R&D Advances in 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

Introduction

Distributed real-time and embedded (DRE) systems are playing an increasingly important role in modern application domains. There are many types of DRE systems, but they have one thing in common: *the right answer delivered too late becomes the wrong answer*. Providing the right answer at the right time is clearly crucial for life-critical military DRE systems, such as those that defend ships against missile attacks or that control unmanned combat air vehicles over wireless links. It is also crucial for safety-critical civilian DRE systems, such as control systems that regulate the temperature of coolant in a nuclear reactor or maintain safe operation of steel manufacturing machinery.

The affordability of certain types of distributed systems, such as two and three-tier business systems, can often be enhanced by using commercial-off-the-shelf (COTS) technologies. Today's efforts aimed at integrating COTS into mission-critical DRE systems, however, focus mainly on initial non-recurring acquisition costs and do not reduce recurring software lifecycle costs. Likewise, many COTS products lack support for controlling key quality of service (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.

Conventional COTS software has historically been 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 limits system assurability or
2. Partially QoS-enabled, but inflexible and non-standard, which limits system adaptability and affordability.

As a result, the rapid progress in COTS software for mainstream business systems has not yet become as broadly applicable for mission-critical DRE systems. Until this problem is resolved effectively, DRE system integrators and end-users will not be able to take advantage of future advances in COTS software in a dependable, timely, and cost effective manner.

This article describes key R&D efforts that are creating the new generation of assurable, adaptable, and affordable COTS software technologies to meet the stringent demands of mission-critical DRE systems. Although the use of COTS software in DRE systems has been limited in scope and domain, future prospects will be much brighter as a result of the work described in this article.

Technical Challenges & Solution Approaches

Some of the most challenging 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.

Although conventional COTS software cannot meet all of these requirements, today's economic and organizational constraints—along with increasingly complex requirements and competitive pressures—are 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* [Bla99] technologies that can support stringent DRE system functionality and QoS requirements.

Middleware [Sch01] is reusable systems 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

- *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 mission-critical 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. Reflective middleware therefore supports more advanced adaptations that can be performed autonomously based on conditions within the system, in the system's environment, or in DRE system policies defined by operators and administrators.

Middleware Layers and R&D Efforts

Just as networking protocol stacks can be decomposed into multiple layers, middleware can also be decomposed into multiple layers, such as those shown in Figure 1.

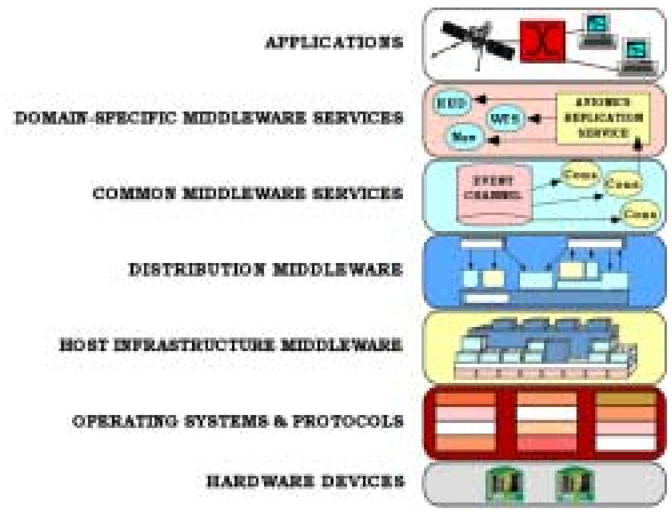
APPLICATIONS
DOMAIN-SPECIFIC MIDDLEWARE SERVICES
HUD
WTS
Nav
AVIONICS REPLICATION SERVICE
COMMON MIDDLEWARE SERVICES
EVENT CHANNEL
Cons
Cons
Cons
DISTRIBUTION MIDDLEWARE
HOST INFRASTRUCTURE MIDDLEWARE
OPERATING SYSTEMS & PROTOCOLS
HARDWARE DEVICES

Figure 1. Layers of Middleware & Their Context

Each of these middleware layers is described below, along with a summary of key R&D efforts at each layer that are helping to evolve the ability of middleware 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 [Sch00]. 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.

An example of host infrastructure middleware R&D that is relevant for DRE systems is the Open Virtual Machine (OVM) project <http://www.ovmj.org> conducted by researchers at Purdue, University of Maryland, and SUNY Oswego as part of the DARPA ITO PCES program. OVM is an open-source *Real-time Java Virtual Machine* that implements 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 within a thread.

The OVM virtual machine is written entirely in Java and its architecture emphasizes customizability and pluggable components. Its implementation strives to maintain a balance between performance and flexibility, allowing users to customize the implementation of operations such as message dispatch, synchronization, field access, and speed. OVM allows dynamic updates of the implementation of instructions of a running VM. Although RTSJ VMs like OVM or TimeSys Jtime are relatively new, they have generated tremendous interest in the R&D and DRE systems 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 (ORBs), such as CORBA, COM+, and Java RMI. These ORBs allow objects to interoperate across networks regardless of the language in which they were written or the OS platform on which they are deployed.

An example of distribution middleware R&D that is relevant for DRE systems is the TAO project <http://www.cs.wustl.edu/~schmidt/TAO.html> [Sch98] conducted by researchers Washington University, St. Louis and the University of California, Irvine as part of the DARPA ITO Quorum program. TAO is an open-source Real-time CORBA ORB [Omg01] that allows DRE applications to reserve and manage

- *Processor resources* via thread pools, priority mechanisms, intra-process mutexes, and a global scheduling service for real-time systems with fixed priorities
- *Communication resources* via protocol properties and explicit bindings to server objects using priority bands and private connections and
- *Memory resources* via buffering requests in queues and bounding the size of thread pools.

TAO is implemented with reusable frameworks from the ACE [Sch02] host infrastructure middleware toolkit <http://www.cs.wustl.edu/~schmidt/ACE.html>. ACE and TAO are mature examples of middleware R&D transition, having been used in hundreds of DRE systems, including telecom network management and call processing, online trading services, avionics mission computing, software defined radios, radar systems, surface mount "pick and place" systems, and hot rolling mills.

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, such as event notification, logging, persistence, real-time scheduling, fault tolerance, and transactions, that would otherwise be implemented in an *ad hoc* manner by each application or integrator.

An example of common middleware services R&D that is relevant for DRE systems is the QuO project <http://www.dist-systems.bbn.com/tech/QuO> [Loy01] conducted by researchers at BBN Technologies as part of the DARPA ITO Quorum and PCES programs. QuO is a set of open-source middleware services based on the layered middleware architecture shown in Figure 2. The QuO 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, scaleably, 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 DRE 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.

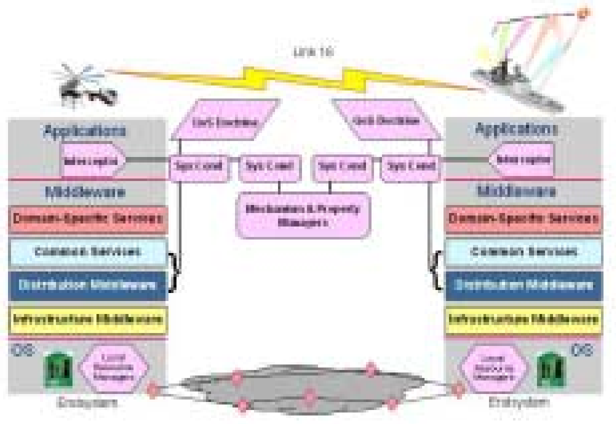

Figure 2. The QuO Architecture

The QuO middleware is responsible for collecting, organizing, and disseminating QoS-related meta-information needed to monitor and manage how well the functional interactions occur at multiple levels of DRE systems. It also enables the adaptive and reflective decision-making needed to support non-functional QoS properties robustly in the face of rapidly changing application requirements and environmental conditions, such as local failures, transient overloads, and dynamic functional or QoS reconfigurations.

Domain-specific middleware services are tailored to the requirements of particular DRE system domains, such as avionics mission computing, radar processing, online financial trading, or distributed process control. Unlike the previous three middleware layers—which provide broadly reusable "horizontal" mechanisms and services—domain-specific middleware services are targeted at vertical markets. From both a COTS and R&D perspective, domain-specific services are the least mature of the middleware layers, due in part to the historical lack of distribution middleware and common middleware service *standards* 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.

An example of domain-specific middleware services R&D that is relevant for DRE systems is the Boeing Bold Stroke architecture [Sha98], which has been used as the open experimentation platform on many DARPA ITO programs. Bold Stroke is 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 (TAO), and common middleware services (QuO and the CORBA Event Service).

Recent Progress and Future Needs

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

- ***Years of research, iteration, refinement, and successful use*** – The use of middleware and DOC middleware is not new [Sch01]. 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, with the advent of BBN's Cronus and Corbus systems. 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 funding and 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.

  Multiple interoperable and robust implementations of these CORBA capabilities and services are now available. 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 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, Sch00]. 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, such as OVM, ACE, TAO, and QuO, 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.

- ***Sustained government R&D investments*** – Much of the pioneering R&D on middleware patterns and frameworks was conducted over the past five years in the DARPA ITO Quorum and PCES programs. These programs focused heavily on CORBA and Java open systems middleware and yielded many results that transitioned into standardized service definitions and implementations for the Real-time and Fault-tolerant CORBA specification and commercialization efforts. Quorum and PCES are examples of how focused government R&D efforts 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 open 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 fundamentally reshaping the R&D and integrator communities.

Due to the advances described above, standards-based middleware has 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, these middleware applications have been relatively small-scale and statically configured DRE systems.

To satisfy the highly application- and mission-specific QoS requirements in network-centric DRE "system of system" environments, considerable additional R&D efforts are required to enhance middleware, particularly common and domain-specific middleware services. If these efforts are successful, future middleware technologies will be able to control individual and aggregate resources used by multiple system components at multiple system levels to dependably manage communication bandwidth, scheduling and allocation of DRE system artifacts, dependability, and security.

Concluding Remarks

Middleware has become strategic to developing effective distributed real-time and embedded (DRE) systems by bridging the gap between application programs and the underlying operating systems and network protocol stacks to provide reusable services whose qualities are critical to DRE systems. The economic payoffs of middleware R&D stem from moving standardization up several levels of abstraction by maturing DRE software technology artifacts, such as middleware frameworks, protocols, service components, and patterns, so that they will ultimately be available for COTS acquisition and customization. Given the proper advanced R&D context and an effective process for transitioning R&D results, the COTS middleware market will adapt, adopt, and implement the types of robust hardware and software capabilities needed for mission-critical DRE systems.

As a result of the R&D efforts described in this article—and many other similar efforts throughout academia and industry—the next generation of middleware will be able to adapt effectively to dynamically changing conditions for the purpose of utilizing the available computer and network infrastructure to the highest degree possible in support of application needs. Additional information on DRE middleware R&D efforts are available at http://www.cs.wustl.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.

[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.

[Omg01] Object Management Group, "The Common Object Request Broker: Architecture and Specification," Revision 2.6, OMG Technical Document, December, 2001.

[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.

[Sch00] Schmidt D., Stal M., Rohnert H., Buschmann F., Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Wiley and Sons, 2000.

[Sch01] Schantz R., Schmidt D., "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," Encyclopedia of Software Engineering, Wiley & Sons, 2001.

[Sch02] Schmidt D. and Huston S., "C++ Network Programming: Mastering Complexity with ACE and Patterns," Addison-Wesley, 2002.

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