive scheduling and dynamic dispatching in that model. Second, it categorizes event-mediated applications that benefit from different forms of QoS context propagation and provides architectural guidelines for both configuring them and selecting among them, based on application requirements.

¹ In-band propagation occurs as the event itself is pushed through the event channel, in each thread of execution handling the event; out-of-band propagation occurs as a hand-off to a thread of execution not in the event path.

2. Event Mediated Architectures

We consider four generations of event-mediated avionics software architectures in this paper, each of which includes and extends the capabilities of the previous generations:

- Static scheduling using RMS
- Hybrid static/dynamic scheduling
- Adaptively reconfigured scheduling via higher-level resource managers
- Integrated scheduling of operations with hard or statistical execution time bounds

Each of these generations adds to the QoS context propagation requirements of the previous generations. We consider each generation and its extensions to the propagated QoS context, in turn.

2.1. Statically Scheduled Event-Mediated Architectures

The first-generation architecture applied to avionics mission computing software [1] addressed two principal areas of concern to real-time avionics mission computing applications: (1) efficient decoupling of concurrency flow from application functionality, and (2) support for deterministic scheduling of periodic operations. The first area of concern was addressed through event correlation and filtering techniques [1] that are described outside the scope of this discussion.

The first-generation QoS context propagation model illustrated in Figure 1 addressed the second area of concern. As shown in Figure 1, a real-time event channel event (`RtecEventComm::Event`) need only be decorated with a handle uniquely identifying the operation associated with that event, to be scheduled deterministically in a periodic system. Priorities in this architecture are assigned to events off-line [1], using an appropriate scheduling algorithm such as Rate Monotonic Scheduling [8] (RMS). A priority lookup² (by `handle`) is done for each arriving event, which is then enqueued according to the obtained priority for

² Separate priority assignments are calculated for each of several large-scale *modes* of the system, but these modes are transparent to the dispatching infrastructure and the events themselves. A mode switch changes an index within the scheduler, so a different priority table is consulted on lookup.

dispatching by the appropriate operating system thread in the dispatcher [1].

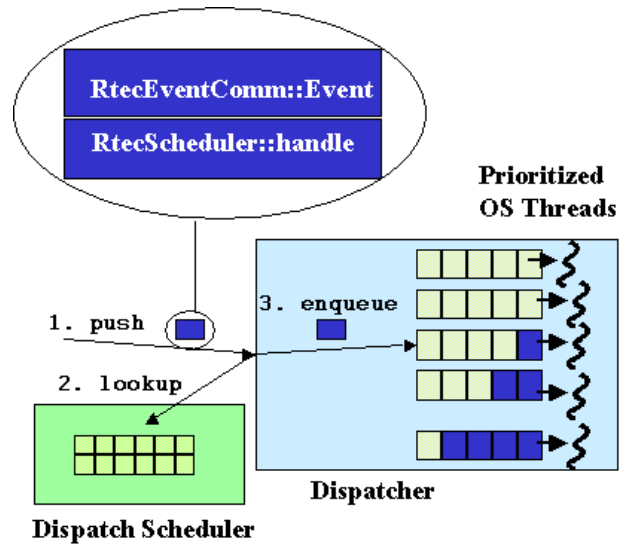


Figure 1. QoS Context Propagation in a Statically Scheduled Architecture

2.2. Hybrid Static/Dynamic Architectures

The second-generation architecture applied to avionics mission computing software [2][3][4] addressed challenges of (1) protecting determinism of critical operations from non-critical operation CPU requirements, while (2) increasing total utilization by completely scheduling or even slightly over-scheduling the CPU with non-critical operations. Developing infrastructure to support hybrid static/dynamic scheduling and dispatching techniques [2][3] addressed both areas of concern.

Figure 2 shows the QoS context propagation model resulting from this approach. Compared to the statically scheduled case, this approach introduces three kinds of overhead for event dispatching. First, an event must carry additional information used for dynamic scheduling, such as *deadline* under Earliest Deadline First [8] (EDF) scheduling, to which we must add *execution time* under Minimum Laxity First [9] (MLF) scheduling. These additional fields increase the memory footprint of the application and may also incur greater latency for transmission of events across a network.

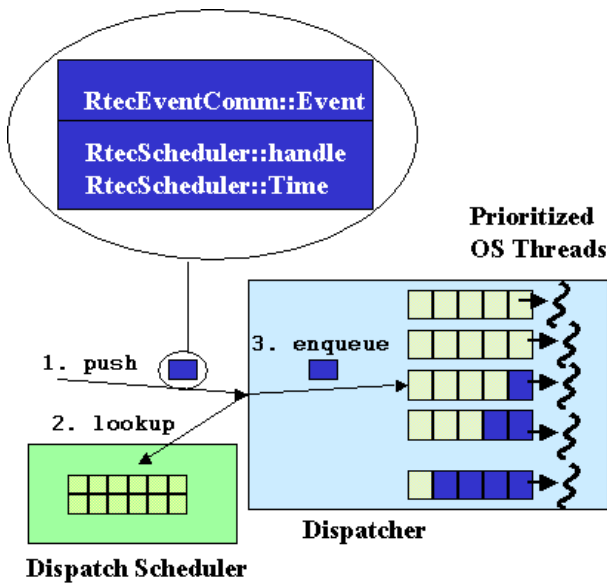


Figure 2. QoS Context Propagation in a Hybrid Static/Dynamic Architecture

Second, additional space may be needed for fields added to the scheduler’s lookup tables, such as criticality values under Maximum Urgency First [9] (MUF) scheduling. Third, the dispatching queues themselves must order operations according to run-time values [3], further increasing both the amount and performance cost of QoS context propagated and used in the dispatcher.

Despite these additional sources of overhead, our empirical studies [4] of the effectiveness of this second-generation architecture have demonstrated that these techniques can be used to achieve both (1) protection of critical operation deadlines through priority isolation and (2) increased utilization of the CPU by non-critical operations, in realistic avionics mission computing environments. Therefore, this second-generation architecture has been retained in the subsequent generations.

2.3. Adaptively Scheduled Architectures

The third-generation architecture applied to avionics mission computing software [5][6] has addressed the challenges of (1) providing closed-loop adaptive resource management in a highly variable execution environment, while (2) avoiding prohibitive levels of overhead in managing resources. Figure 3 illustrates closed-loop adaptive resource management with a single resource manager and the scheduling and dispatching

architecture described previously, as was described in [5]. For greater adaptive control, additional managers can be added as in [6], with a corresponding increase in the number and complexity of QoS context propagation loops in the system.

As shown in Figure 3, no additional QoS context propagates with the event, beyond that described in Section 2.2 for the previous generation architecture. However, closing the resource management control loop requires that each event dispatch be checked for deadline success or failure.

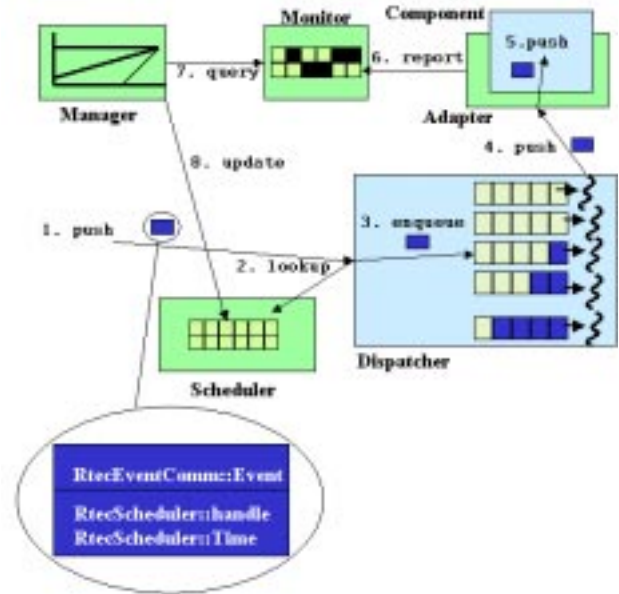


Figure 3. QoS Context Propagation in an Adaptively Scheduled Architecture

While component adapters and monitors were optionally added to the previous-generation architecture for system measurement purposes [4], these objects become part of the normal run-time infrastructure in an adaptive architecture. Their use in the adaptive architecture further serves to decouple the functionality of application components from QoS propagation activities. For example, an adapter may intercept the return from an event push to an application component, and assess deadline success or failure.

Deadline success and failure information propagates to the monitor in-band, in the dispatching thread. The resource manager queries the monitor out-of-band and then may adapt the behavior of the system, in conjunction with the

scheduler, by modifying the rates at which operations execute and the priorities at which they are dispatched [5].

To reduce the overhead of resource management along this closed loop, a number of optimizations are possible, as described in [10]. We will perform characteristic measurements of a realistic closed loop resource management configuration under the Weapon Systems Open Architecture [11] (WSOA) program’s scheduled ground and flight test demonstrations, to assess the applicability of these optimizations to adaptive resource management in an operating avionics mission computing system.

2.4. Hybrid Deterministic/Statistical Architectures

The current fourth-generation architecture we have applied to avionics mission computing software addresses the challenges of (1) unifying dispatching models for traditional avionics mission computing applications with those for advanced reasoning-based avionics applications [12] while (2) providing suitable schedulability assurances for the combined set of tasks, some of which have varying degrees of execution time determinism.

We address these challenges by extending our scheduling and dispatching infrastructure from previous generations, described in Sections 2.1-2.3, to (1) provide common dispatching abstractions across mission computing components and reasoning task network elements and (2) apply advanced scheduling techniques such as Statistical Rate Monotonic Scheduling [13] (SRMS) that can co-schedule operations with acceptable deterministic execution bounds and those with only statistically acceptable execution bounds.

A complete consideration of the latter subject is outside the scope of this paper. However, its implications influence the resulting QoS context propagation model for the common dispatching abstractions, which are the focus of this section. Figure 4 shows the integrated dispatching environment, in which both (1) a traditional avionics mission computing dispatch scheduler and (2) a task network architecture [12] (TNA) scheduler interact with the dispatcher.

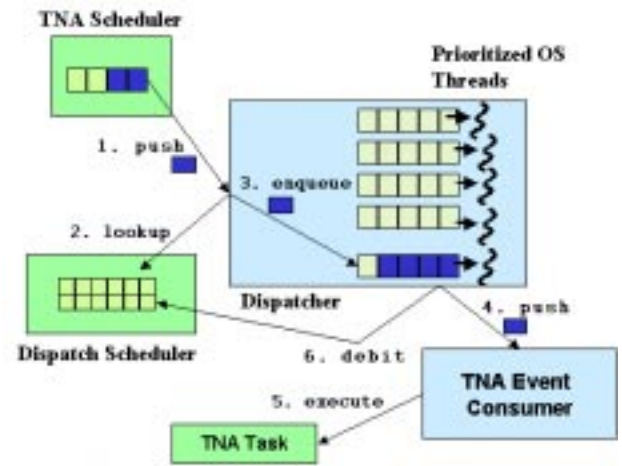


Figure 4. QoS Context Propagation in a Unified Deterministic/Statistical Architecture

Unlike the traditional scheduler, which is passively used for lookups, the TNA scheduler actively pushes events to the dispatcher. Thus, the TNA scheduler acts as a supplier of events emanating from the reasoning application, while the traditional scheduler operates in-band for events emanating from either (1) the reasoning application or (2) the mission computing application. Thus, the TNA must register its events with the traditional scheduler as well as with its own scheduler to ensure proper priority-based queue selection in the dispatcher.

A second area where QoS context propagation is expanded in the current fourth-generation architecture is the deadline accounting required to implement SRMS. SRMS maintains budgets for each operation, and debits those budgets to ensure protection of resource allocations in operations at lower rates [13]. We implement these budgets within the scheduler, and thus require that upon successful dispatch of an operation its budget be debited appropriately. We perform these debits in-band, in the dispatching thread. In addition, the decision to dispatch an operation must be weighed against its remaining budget, so that a lookup similar to the priority lookup must be performed upon event arrival.

Task network elements themselves represent a third area where QoS context is expanded in the current fourth-generation architecture. Though they have many properties in common with operations dispatched by pushing events to application

components [1], task network element executions are fundamentally different, as shown in Figure 5.

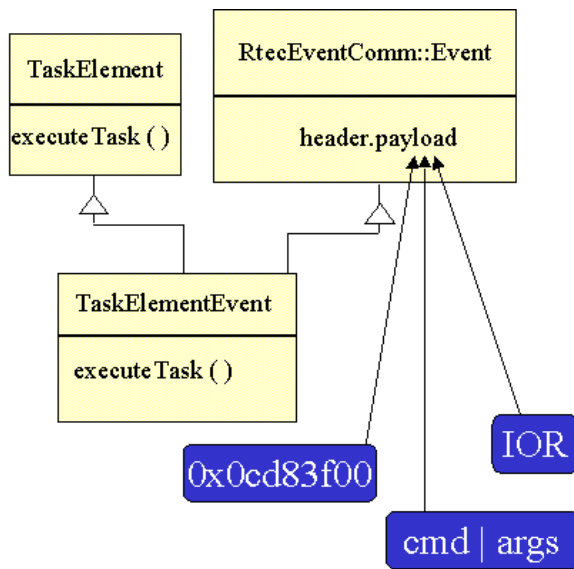


Figure 5. Callback Context Alternatives for Integrated Operation/TaskElement Dispatching

Task network element objects are invoked directly by calls to their `executeTask()` methods, so that integrating these objects with an event-mediated dispatching infrastructure requires use of the Adapter design pattern [14] to provide an appropriate event interface. We use the *class form* [14] of Adapter to *mix-in* task element and event interfaces. In addition, we provide a reference to the task element object in the combined *task element event*. Task element events are pushed to special TNA event consumer objects, which obtain and use these references to invoke each pushed task element’s `executeTask()` method.

We support polymorphic variations in this approach, so that the embedded reference need only be safe with respect to the context in which it is pushed to a TNA event consumer. For example, if the task element and the TNA event consumer were known to be in the same address space, then a native C++ pointer would suffice. However, if the task element and the TNA event consumer were in separate processes, or even on remote endsystems, then a CORBA [15] interoperable object reference (IOR) would be appropriate.

We implement polymorphic dispatch execution by using a CORBA octet sequence to hold the reference in the propagated event QoS

context. While we address the specific issues raised by the TNA execution model, Figure 5 illustrates that this approach opens other potentially useful alternatives, such as applying the Command [14] pattern to pass commands and arguments to operations with other styles of invocation.

2.5. Synopsis: Evolution of Architectures and Corresponding QoS Contexts

Each of the architectures described in Sections 2.1 through 2.4 extends the QoS context propagated by the previous architectures. Table 1 summarizes the QoS context information propagated in each of these architectures.

Table 1. Summary of QoS Context Extensions

Architecture	Additional Context
Statically Scheduled	Handle
Hybrid Static/Dynamic	Deadline, execution
Adaptive	Deadline statistics, operation updates
Deterministic/Statistical	Deadline accounting, callback propagation

With the QoS context extensions in each architecture comes a corresponding evolution of QoS capabilities. Table 2 summarizes the overall distinctions between these architectures.

Table 2. Summary of QoS Evolution

Architecture	QoS Evolution
Statically Scheduled	All scheduling determined prior to run-time
Hybrid Static/Dynamic	Non-critical functionality scheduled at run-time
Adaptive	Adaptive rescheduling of (potentially all) functionality at run-time
Deterministic/Statistical	Some operation characteristics not known prior to run-time execution

3. Application Requirements for QoS Context Propagation

Matching different applications to the architectures described in Section 2 requires close examination of the specific requirements of each application. As each of the QoS context extensions described in Section 2 adds overhead in program footprint, run-time performance, and overall system complexity, choosing the minimal architecture that meets an application's requirements is desirable. We therefore categorize applications according to a mapping of their requirements onto the architectural generations described in Section 2:

- Static applications
- Dynamic applications
- Adaptive applications
- Statistically bounded applications

In this section we identify the characteristics of applications in each of these categories. We describe an example application in each category.

3.1. Static Applications

Static applications are those whose static and dynamic resource management characteristics can be defined *a priori* either off-line at system configuration time, or on-line at system start time. In either case, the application's resource management characteristics are rigidly defined prior to system run time.

As noted in Section 2.1, an appropriate architecture for these applications need only support handle-based lookup for operation dispatching, possibly coupled with coarse-grained modes of system execution. Priority lookup based on handles (1) provides static resource management within a mode, and (2) allows efficient de-multiplexing of dispatch requests onto queues managed by prioritized operating system threads. Modes provide coarse-grained dynamic resource management, allowing different static lookup tables to be computed and stored in advance, then used in alternation at run-time to tailor static resource management to large-scale changes in the operating environment.

Given both their relative simplicity and resulting system determinism, the majority of traditional systems have been designed with static

schedules. In safety critical applications, like flight control systems, static scheduling facilitates *a priori* analysis to ensure that the system will always meet deadlines. Even in less critical environments, the simplicity, repeatability, and testability afforded by static scheduling approaches have made them a popular alternative.

More dynamic systems, like the cockpit pilot-vehicle interface software in mission computers, often have a small fraction of the complete system functionality executing simultaneously. In these cases coarse groupings of coupled functionality are analyzed and scheduled as different *modes*. For example, multi-role fighter aircraft might support navigation, air-to-air interdiction, and bombing. These different mission roles might be used as a means of coarsely partitioning scheduling of the system.

3.2. Dynamic Applications

Dynamic applications are those whose *dynamic* resource management characteristics cannot be adequately defined *a priori* before system run time. This is often a function of a more rapidly and widely changing operating environment, and at finer granularity, than the environments in which static resource management is appropriate. As a result, the resources needed to schedule the entire system may vary widely compared to the more static requirements of the core set of critical operations.

To balance the strict static requirements of the critical core with the desire to limit total resources provisioned, different combinations of static and dynamic resource management can be used. For some approaches, static priority assignment is combined with dynamic queue management for all operations, as in MUF [9] scheduling. For others, static priority assignment is again used, but dynamic resource management is only applied to the non-critical operations, as in RMS+LLF [16].

As described in Section 2.2, an appropriate architecture for these applications needs to support propagation of run-time parameters such as an operation's deadline and execution time. Handle-based lookup for operation dispatching may still be used, again possibly coupled with coarse-grained modes of system execution. Queues for dynamically managed operations order them

according to run-time properties such as time to deadline or laxity³.

The distinction between static and dynamic functionality is frequently coupled to their criticality. For example, in a flight control system, the low level control laws that produce actuator commands run at all times statically to maintain safe flight. Higher-level functionality, such as performing internal built-in-test to assess system health, can easily exceed processing resources and can be scheduled opportunistically. Similar situations arise within mission computing applications where core navigation displays must be provided at all times, but where pilot selections can result in widely varying display requirements for other data products.

3.3. Adaptive Applications

Adaptive applications are essentially Dynamic Applications whose *static* resource management characteristics such as execution rates and priorities must become *dynamic*, *i.e.*, they cannot be defined adequately *a priori* before system run time. This is often a function of an even more widely and unpredictably changing operating environment than those in which static or dynamic resource management alone is appropriate. As a result, the previously static specification of resources needed to schedule non-critical (or even in some cases, *critical*) operations may need to be updated adaptively at run-time.

Adaptation requires orders-of-magnitude longer time-scales than dynamic scheduling decisions, as (1) re-evaluation of operation parameters, *e.g.*, rate selection, (2) priority assignment, and (3) schedule feasibility assessment, must all be performed. These factors, in conjunction with application requirements for close interaction with the environment, may justify the additional overhead for adaptive resource management. Applications in this category are those with (1) the ability to tolerate large-scale shifts in run-time resource allocation, (2) important degrees of freedom in operation characteristics, and (3) operating environments where demands on the application can vary widely and unpredictably across a single execution of the entire system.

In an RMS [8] context, adaptive scheduling is beneficial in cases where adjusting execution rates in a fine-grained way improves overall system usefulness. In particular, when the number of possible fine-grained combinations of rates is larger than can be stored or managed reasonably using modes, adjusting them adaptively at run-time may provide an improved QoS management capability.

Adjusting execution rates adaptively can be helpful in many cases. For example, Synthetic Aperture Radar (SAR) mapping requires very precise position information to support subsequent image processing. Increasing the update rates of navigation information during SAR mapping, while compensating by reducing execution rates of fault detection logic, for example, could improve the overall effectiveness of the system.

3.4. Statistically Bounded Applications

Statistically bounded applications are those whose individual operations' resource requirement characteristics cannot be adequately bounded *a priori* before system run time. This is often a function of operations that perform open-ended activities such as reasoning as described in Section 2.4. As a result, the resource management needed to schedule the entire system must provide statistical resource allocations, often in conjunction with deterministic and/or dynamic resource allocations for operations whose characteristics can be bounded *a priori*.

Applications in the statistically bounded category are those with (1) execution or other requirements exceeding the allowed tolerances of determinism in the system, (2) operations that cannot be bounded by other techniques such as anytime execution [16], and (3) the ability to tolerate statistical assurances of resource allocation.

An example of this type of application is enroute mission replanning triggered by changes in the tactical situation. Statistical budgets can be established for the number of threats and path segments, and the target complexity to be included in the computation. However, pop-up threats and coupling between threat avoidance, best paths, and target goals encountered during the computation may cause wide variations in the execution time of any particular mission recomputation.

³ Laxity, also known as *slack*, is defined as the time to deadline minus the expected execution time of an operation.

3.5. Synopsis of Application Requirements

Table 3 summarizes the types of applications considered in this section, and their requirements leading to selection of one of the architectures and associated QoS context propagation models described in Section 2.

Table 3. Summary of Application Requirements

Application Type	Requirements
Static	Static modes
Dynamic	Dynamic total utilization
Adaptive	Adaptive mode definition
Statistically Bounded	Non-deterministic operation characteristics

4. Concluding Remarks

This paper presented four generations of resource management architectures we have applied to avionics mission computing systems. For each generation we identify and describe the additional propagation of QoS context needed beyond the previous generations, and how we have implemented each QoS context propagation model.

We describe the relationship between categories of applications and the above architectural generations, mapping application requirements to the resource management capabilities of each generation. An appropriate architecture, that meets all application requirements but does not incur unnecessary overhead, can then be chosen by comparison to the mapping between architectures and applications we describe. Thus, we offer guidance to developers and architects of applications requiring various levels of resource management, within and beyond the avionics mission computing domain.

5. Acknowledgements

The work described in this paper was funded in part by the Air Force Research Labs under the ASFD, ASTD (phases 1 and 2), and WSOA programs, and builds upon work funded by DARPA ITO under the Quorum program. We gratefully acknowledge the support and direction of the AFRL program manager, Kenneth Littlejohn, and the DARPA program manager, Dr. Gary Koob.

References

- [1] Harrison, Tim, David Levine, Douglas Schmidt, October 1997, *The Design and Performance of a Real-time CORBA Event Service*, Proceedings of OOPSLA '97, Atlanta, Georgia, ACM.
- [2] Levine, David, Christopher Gill, Douglas Schmidt, October 1998, *Dynamic Scheduling Strategies for Avionics Mission Computing*, 17th Digital Avionics System Conference (DASC), Seattle, Washington, IEEE/AIAA
- [3] Gill, Christopher, David Levine, Douglas Schmidt, March 2001, *The Design and Performance of a Real-Time CORBA Scheduling Service*, Vol. 20 No. 2, Real-Time Systems: the International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware
- [4] Gill, Christopher, David Levine, October 2000, *Quality of Service Management for Real-Time Embedded Information Systems*, 19th Digital Avionics System Conference (DASC), Philadelphia, Pennsylvania, IEEE/AIAA
- [5] Doerr, Bryan, Thomas Venturella, Rakesh Jha, Christopher Gill, and Douglas Schmidt, October 1999, *Adaptive Scheduling for Real-time Embedded Information Systems*, 18th Digital Avionics Systems Conference (DASC), St. Louis, Missouri, IEEE/AIAA.
- [6] Loyall, Joseph, Jeanna Gossett, Christopher Gill, Richard Schantz, John Zinky, Partha Pal, Richard Shapiro, Craig Rodrigues, Michael Atighetchi, David Karr, April 2001, *Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications*, 21st International Conference on Distributed Computing Systems (ICDCS-21), Phoenix, Arizona, IEEE Computer Society.

- [7] Object Management Group, April 2001, *Dynamic Scheduling Real-Time CORBA 2.0 Joint Final Submission*, Document orbos/2001-04-01, <http://www.omg.org>
- [8] Liu, C.L., J.W. Layland, January 1973, *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*, Vol. 20, No. 1, Journal of the ACM, ACM
- [9] Stewart, David, Pradeep Khosla, 1992, *Real-Time Scheduling of Sensor-Based Control Systems*, Real-Time Programming, Tarrytown, NY, Pergamon Press
- [10] Gill, Christopher, Douglas Schmidt, and Ron Cytron, submitted May 2001, *Middleware Scheduling Optimization Techniques for Distributed Real-Time and Embedded Systems*, International Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany, IFIP/ACM.
- [11] Corman, David, October 2001, *WSOA—Weapon Systems Open Architecture Demonstration—Using Emerging Open System Architecture Standards to Enable Innovative Techniques for Time Critical Target (TCT) Prosecution*, 20th Digital Avionics Systems Conference (DASC), Daytona Beach, Florida, IEEE/AIAA.
- [12] McBryan, B. and M. Joy, May 1999, *Rotorcraft Pilot's Associate Shared Memory Task Network Architecture*, AHS International Forum 55 Proceedings, AHS.
- [13] Atlas, Alia, Azer Bestavros, December 1998, *Statistical Rate Monotonic Scheduling*, 18th IEEE Real-Time Systems Symposium (RTSS '98), Madrid, Spain, IEEE
- [14] Gamma, Eric, Richard Helm, Ralph Johnson, John Vlissides, 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley
- [15] Object Management Group, October 2000, *The Common Object Request Broker: Architecture and Specification*, Ver. 2.4, OMG, <http://www.omg.org>
- [16] Chung, J.-Y., J. W.-S. Liu, K.-J. Lin, September 1990, *Scheduling Periodic Jobs that Allow Imprecise Results*, Vol. 39, IEEE Transactions on Computers, IEEE