Towards Adaptive and Reflective Middleware For Network-Centric Combat Systems

Douglas C. Schmidt
schmidt@uci.edu
Elec. & Comp. Eng. Dept.
Uni. of California, Irvine, CA

Richard E. Schantz
schantz@bbn.com
BBN Technologies
Cambridge, MA

Michael W. Masters
MastersMW@nswc.navy.mil
Naval Surface Warfare Center
Dahlgren, VA

Joseph K. Cross
joseph.k.cross@lmco.com
Lockheed Martin
Eagan, MN

David C. Sharp
david.sharp@boeing.com
The Boeing Company
St. Louis, MO

Louis P. DiPalma
Louis_P_DiPalma@raytheon.com
Raytheon
Portsmouth, RI

Abstract

Software is increasingly important to the development of effective network-centric DoD combat systems. Next-generation combat 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 combat 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 article describes how adaptive and reflective middleware systems (ARMS) are being developed to bridge the gap between military application programs and the underlying operating systems and communication software in order to provide reusable services whose qualities are critical to network-centric combat systems. ARMS software can adapt in response to dynamically changing conditions for the purpose of utilizing the available computer and communication resources to the highest degree possible in support of mission needs.

Motivation

New and planned DoD combat systems are inherently network-centric distributed real-time and embedded (DRE) "systems of systems." Combat 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 by tightly coupling many functional and quality of service (QoS) aspects they impede DRE system

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

In recognition of the importance of enhancing affordability, recent DoD programs, such as the Aegis destroyer program [Holzer00], the New Attack Submarine program [NAS94], the Weapons Systems Open Architecture program [Loy01], and the Unmanned Combat Air Vehicle (UCAV) program [Sha98] have adopted strong open systems approaches to system design and commercial-off-the-shelf (COTS) refresh strategies. Ultimately, open systems approaches are more likely to be robust with respect to change over the long life-cycles typical of military systems. For example, the affordability of certain types of DoD systems, such as logistics and mission planning, has been improved by using COTS technologies.

However, many of today's procurement efforts aimed at integrating COTS into mission-critical DRE combat systems have largely failed to support life-cycle 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 combat systems for foreign military sales [COTS98]. 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 combat system adaptability and assurability, *e.g.*, a perturbation in the behavior of a COTS product that would be acceptable in commercial applications could lead to loss of life and property in military applications.

Historically, conventional COTS software has been unsuitable for use in mission-critical DRE combat 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 combat 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. Developing the new generation of assurable, adaptable, and affordable COTS software technologies is therefore essential for US national security.

Although the near-term use of COTS software in DRE systems will be limited in scope and domain, the prospects for the longer term are much brighter. Given the proper advanced R&D context and an effective process for transitioning R&D results, the COTS market can adapt, adopt, and implement the types of robust hardware and software capabilities needed for military applications. This 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 successful, however, this process results in *standards* that codify the best-of-breed practices and technologies, and the *patterns and frameworks* that reify the knowledge of how to apply these practices and technologies.

Key Technical Challenges and Solutions

Today's economic and organizational constraints—along with increasingly complex requirements and competitive pressures—make it infeasible to build complex distributed real-time system software entirely from scratch. It has long been accepted that the use of commercial operating systems and communication support software is cost-effective for all but the most resource-constrained DRE systems. Increasingly, this same logic is being applied to *middleware*, which is reusable service/protocol component and framework software that services end-to-end and aggregate combat systems needs [Sch01a]. Middleware bridges the gap between

1. Application-level requirements and mission doctrine and
2. The lower-level underlying localized viewpoints of the operating systems and communications support mechanisms.

From the application perspective, when middleware and the services it constitutes are combined with traditional network and operating system components, it forms the new infrastructure for developing modern network-centric combat systems. In both commercial and military systems, middleware performs functions that are essential to meet application-level requirements. In military systems, moreover, the qualities of the services provided by the middleware are critical to the qualities of service that are presented to the end users – the warfighters.

Thus, there is a pressing need to develop, validate, and ultimately standardize a new generation of *adaptive and reflective middleware systems* (ARMS) technologies that will be readily available and able to support stringent combat system functionality and QoS requirements. Some of the most challenging computing and communication requirements for new and planned DoD combat 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, *e.g.,* the security and dependability of message transmission must be traded off against latency and predictability, 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.

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 combat 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 in providing the means for examining the capabilities it offers while the system is running, thereby enabling automated adjustment for optimizing 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 combat 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.

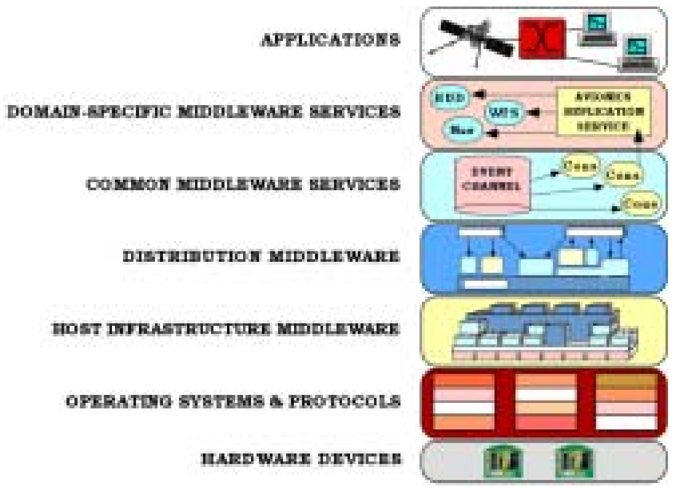
APPLICATIONS
DOMAIN-SPECIFIC MIDDLEWARE SERVICES
HUD
WTS
Nav
AVIONICS REPLICATION SERVICE
COMMON MIDDLEWARE SERVICES
EVENT CHANNEL
Cons
Cons
Cons
DISTRIBUTION MIDDLEWARE
HOST INFRASTRUCTURE MIDDLEWARE
OPERATING SYSTEMS & PROTOCOLS
HARDWARE DEVICES

Figure 1. Middleware Layers and Their Surrounding Context

We describe each of these middleware layers below 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 combat systems.

Host infrastructure middleware encapsulates and enhances native operating system 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 operating system programming application program interfaces (APIs), such as Sockets or POSIX Pthreads. Examples of COTS host infrastructure middleware that are relevant for DRE combat systems include:

- *The ADAPTIVE Communication Environment* (ACE) [Sch01], which is a portable and efficient toolkit that encapsulates native operating system network programming capabilities, such as interprocess communication, static and dynamic configuration of application components, and synchronization. ACE has been used in a wide range of DoD DRE systems, including missile control, avionics mission computing, software defined radios, and radar systems.
- *Real-time Java Virtual Machines*, 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 deal with memory management and concurrency. Although RTSJ implementations are still in their infancy, they have generated tremendous interest in the DoD R&D and integrator communities due to their potential for reducing software development and evolution costs significantly.

Distribution middleware defines a higher-level distributed programming model whose reusable application program interfaces and mechanisms automate and extend the native operating system 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 or distributed components.

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, Sch98]. CORBA is distribution middleware that allows objects to interoperate across networks without hard-coding dependencies on their location, programming language, operating system platform, communication protocols and interconnects, and hardware characteristics. In 1998 the OMG adopted the Real-time CORBA specification [Sch00a], which extends CORBA with features that allow DRE applications to reserve and manage CPU, memory, and networking resources. Real-time CORBA implementations have been used in dozens of DoD combat systems, including avionics mission computing [Sha98], submarine combat control systems [DiPalma99], signal intelligence and C4ISR systems, software defined radios, and radar systems.

Common middleware services augment distribution middleware by defining higher-level, domain-independent, reusable services that have proven necessary in most distributed application contexts to deal with multi-computer environments effectively. In addition, these services provide components that allow application developers to concentrate on programming application logic, without the need to write the "plumbing" code needed to develop distributed applications using lower level middleware features directly. For example, 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 reimplemented 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, naming, security, and fault tolerance. Not all of these standard services are sufficiently refined today to be usable off-the-shelf for DRE combat systems. However, the form and content of these common middleware services will continue to mature and evolve to meet the expanding requirements of DRE.

Domain-specific middleware services are tailored to the requirements of particular combat 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 market segments. 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 DoD integrators require to develop particular classes of DRE combat systems.

A mature example of domain-specific middleware services appears in the Boeing Bold Stroke architecture [Sha98]. Bold Stroke uses COTS hardware and middleware to produce a non-proprietary, standards-based component architecture for military avionics mission computing capabilities, such as navigation, data link management, and weapons control. A driving objective of Bold Stroke was to support reusable product-line applications, leading to a highly configurable application component model and supporting middleware services. The domain-specific middleware services in Bold Stroke are layered upon common middleware services (the CORBA Event Service), distribution middleware (Real-time CORBA and the TAO ORB [Sch98]), and infrastructure middleware (ACE), and have been demonstrated to be highly portable for different COTS operating systems (*e.g.,* VxWorks), interconnects (*e.g.,* VME), and processors (*e.g.,* PowerPC).

Recent Progress

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

- ***The maturation of standards*** – Over the past decade, middleware standards have been established and have matured considerably with respect to DRE requirements. For example, the OMG has adopted the following DRE-related specifications recently:
  - *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 and interoperable implementations of these CORBA capabilities and services are now available from multiple vendors. Moreover, emerging standards such as Dynamic Scheduling Real-Time CORBA, Real-time CORBA publish-subscribe services, 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 DoD 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, architects, and systems engineers who have already confronted similar problems successfully.
  - *Frameworks* are concrete realizations of related patterns [John97] that provide an integrated set of components that collaborate to provide a reusable architecture for a family of related applications. Middleware frameworks 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 DoD DRE combat 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, which complicates subsequent software evolution and optimization.

Middleware patterns and frameworks help address these problems by systematically capturing combat system design expertise in a readily accessible and reusable format, thereby raising the level at which systems engineers and application developers approach the decision making and implementation of their systems. Two efforts to provide suitable guidance for the development of military systems are the New Attack Submarine (NAS) [NAS94] and the Aegis Shipbuilding Program. NAS developed a guidance document detailing allowable standards for the NAS C^3I system, and the Aegis program developed a guidance document for Baseline 7 phase I [Aegis7]. These documents were instrumental in guiding the design of these systems.

Much of the pioneering R&D on middleware patterns, frameworks, and standards for DRE combat systems has been conducted in the DARPA Information Technology Office (ITO) *Quorum* program [DARPA99], which played a leading role in:

- Demonstrating the viability of host infrastructure middleware and distribution middleware for DoD combat systems by providing the foundation for managing key QoS attributes, such as real time behavior, dependability and system survivability, from a network-centric middleware perspective
- Transitioning a number of new middleware perspectives and capabilities into DoD acquisition programs [Sha98, AegisOA] and commercially supported products and
- Establishing the technical viability of collections of systems that can dynamically adapt [Loy01] their collective behavior to varying operating conditions, in service of delivering the appropriate application level response under these different conditions.

The Quorum 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 by using 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 DoD R&D and integrator communities.

Successful DoD technology transition most often results from a partnership between technology developers and technology users. One of the most successful examples of such partnerships is the joint DARPA/Aegis High Performance Distributed Computing program (HiPer-D). Through the use of prototyping and system-scale experiments, this program has demonstrated the effectiveness of a number of DARPA and standards-based COTS technologies for building DRE combat systems that are efficient, scalable, fault tolerant, and flexible in their design and operation.

Looking Ahead

Due to advances in COTS technologies outlined earlier, host infrastructure middleware and distribution middleware have now been demonstrated and deployed in a number of mission-critical DRE combat systems. Since off-the-shelf middleware technology has not yet matured to cover the realm of large-scale, dynamically changing systems, however, COTS 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 must therefore be enhanced to support common and domain-specific middleware services that can manage the following resources effectively:

- *Communication bandwidth, e.g.,* network level status information and management services, scalability to 10^2 subnets and 10^3 nodes, and dynamic connections with reserved bandwidth to enhance real-time predictability.
- *Distributed real-time scheduling and allocation of DRE system artifacts (such as CPUs, networks, UAVs, missiles, torpedoes, radar, illuminators, etc), e.g.,* fast and predictable behavior of widely dispersed components using managed communication capabilities and bandwidth reservations.
- *Distributed system dependability, e.g.,* policy-based selection of replication options.
- *Distributed system security, e.g.,* dynamically variable object access control policies and effective, combined real-time, dependability, and security interactions.

Ironically, there is little or no scientific underpinning for QoS-enabled resource management, despite the demand for it in most distributed systems [Narain01]. Today's system designers 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 combat systems.

To address this problem, 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 combat systems depend upon by creating middleware and distributed resource management 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 DoD standards that are relevant for DRE combat systems. However, the protocol and service *implementations* should be customizable—statically and dynamically—for specific DoD DRE combat system requirements.

To achieve these goals, middleware technologies and tools need to be based upon some type of layered architecture along with QoS adaptive middleware services, such as the one shown in Figure 2 and based on empirical investigations of this type of capability [Loy01]. The Quality Objects (QuO) [ZBS97] project is an example of such a layered architecture designed to manage and package adaptive QoS capabilities as common middleware services. 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, 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 attribute 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 are less than the expected resources and
  3. The failure detection and recovery strategies necessary to meet end-to-end dependability requirements.

In next-generation combat systems, the middleware—rather than operating systems or networks in isolation—will be responsible for separating DRE system QoS attribute 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 application.

The architecture in Figure 2 is based on the expectation that QoS attribute 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 that is 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 QoS attribute properties robustly in the face of rapidly changing mission requirements and environmental conditions.

Researching and developing these middleware capabilities is crucial to ensure that the aggregate behavior of future network-centric combat systems is dependable, despite local failures, transient overloads, and dynamic functional or QoS reconfigurations.

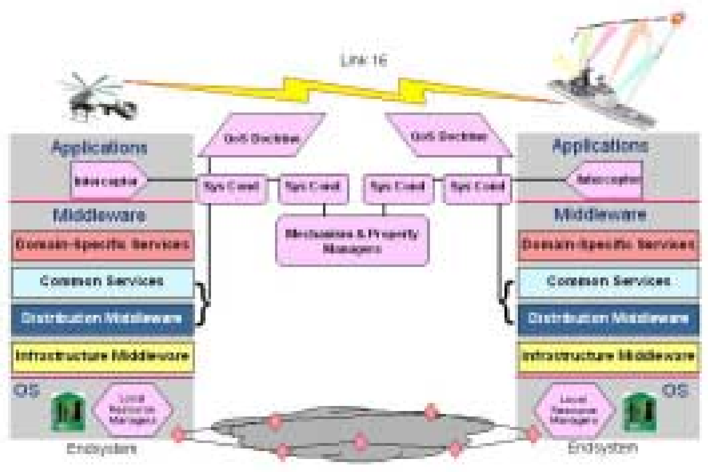

Figure 2. Decoupling Functional and QoS Attribute Paths in QuO

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 Task Force
- The Java Expert Group Real-time Specification for Java (RTSJ) and the emerging Distributed RTSJ and
- The IEEE Real-time Portable Operating System (POSIX) specification.

Concluding Remarks

As a result of much previous R&D and transition experience, network-centric systems today are constructed as a series of layers of intertwined technical capabilities and innovations. The main emphasis at the lower layers is in providing the core computing and communication resources and services that drive network-centric computing: the individual computers, the networks, and the operating systems that control the individual host and the message level communication.

At the upper layers, various types of middleware are starting to bridge the previously formidable gap between the lower-level resources and services and the abstractions that are needed to program, organize, and control systems composed of coordinated, rather than isolated, components. Key capabilities in the upper layers include common and domain-specific middleware services that

- Enforce real-time behavior across computational nodes
- Manage redundancy across elements to support dependable computing and
- Control end-to-end adaptive behavior as responses to changes in operating conditions.

These new middleware services make the coordinated use of multiple computing elements feasible and affordable by controlling the hardware, network, and endsystem mechanisms that affect mission, system, and application QoS delivery and tradeoffs.

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

- Devising optimizers, meta-programming techniques, and multi-level distributed dynamic resource management protocols and services for ARMS that will enable DoD DRE systems to configure standard COTS interfaces, without the penalties incurred by today's conventional COTS software product implementations. Many network-centric DoD combat systems require these DRE middleware capabilities.
- Standardizing COTS at the middleware level, rather than just at lower hardware/networks/operating system levels. The primary economic benefits of middleware stem from their extending standardization up several levels of abstraction so that DRE middleware technology is readily available for COTS acquisition and customization.

As COTS implementations of middleware standards mature in their functional quality and quality of service, they are helping to lower the total ownership costs of combat systems. For example, Real-time and Fault-tolerant CORBA implementations are creating a common base of COTS technology that enables complex DRE middleware capabilities to be reconfigured and reused, rather than re-invented repeatedly or reworked from proprietary "stove-pipe" architectures that are inflexible and expensive to evolve and optimize. Additional information on middleware for DRE systems is available at http://www.ece.uci.edu/~schmidt/TAO.html.

About the Authors

Dr. Schmidt is an Associate Professor in the Electrical and Computer Engineering Department at the University of California, Irvine. He currently serves as a Program Manager at DARPA ITO, where he leads the national effort on distributed object computing middleware R&D. His research focuses on design patterns, implementation, and experimental analysis of object-oriented frameworks that facilitate the development of high-performance, real-time distributed object computing systems on parallel processing platforms running over high-speed networks and embedded system interconnects.

Dr. Schantz is a Principal Scientist at BBN Technologies in Cambridge, Massachusetts. His research has been instrumental in defining and evolving the concepts underlying middleware since its emergence in the early days of the Internet. He was directly responsible for developing the first operational distributed object computing capability and transitioning it to production use. More recently, he has led research efforts toward developing and demonstrating the effectiveness of middleware support for adaptively managed QoS control, as Principal Investigator on a number of key DARPA ITO projects.

Michael W. Masters serves as Chief Scientist for the U.S. Navy's High Performance Distributed Computing program (HiPer-D), a joint effort between the Aegis shipbuilding program and several DARPA ITO programs. HiPer-D is defining a new distributed real-time computing architecture for shipboard use. Mr. Masters is co-inventor of a technology called dynamic resource management, an enterprise-wide system control capability that allows large-scale real-time systems to dynamically reconfigure themselves to adapt to varying environments, changing mission demands and current resource availability.

Dr. Cross is a Senior Staff System Engineer at Lockheed Martin Tactical Systems, Eagan. He is currently serving as Principal Investigator of the Meta-Interfaces for Embedded Real-time Systems (MINERS) project in DARPA ITO. His other activities focus on middleware for Navy standard products and mechanisms for automatic configuration of complex communication systems.

David Sharp is a Technical Fellow at Boeing Phantom Works in St. Louis, MO, USA. As Lead Architect and Core Architecture Team Leader for Boeing's Bold Stroke product line avionics software initiative, David spearheaded the development, documentation, and presentation of the Bold Stroke Software Architecture, a reusable product-line software architecture used as the basis for avionics program work on a range of Boeing production and experimental aircraft programs, and as the foundation for several highly influential US government-sponsored R&D programs. David serves as Principal Investigator for a number of DARPA ITO and AFRL programs.

Mr. DiPalma is the Manager of the SubSurface Warfighter Information Center Systems Engineering Department of the Portsmouth, RI Headquarters of the Naval & Maritime Integrated Systems Operation of the Raytheon Electronic Systems Company. Lou has been involved in the design and development of Submarine Combat Control Systems including the New Attack Submarine (NSSN) CC, the Combat Control System (CCS) Mk 2 and AN/BSG-1 Weapon Launching System Programs. He has been actively involved with the infusion of new technology into the aforementioned systems, including CORBA and RT-CORBA.

References

[Aegis7] Guidance Document for Aegis Baseline 7 Phase 1 and II Specification Development: Information Architecture and Baseline Applicability, Version 1.0, 20 March 1998.

[AegisOA] Guidance Document for Aegis Open Architecture Baseline Specification Development, Version 2.0 (Draft), 5 July 2001.

[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.

[COTS98] Clapp J., Taub A., “A Management Guide to Software Maintenance in COTS-Based Systems,” MP 98B0000069, The MITRE Corporation, Bedford, MA, November 1998.

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

[DiPalma99] DiPalma, L., “The Infusion of CORBA into the U.S. Navy’s Submarine Fleet”, *Software Technology Conference*, May 1999.

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

[Holzer00] Holzer R., “U.S. Navy Looking for More Adaptable Aegis Radar,” *Defense News*, 18 September 2000.

[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.

[Narain01] Narain S., Vaidyanathan R., Moyer S., Stephens W., Parameswaran K., and Shareef A., “Middleware For Building Adaptive Systems via Configuration,” ACM Optimization of Middleware and Distributed Systems (OM 2001) workshop, Snowbird, Utah, June, 2001.

[NAS94] New Attack Submarine Open System Implementation, Specification and Guidance, August 1994.

[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.

[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. and 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.

[ZBS97] Zinky JA, Bakken DE, Schantz RE. "Architectural Support for Quality of Service for CORBA Objects", Theory and Practice of Object Systems (TAPOS), Volume 3, Number 1, April 1997.