Adaptive and Reflective Middleware for Distributed Real-time and Embedded Systems

Douglas C. Schmidt
Electrical & Computer Engineering Dept.
University of California, Irvine
Irvine, CA 92697-2625, USA
schmidt@uci.edu

Abstract

Software has become strategic to developing effective distributed real-time and embedded (DRE) systems. Next-generation DRE systems, such as total ship computing environments, coordinated unmanned air vehicle systems, and national missile defense, will use many geographically dispersed sensors, provide on-demand situational awareness and actuation capabilities for human operators, and respond flexibly to unanticipated run-time conditions. These DRE systems will also increasingly run unobtrusively and autonomously, shielding operators from unnecessary details, while communicating and responding to mission-critical information at an accelerated operational tempo. In such environments, it's hard to predict system configurations or workloads in advance. This paper describes the need for adaptive and reflective middleware systems (ARMS) to bridge the gap between application programs and the underlying operating systems and network protocol stacks in order to provide reusable services whose qualities are critical to DRE systems. ARMS middleware can adapt in response to dynamically changing conditions for the purpose of utilizing the available computer and network infrastructure to the highest degree possible in support of mission needs.

Motivation

New and planned distributed real-time and embedded (DRE) systems are inherently network-centric "systems of systems." DRE systems have historically been developed via *multiple technology bases*, where each system brings its own networks, computers, displays, software, and people to maintain and operate it. Unfortunately, not only are these "stove-pipe" architectures proprietary, but they tightly couple many functional and non-functional DRE system aspects, which impedes their

1. *Assurability*, which is needed to guarantee efficient, predictable, scalable, and dependable quality of service (QoS) from sensors to shooters
2. *Adaptability*, which is needed to (re)configure DRE systems dynamically to support varying workloads or missions over their lifecycles and
3. *Affordability*, which is needed to reduce initial non-recurring DRE system acquisition costs and recurring upgrade and evolution costs.

The affordability of certain types of systems, such as logistics and mission planning, can often be enhanced by using commercial-off-the-shelf (COTS) technologies. However, today's efforts aimed at integrating COTS into mission-critical DRE systems have largely failed to support affordability *and* assurability and adaptability effectively since they focus mainly on initial non-recurring acquisition costs and do not reduce recurring software lifecycle costs, such as "COTS refresh" and subsetting military systems for foreign military sales. Likewise, many COTS products lack support for controlling key QoS properties, such as predictable latency, jitter, and throughput; scalability; dependability; and security. The inability to control these QoS properties with sufficient confidence compromises DRE system adaptability and assurability, *e.g.*, minor perturbations in conventional COTS products can cause failures that lead to loss of life and property.

Historically, conventional COTS software has been particularly unsuitable for use in mission-critical DRE systems due to its either being:

1. Flexible and standard, but incapable of guaranteeing stringent QoS demands, which restricts assurability or
2. Partially QoS-enabled, but inflexible and non-standard, which restricts adaptability and affordability.

As a result, the rapid progress in COTS software for mainstream business information technology (IT) has not yet become as broadly applicable for mission-critical DRE systems. Until this problem is resolved effectively, DRE system integrators and warfighters will be unable to take advantage of future advances in COTS software in a dependable, timely, and cost effective manner. Thus, developing the new generation of assurable, adaptable, and affordable COTS software technologies is an important R&D goal.

Key Technical Challenges and Solutions

Some of the most challenging IT requirements for new and planned DRE systems can be characterized as follows:

- Multiple QoS properties must be satisfied in real-time
- Different levels of service are appropriate under different configurations, environmental conditions, and costs
- The levels of service in one dimension must be coordinated with and/or traded off against the levels of service in other dimensions to meet mission needs and
- The need for autonomous and time-critical application behavior necessitates a flexible distributed system substrate that can adapt robustly to dynamic changes in mission requirements and environmental conditions.

Standards-based COTS software available today cannot meet all of these requirements simultaneously for the reasons outlined in Section *Motivation*. However, contemporary economic and organizational constraints—along with increasingly complex requirements and competitive pressures—are also making it infeasible to built complex DRE system software entirely from scratch. Thus, there is a pressing need to develop, validate, and ultimately standardize a new generation of *adaptive and reflective middleware systems* (ARMS) technologies that can support stringent DRE system functionality and QoS requirements.

Middleware [Sch01a] is reusable service/protocol component and framework software that functionally bridges the gap between

1. the end-to-end functional requirements and mission doctrine of applications and
2. the lower-level underlying operating systems and network protocol stacks.

Middleware therefore provides capabilities whose quality and QoS are critical to DRE systems.

Adaptive middleware [Loy01] is software whose functional and QoS-related properties can be modified either

- *Statically*, *e.g.*, to reduce footprint, leverage capabilities that exist in specific platforms, enable functional subsetting, and minimize hardware and software infrastructure dependencies or
- *Dynamically*, *e.g.*, to optimize system responses to changing environments or requirements, such as changing component interconnections, power-levels, CPU/network bandwidth, latency/jitter, and dependability needs.

In DRE systems, adaptive middleware must make these modifications dependably, *i.e.*, while meeting stringent end-to-end QoS requirements.

Reflective middleware [Bla99] goes a step further to permit automated examination of the capabilities it offers, and to permit automated adjustment to optimize those capabilities. Thus, reflective middleware supports more advanced adaptive behavior, *i.e.*, the necessary adaptations can be performed autonomously based on conditions within the system, in the system's environment, or in DRE system doctrine defined by operators and administrators.

The Structure and Functionality of Middleware

Networking protocol stacks can be decomposed into multiple layers, such as the physical, data-link, network, transport, session, presentation, and application layers. Similarly, middleware can be decomposed into multiple layers, such as those shown in Figure 1.

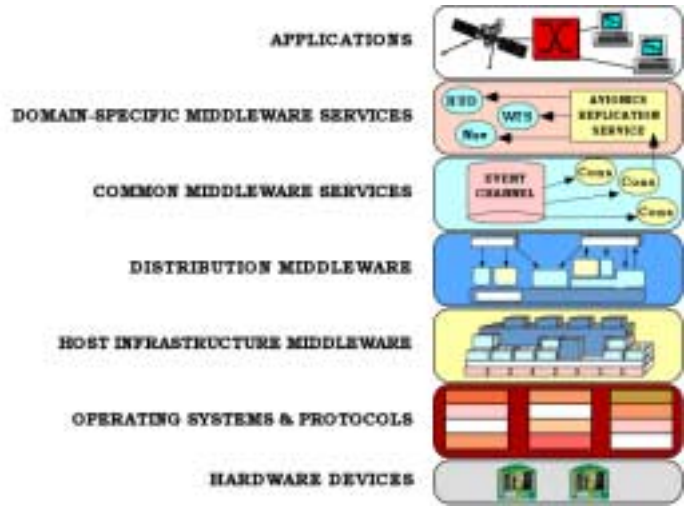

Figure 1. Layers of Middleware and Their Surrounding Context

Below, we describe each of these middleware layers and outline some of the COTS technologies in each layer that are suitable (or are becoming suitable) to meet the stringent QoS demands of DRE systems.

Host infrastructure middleware encapsulates and enhances native OS communication and concurrency mechanisms to create portable and reusable network programming components, such as reactors, acceptor-connectors, monitor objects, active objects, and component configurators [Sch00b]. These components abstract away the accidental incompatibilities of individual operating systems, and help eliminate many tedious, error-prone, and non-portable aspects of developing and maintaining networked applications via low-level OS programming API, such as Sockets or POSIX Pthreads. Examples of COTS host infrastructure middleware that are relevant for DRE systems include:

- *The ADAPTIVE Communication Environment* (ACE) [Sch01], which is a highly portable and efficient toolkit written in C++ that encapsulates native operating system (OS) network programming capabilities, such as connection establishment, event demultiplexing, interprocess communication,

(de)marshaling, static and dynamic configuration of application components, concurrency, and synchronization. ACE has been used in a wide range of commercial and military DRE systems, including hot rolling mill control software, surface mount technology for "pick and place" systems, missile control, avionics mission computing, software defined radios, and radar systems.

- *Real-time Java Virtual Machines* (RT-JVMs), which implement the Real-time Specification for Java (RTSJ) [Bol00]. The RTSJ is a set of extensions to Java that provide a largely platform-independent way of executing code by encapsulating the differences between real-time operating systems and CPU architectures. The key features of RTSJ include scoped and immortal memory, real-time threads with enhanced scheduling support, asynchronous event handlers, and asynchronous transfer of control between threads. Although RT-JVMs based on the RTSJ are in their infancy, they have generated tremendous interest in the R&D and integrator communities due to their potential for reducing software development and evolution costs.

Distribution middleware defines higher-level distributed programming models whose reusable APIs and mechanisms automate and extend the native OS network programming capabilities encapsulated by host infrastructure middleware. Distribution middleware enables developers to program distributed applications much like stand-alone applications, *i.e.*, by invoking operations on target objects without hard-coding dependencies on their location, programming language, OS platform, communication protocols and interconnects, and hardware characteristics.

At the heart of distribution middleware are QoS-enabled object request brokers, such as the Object Management Group's (OMG) *Common Object Request Broker Architecture* (CORBA) [Omg00]. CORBA is distribution middleware that allows objects to interoperate across networks regardless of the language in which they were written or the OS platform on which they are deployed. In 1998 the OMG adopted the Real-time CORBA (RT-CORBA) specification [Sch00a], which extends CORBA with features that allow DRE applications to reserve and manage CPU, memory, and networking resources. RT-CORBA implementations have been used in dozens of DRE systems, including telecom network management and call processing, online trading services, avionics mission computing, submarine DRE systems, signal intelligence and C4ISR systems, software defined radios, and radar systems.

Common middleware services augment distribution middleware by defining higher-level domain-independent components that allow application developers to concentrate on programming application logic, without the need to write the "plumbing" code needed to develop distributed applications by using lower level middleware features directly. Whereas distribution middleware focuses largely on managing end-system resources in support of an object-oriented distributed programming model, common middleware services focus on allocating, scheduling, and coordinating various end-to-end resources throughout a distributed system using a component programming and scripting model. Developers can reuse these services to manage global resources and perform recurring distribution tasks that would otherwise be implemented in an *ad hoc* manner by each application or integrator.

Examples of common middleware services include the OMG's CORBAServices [Omg98b] and the CORBA Component Model (CCM) [Omg99], which provide domain-independent interfaces and distribution capabilities that can be used by many distributed applications. The OMG CORBAServices and CCM specifications define a wide variety of these services, including event notification, logging, multimedia streaming, persistence, security, global time, real-time scheduling, fault tolerance, concurrency control, and transactions. Not all of these services are sufficiently refined today to be usable off-the-shelf for DRE systems. The form and content of these common middleware services will continue to mature and evolve, however, to meet the expanding requirements of DRE.

Domain-specific middleware services are tailored to the requirements of particular DRE system domains, such as avionics mission computing, radar processing, weapons targeting, or command and decision systems. Unlike the previous three middleware layers—which provide broadly reusable "horizontal" mechanisms and services—domain-specific middleware services are targeted at vertical markets. From a COTS perspective, domain-specific services are the least mature of the middleware layers today. This immaturity is due in part to the historical lack of distribution middleware and common middleware service *standards*, which are needed to provide a stable base upon which to create domain-specific middleware services. Since they embody knowledge of a domain, however, domain-specific middleware services have the most potential to increase the quality and decrease the cycle-time and effort that integrators require to develop particular classes of DRE systems.

A mature example of domain-specific middleware services is the Boeing Bold Stroke architecture [Sha98]. Bold Stroke uses COTS hardware, operating systems, and middleware to produce an open architecture for mission computing avionics capabilities, such as navigation, heads-up display management, weapons targeting and release, and airframe sensor processing. The domain-specific middleware services in Bold Stroke are layered upon COTS processors (PowerPC), network interconnects (VME), operating systems (VxWorks), infrastructure middleware (ACE), distribution middleware (Real-time CORBA), and common middleware services (the CORBA Event Service).

Recent Progress

Significant progress has occurred during the last five years in DRE middleware research, development, and deployment, stemming in large part from the following trends:

- ***Years of research, iteration, refinement, and successful use*** – The use of middleware and DOC middleware is not new [Sch86]. Middleware concepts emerged alongside experimentation with the early Internet (and even its predecessor ARPAnet), and DOC middleware systems have been continuously operational since the mid 1980's. Over that period of time, the ideas, designs, and most importantly, the software that incarnates those ideas have had a chance to be tried and refined (for those that worked), and discarded or redirected (for those that didn't). This iterative technology development process takes a good deal of time to get right and be accepted by user communities, and a good deal of patience to stay the course. When this process is successful, it often results in *standards* that codify the boundaries, and *patterns and frameworks* that reify the knowledge of how to apply these technologies, as described in the following bullets.

- ***The maturation of standards*** – Over the past decade, middleware standards have been established and have matured considerably with respect to DRE requirements. For instance, the OMG has adopted the following specifications in the past three years:
  - *Minimum CORBA*, which removes non-essential features from the full OMG CORBA specification to reduce footprint so that CORBA can be used in memory-constrained embedded systems.
  - *Real-time CORBA*, which includes features that allow applications to reserve and manage network, CPU, and memory resources predictably end-to-end.
  - *CORBA Messaging*, which exports additional QoS policies, such as timeouts, request priorities, and queueing disciplines, to applications.
  - *Fault-tolerant CORBA*, which uses entity redundancy of objects to support replication, fault detection, and failure recovery.

  Robust implementations of these CORBA capabilities and services are now available from multiple vendors. Moreover, emerging standards such as Dynamic Scheduling Real-Time CORBA, the Real-Time Specification for Java, and the Distributed Real-Time Specification for Java are extending the scope of open standards for a wider range of DRE applications.

- ***The dissemination of patterns and frameworks*** – A substantial amount of R&D effort during the past decade has also focused on the following means of promoting the development and reuse of high quality middleware technology:
  - *Patterns* codify design expertise that provides time-proven solutions to commonly occurring software problems that arise in particular contexts [Gam95]. Patterns can simplify the design, construction, and performance tuning of DRE applications by codifying the accumulated expertise of developers who have successfully confronted similar problems before. Patterns also elevate the level of discourse in describing software development activities to focus on strategic architecture and design issues, rather than just the tactical programming and representation details.
  - *Frameworks* are concrete realizations of groups of related patterns [John97]. Well-designed frameworks reify patterns in terms of functionality provided by the middleware itself, as well as functionality provided by an application. Frameworks also integrate various approaches to problems where there are no *a priori*, context-independent, optimal solutions. Middleware frameworks can include strategized selection and optimization patterns so that multiple independently-developed capabilities can be integrated and configured automatically to meet the functional and QoS requirements of particular DRE applications.

Historically, the knowledge required to develop predictable, scalable, efficient, and dependable mission-critical DRE systems has existed largely in programming folklore, the heads of experienced researchers and developers, or buried deep within millions of lines of complex source code. Moreover, documenting complex systems with today's popular software modeling methods and tools, such as the Unified Modeling Language (UML), only capture *how* a system is designed, but do not necessarily articulate *why* a system is designed in a particular way. This situation has several drawbacks:

- Re-discovering the rationale for complex DRE system design decisions from source code is expensive, time-consuming, and error-prone since it's hard to separate essential QoS-related knowledge from implementation details.
- If the insights and design rationale of expert system architects are not documented they will be lost over time, and thus cannot help guide future DRE system evolution.
- Without proper guidance, developers of mission-critical DRE software face the Herculean task of engineering and assuring the QoS of complex DRE systems from the ground up, rather than by leveraging proven solutions.

Middleware patterns and frameworks are therefore essential to help capture DRE system design expertise in a more readily accessible and reusable format.

Much of the pioneering R&D on middleware patterns and frameworks was conducted in the DARPA ITO Quorum program [DARPA99]. This program focused heavily on CORBA open systems middleware and yielded many

results that transitioned into standardized service definitions and implementations for the Real-time [Sch98] and Fault-tolerant [Omg98a] CORBA specification and productization efforts. Quorum is an example of how a focused government R&D effort can leverage its results by exporting them into, and combining them with, other on-going public and private activities that also used a common open middleware substrate. Prior to the viability of standards-based COTS middleware platforms, these same R&D results would have been buried within custom or proprietary systems, serving only as an existence proof, rather than as the basis for realigning the R&D and integrator communities.

Looking Ahead

Due to advances in COTS technologies outlined earlier, host infrastructure middleware and distribution middleware have now been successfully demonstrated and deployed in a number of mission-critical DRE systems, such as avionics mission computing, software defined radios, and submarine information systems. Since COTS middleware technology has not yet matured to cover the realm of large-scale, dynamically changing systems, however, DRE middleware has been applied to relatively small-scale and statically configured embedded systems. To satisfy the highly application- and mission-specific QoS requirements in network-centric "system of system" environments, DRE middleware—particularly common middleware services and domain-specific services—must be enhanced to support the management of individual and aggregate resources used by multiple system components at multiple system levels in order to:

- *Manage communication bandwidth*, *e.g.,* network level resource capability and status information services, scalability to 10^2 subnets and 10^3 nodes, dynamic connections with reserved bandwidth, aggregate policy-controlled bandwidth reservation and sharing, incorporation of non-network resource status information, aggregate dynamic network resource management strategies, and managed bandwidth to enhance real-time predictability.
- *Manage distributed real-time scheduling and allocation of DRE system artifacts (such as CPUs, networks, UAVs, missiles, radar, illuminators, etc)*, *e.g.,* fast and predictable queueing time properties, timeliness assurances for end-to-end activities based on priority/deadlines, admission controlled request insertion based on QoS parameters and global resource usage metrics, and predictable behavior over WANs using bandwidth reservations.
- *Manage distributed system dependability*, *e.g.,* group communication-based replica management, dependability manager maintaining aggregate levels of object replication, run-time switching among dependability strategies, policy-based selection of replication options, and understanding and tolerating timing faults in conjunction with real-time behavior.
- *Manage distributed security*, *e.g.,* object-level access control, layered access control for adaptive middleware, dynamically variable access control policies, and effective real-time, dependability, and security interactions.

Ironically, there is currently little or no scientific underpinning for QoS-enabled resource management, despite the demand for it in most distributed systems. Today's system designers and mission planners develop concrete plans for creating global, end-to-end functionality. These plans contain high-level abstractions and doctrine associated with resource management algorithms, relationships between these, and operations upon these. There are few techniques and tools, however that enable *users*, *i.e.,* commanders, administrators, and operators, *developers*, *i.e.,* systems engineers and application designers and/or *applications* to express such plans systematically, reason about and refine them, and have these plans enforced automatically to manage resources at multiple levels in network-centric DRE systems.

Systems today are built in a highly static manner, with allocation of processing tasks to resources assigned at design time. For systems that never change, this is an adequate approach. Large and complex military DRE combat systems change and evolve over their lifetime, however, in response to changing missions and operational environments. Allocation decisions made during initial design often become obsolete over time, necessitating expensive and time-consuming redesign. If the system's requisite end-to-end functionality becomes unavailable due to mission and environment changes, there are no standard tools or techniques to diagnose configuration or run-time errors automatically. Instead, designers and operators write down their plans on paper and perform such reasoning, refinement, configuration generation, and diagnosis manually. This *ad hoc* process is clearly inadequate to manage the accelerated operational tempo characteristic of network-centric DRE combat systems.

To address these challenges, the R&D community needs to discover and set the technical approach that can significantly improve the effective utilization of networks and endsystems that DRE systems depend upon by creating middleware technologies and tools that can automatically allocate, schedule, control, and optimize customizable—yet standards-compliant and verifiably correct—software-intensive systems. To promote a *common technology base*, the interfaces and (where appropriate) the protocols used by the middleware should be based on established or emerging industry or military standards that are relevant for DRE systems. However, the protocol and service *implementations* should be

customizable—statically and dynamically—for specific DRE system requirements.

To achieve these goals, middleware technologies and tools need to be based upon some type of layered architecture, such as the one shown in Figure 2 [Loy01]. This architecture decouples DRE middleware and applications along the following two dimensions:

- *Functional paths*, which are flows of information between client and remote server applications. In distributed systems, middleware ensures that this information is exchanged efficiently, predictably, scalably, dependably, and securely between remote peers. The information itself is largely application-specific and determined by the functionality being provided (hence the term "functional path").
- *QoS paths*, which are responsible for determining how well the functional interactions behave end-to-end with respect to key DRE system QoS properties, such as
  1. How and when resources are committed to client/server interactions at multiple levels of distributed systems
  2. The proper application and system behavior if available resources do not satisfy the expected resources and
  3. The failure detection and recovery strategies necessary to meet end-to-end dependability requirements.

In next-generation DRE systems, the middleware—rather than operating systems or networks in isolation—will be responsible for separating non-functional DRE system QoS properties from the functional application properties. Middleware will also coordinate the QoS of various DRE system and application resources end-to-end. The architecture in Figure 2 enables these properties and resources to change independently, *e.g.*, over different distributed system configurations for the same applications.

The architecture in Figure 2 is based on the expectation that non-functional QoS paths will be developed, configured, monitored, managed, and controlled by a different set of specialists (such as systems engineers, administrators, operators, and perhaps someday automated agents) and tools than those customarily responsible for programming functional paths in DRE systems. The middleware is therefore responsible for collecting, organizing, and disseminating QoS-related meta-information needed to

1. Monitor and manage how well the functional interactions occur at multiple levels of DRE systems and
2. Enable the adaptive and reflective decision-making needed to support non-functional QoS properties robustly in the face of rapidly changing mission requirements and environmental conditions.

These middleware capabilities are crucial to ensure that the aggregate behavior of complex network-centric DRE systems is dependable, despite local failures, transient overloads, and dynamic functional or QoS reconfigurations.

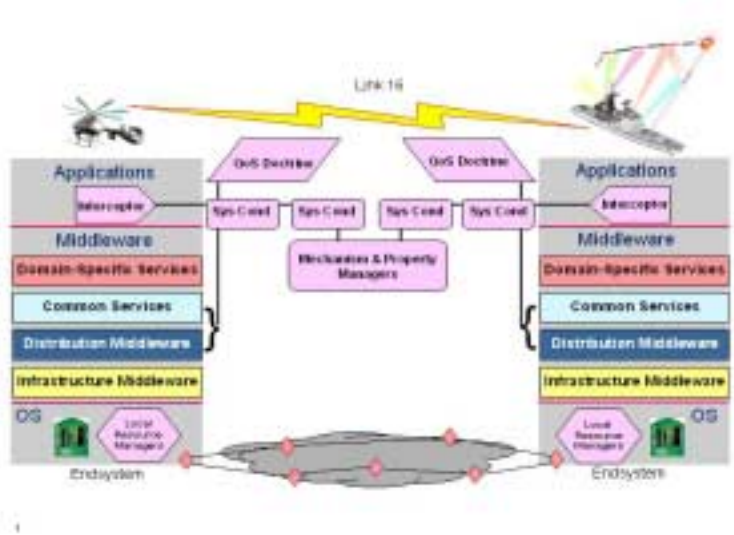

Figure 3. Decoupling Functional and QoS Paths

To simultaneously enhance assurability, adaptability, *and* affordability, the middleware techniques and tools developed in future R&D programs increasingly need to be application-independent, yet customizable within the interfaces specified by a range of open standards, such as

- The OMG Real-time CORBA specifications and The Open Group's QoS Forum
- The Java Expert Group Real-time Specification for Java (RTSJ) and the Distributed RTSJ
- The DMSO/IEEE High-level Architecture Run-time Infrastructure (HLA/RTI) and
- The IEEE Real-time Portable Operating System (POSIX) specification.

Concluding Remarks

Advances in wireless networks and COTS hardware technologies are enabling the lower level aspects of network-centric DRE systems. The emerging middleware software technologies and tools are likewise enabling the higher level distributed real-time and embedded (DRE) aspects of network-centric DRE systems, making them tangible and affordable by controlling the hardware, network, and endsystem mechanisms that affect mission, system, and application QoS tradeoffs.

The economic benefits of middleware stem from moving standardization up several levels of abstraction by maturing DRE software technology artifacts, such as middleware frameworks, protocol/service components, and patterns, so that they are readily available for COTS acquisition and customization. This middleware focus is helping to lower the total ownership costs of DRE systems by leveraging common technology bases so that complex and DRE functionality need not be re-invented

repeatedly or reworked from proprietary "stove-pipe" architectures that are inflexible and expensive to evolve and optimize.

Adaptive and reflective middleware systems (ARMS) are a key emerging theme that will help to simplify the development, optimization, validation, and integration of middleware in DRE systems. In particular, ARMS will allow researchers and system integrators to develop and evolve complex DRE systems assurably, adaptively, *and* affordably by:

- Standardizing COTS at the middleware level, rather than just at lower hardware/networks/OS levels and
- Devising optimizers, meta-programming techniques, and multi-level distributed dynamic resource management protocols and services for ARMS that will enable DRE systems to customize standard COTS interfaces, without the penalties incurred by today's conventional COTS software product implementations.

Many DRE systems require these middleware capabilities. Additional information on DRE middleware is available at `www.ece.uci.edu/~schmidt`.

References

[Bla99] Blair, G.S., F. Costa, G. Coulson, H. Duran, et al, "The Design of a Resource-Aware Reflective Middleware Architecture", *Proceedings of the 2nd International Conference on Meta-Level Architectures and Reflection*, St.-Malo, France, Springer-Verlag, LNCS, Vol. 1616, 1999.

[Bol00] Bollella, G., Gosling, J. "The Real-Time Specification for Java," *Computer*, June 2000.

[DARPA99] DARPA, *The Quorum Program*, http://www.darpa.mil/ito/research/quorum/index.html, 1999.

[Gam95] Gamma E., Helm R., Johnson R., Vlissides J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[John97] Johnson R., "Frameworks = Patterns + Components", *Communications of the ACM*, Volume 40, Number 10, October, 1997.

[Loy01] Loyall JL, Gossett JM, Gill CD, Schantz RE, Zinky JA, Pal P, Shapiro R, Rodrigues C, Atighetchi M, Karr D. "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications". *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS-21)*, April 16-19, 2001, Phoenix, Arizona.

[Omg98a] Object Management Group, "Fault Tolerance CORBA Using Entity Redundancy RFP", OMG Document orbos/98-04-01 edition, 1998.

[Omg98b] Object Management Group, "CORBA-Servcies: Common Object Service Specification," OMG Technical Document formal/98-12-31.

[Omg99] Object Management Group, "CORBA Component Model Joint Revised Submission," OMG Document orbos/99-07-01.

[Omg00] Object Management Group, "The Common Object Request Broker: Architecture and Specification Revision 2.4, OMG Technical Document formal/00-11-07", October 2000.

[Sch86] Schantz, R., Thomas R., Bono G., "The Architecture of the Cronus Distributed Operating System", *Proceedings of the 6th IEEE International Conference on Distributed Computing Systems (ICDCS-6)*, Cambridge, Massachusetts, May 1986.

[Sch98] Schmidt D., Levine D., Mungee S. "The Design and Performance of the TAO Real-Time Object Request Broker", *Computer Communications Special Issue on Building Quality of Service into Distributed Systems*, 21(4), 1998.

[Sch00a] Schmidt D., Kuhns F., "An Overview of the Real-time CORBA Specification," *IEEE Computer Magazine*, June, 2000.

[Sch00b] Schmidt D., Stal M., Rohnert H., Buschmann F., *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, Wiley and Sons, 2000.

[Sch01] Schmidt D., Huston S., *C++ Network Programming: Resolving Complexity with ACE and Patterns*, Addison-Wesley, Reading, MA, 2001.

[Sch01a] Schantz R., Schmidt D., "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," Encyclopedia of Software Engineering, Wiley & Sons, 2001.

[Sha98] Sharp, David C., "Reducing Avionics Software Cost Through Component Based Product Line Development", *Software Technology Conference*, April 1998.