Middleware R&D Challenges for Distributed Real-time and Embedded Systems

Douglas C. Schmidt and Aniruddha Gokhale
Electrical Engineering & Computer Science Department
Vanderbilt University
Nashville, TN 37203, USA
{d.schmidt,a.gokhale}@vanderbilt.edu

Richard E. Schantz and Joseph P. Loyall
BBN Technologies
10 Moulton Street
Cambridge, MA 02138, USA
{schantz, jloyall}@bbn.com

Introduction

Some of the most challenging problems facing the embedded systems community are those associated with producing software for real-time and embedded systems in which computer processors may control physical, chemical, or biological processes or devices. Examples of such systems include airplanes, automobiles, nuclear reactors, oil refineries, and patient monitors, or even CD players and cellular phones. In most of these systems, *the right answer delivered too late becomes the wrong answer*, i.e., achieving end-to-end quality of service (QoS) is essential. In addition, embedded devices have historically had limited memory (e.g., 64-512 KB) available for the platform and applications.

Although real-time and embedded systems have historically been relatively small-scale and standalone, the trend is toward significantly increased functionality, complexity, and scalability. In particular, real-time and embedded systems are increasingly being connected via wireless and wireline networks to create large-scale distributed real-time and embedded (DRE) systems, such as tele-immersion environments, fly-by-wire aircraft, industrial process automation, and total ship computing environments. These DRE systems include many interdependent levels, such as network/bus interconnects, many coordinated local and remote endsystems, and often multiple layers of software, that together derive the following challenges:

- As *distributed systems*, DRE systems require capabilities to manage connections and message exchange between (possibly heterogeneous) networked computing devices.
- As *real-time systems*, DRE systems require predictable and efficient control over end-to-end system resources, such as memory, CPU, and network bandwidth.
- As *embedded systems*, DRE systems have size, weight, cost, and power constraints that often limit their computing and memory resources. For example, embedded systems often cannot use conventional virtual and automatic memory techniques because of space or timing constraints, since software must fit on low-capacity storage media, such as EEPROM or NVRAM.

DRE systems have historically been developed and validated using relatively static development and analysis techniques (such as function-oriented design and rate monotonic analysis) to implement, allocate, schedule, and manage their resources. These static approaches have proven to be acceptable for *closed* DRE systems, where the set of application tasks that will run in the system and the loads they will place on system resources change infrequently and are known in advance. They are not well-suited, however, for the next-generation of *open* DRE systems, which evolve more rapidly and must collaborate with multiple remote sensors, provide on-demand browsing and actuation capabilities for human operators, and respond flexibly to unanticipated situational factors that arise at run-time.

Many of the most challenging next-generation DRE systems will operate in large-scale DRE configurations that take input from large numbers of remote sensors and provide geographically dispersed operators with the ability to interact with the collected information and to control remote effectors. In circumstances where the presence of humans in the loop is too expensive or their responses are too slow, these systems must respond autonomously and flexibly to unanticipated combinations of events at run-time. Moreover, these systems are increasingly being networked to form long-lived "systems of systems" that must run unobtrusively and largely autonomously, shielding operators from unnecessary details, while simultaneously communicating and responding to mission-critical information at heretofore infeasible rates. Examples of these types of systems include (but are not limited to) metropolitan area traffic control systems that process sensor data from 1,000s of vehicles, coordinated swarms of unmanned air vehicles, command and control systems for theater-level battle management, home power management, and terrorist tracking and identification systems. In such systems, it is hard to enumerate, or often even approximate, all possible physical system configurations or workload mixes *a priori*.

Desirable properties of DRE systems include predictability, controllability, and adaptability of operating characteristics for applications with respect to such features as time, quantity of information, accuracy, confidence, and synchronization. All these issues become highly volatile in large-scale systems, due to the dynamic interplay of the many interconnected parts that are often constructed from smaller parts. While it is possible *in theory* to develop these types of complex systems from scratch, contemporary economic and organizational constraints, as well as increasingly complex requirements and competitive pressures, make it infeasible to do so *in practice*.

To address the many competing design forces and run-time QoS demands, sustained R&D efforts on comprehensive software methodologies, design-/run-time environments, and hardware/software co-design are required to dependably compose large, complex, interoperable DRE systems from QoS-enabled reusable components. Moreover, the components themselves must be sensitive to the environments in which they are packaged, deriving the need for runtime QoS

enabled components as well. Ultimately, what is desired is to take components that are built independently by different groups at different times and assemble them to create complete DRE systems that are customized for their requirements and environmental conditions. In the longer run, each complete system often becomes a component embedded in still larger systems of systems. Given the complexity of this undertaking, various tools and techniques are needed to configure and reconfigure these systems systematically so they can adapt to a wider variety of situations than has historically been possible with earlier generations of standalone real-time and embedded systems.

Recent Progress and Current Status

Over the past decade, various technologies have been devised to alleviate many complexities associated with developing software for DRE systems. Their successes have added a new category of systems software to the familiar operating system, programming language, and networking offerings of the previous generation. In particular, some of the most successful of emerging technologies have centered on *middleware*, which is systems software that resides between the applications and the underlying operating systems, network protocol stacks, and hardware. The primary role of middleware is to

1. Functionally bridge the gap between application programs and the lower-level hardware and software infrastructure in order to coordinate how parts of applications are connected and how they interoperate.
2. Enable and simplify the integration of components developed by multiple technology suppliers.
3. Provide a common reusable accessibility for functionality and patterns that formerly were placed directly in applications, but in actuality are application independent and need not be developed separately for each new application.

Middleware was invented originally to help simplify the development and management of distributed computing systems, and bring those capabilities within the reach of many more developers than the few experts at the time who could master the complexities of these environments. Middleware was necessary since complex system integration requirements were not being met from either (1) the *application perspective*, where it was too difficult and not reusable, or (2) the *network or host operating system perspectives*, which were necessarily concerned with providing the communication and endsystem resource management layers, respectively.

Although there are many types of middleware platforms, their architectures are generally composed of relatively autonomous software components that can be distributed or collocated throughout a range of networks and bus interconnects. Clients invoke operations on target components to perform interactions and invoke functionality needed to achieve application goals. When implemented properly, middleware can help to:

- Shield software developers from low-level, tedious, and error-prone platform details, such as socket-level network programming.
- Amortize software lifecycle costs by leveraging previous development expertise and capturing implementations of key patterns in reusable frameworks, rather than rebuilding them manually for each use.
- Provide a consistent set of higher-level network-oriented abstractions that are much closer to application requirements to help simplify the development of distributed systems.
- Provide different communication paradigms, such as request-response, asynchronous messaging and publish/subscribe, that can be used to address different application QoS requirements.
- Provide a wide array of reusable developer-oriented services, such as logging and security that have proven necessary to operate effectively in a networked environment.
- Amortize software lifecycle costs by leveraging previous development expertise and capturing implementations of key patterns in reusable frameworks, rather than rebuilding them manually for each use.

Some notable successes in the middleware domain include:

- Component middleware (such as Java 2 Enterprise Edition (J2EE), CORBA, and .NET), which have introduced advanced software engineering capabilities to the mainstream IT community and which incorporate various levels of middleware as part of the overall development process.
- World Wide Web middleware standards (such as web servers, HTTP protocols, and web services frameworks), which enable easily connecting independently developed browsers and web pages.
- Grid computing (such as Globus), which enables scientists and high performance computing researchers to collaborate on grand challenge problems, such as global climate change modeling.

Unresolved Issues

Despite all the advances in the past decade (which have largely been applied to simplify desktop and enterprise business computing), there remain significant challenges to applying today's middleware to meet the needs of new and planned DRE systems. For example, conventional component middleware has only partial support for performance-critical DRE systems. The world wide web is often the world wide *wait*, because little systems engineering or attention has been paid to enforcing end-to-end QoS issues, and Grid computing architectures and tools are not aligned with mainstream COTS middleware and moreover do not support the stringent QoS requirements of DRE systems. More fundamentally, there are no mature engineering principles, solutions, or established conventions to enable large-scale DRE systems to be repeatably, predictably, and cost effectively created, developed, validated, operated, and enhanced. As a result, we are witnessing a complexity thresh-

old that is stunting our ability to create large-scale DRE systems successfully.

Some of the inherent complexities that contribute to complexity threshold of DRE systems include: (1) discrete platforms that must be scaled to provide seamless end-to-end solutions, (2) integration of heterogeneous components is the norm, (3) partial failures of distributed components are the norm, (4) dynamically changing operating environments and configurations are the norm, (5) large-scale systems must operate continuously, even during upgrades, (6) end-to-end properties must be satisfied in time and resource constrained environments, and (7) maintaining system-wide QoS concerns is becoming expected. To address these complexities, we must create and deploy a new generation of middleware-oriented solutions and engineering principles as part of the commonly available software infrastructure that is needed to develop, validate, and deploy many different types of large-scale DRE systems successfully.

Specific R&D Challenges

An essential part of what is needed to alleviate the inherent complexities outlined above is the integration and extension of concepts and capabilities that have been found traditionally in network management, data management, distributed operating systems, and object-oriented programming languages. The payoff will be reusable middleware that significantly simplifies the development and evolution of large-scale DRE systems. The following are some specific R&D challenges associated with achieving this payoff:

- ***Demand for end-to-end QoS support, not just component-level QoS***. This area represents the next great wave of evolution for middleware. There is now widespread recognition that effective development of large-scale DRE applications requires the use of COTS infrastructure and service components. Moreover, the (re)usability of the resulting products depends heavily on the properties of the whole as derived from its parts. This type of environment requires *visible*, *predictable*, *flexible*, and *integrated* resource management strategies within and between the pieces. Despite the ease of connectivity provided by middleware, however, constructing integrated DRE systems remains hard since it requires significant customization of non-functional QoS properties, such as predictable latency/jitter/throughput, scalability, dependability, and security. In their most useful forms, these properties extend end-to-end and thus have elements applicable to (1) the network substrate, (2) the platform operating systems and system services, (3) the programming system in which they are developed, (4) the applications themselves, and (5) the middleware that integrates all these elements together. The basic premises underlying the push towards end-to-end QoS support mediated by middleware are that different levels of service are possible and desirable under different conditions and costs and the level of service in one property must be coordinated with and/or traded off against the level of service in another to achieve the intended overall results.

- ***Adaptive and reflective solutions that handle both variability and control.*** DRE systems today often work well as long as they receive all the resources for which they were designed in a timely fashion, but fail completely under the slightest anomaly. There is little flexibility in their behavior, i.e., most of the adaptation is pushed to end-users or administrators. Instead of hard failure or indefinite waiting, what is required is either *reconfiguration* to reacquire the needed resources automatically or *graceful degradation* if they are not available. Reconfiguration and operating under less than optimal conditions both have two points of focus: individual and aggregate behavior. Moreover, there is a need for interoperability of control and management mechanisms. To date interoperability concerns have focused on data interoperability and invocation interoperability. Little work has focused on mechanisms for controlling the overall behavior of integrated DRE systems, which is needed to provide "control interoperability." There are requirements for interoperable control capabilities to appear in individual resources first, after which approaches can be developed to aggregate these into acceptable global behavior.

  To manage the broader range of QoS demands for next-generation DRE systems, middleware must become more adaptive and reflective. *Adaptive middleware* is software whose functional and QoS-related properties can be modified either: (1) *statically*, *e.g.*, to reduce footprint, leverage capabilities that exist in specific platforms, enable functional subsetting, and minimize hardware/software infrastructure dependencies or (2) *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 such modifications dependably, *i.e.*, while meeting stringent end-to-end QoS requirements. *Reflective middleware* goes further to permit automated examination of the capabilities it offers, and to permit automated adjustment to optimize those capabilities. Reflective techniques make the internal organization of systems–as well as the mechanisms used in their construction–both visible and manipulatable for middleware and application programs to inspect and modify at run-time. Reflective middleware therefore supports more advanced adaptive behavior and more dynamic strategies keyed to current circumstances, *i.e.*, necessary adaptations can be performed autonomously based on conditions within the system, in the system' s environment, or in system QoS policies defined by administrators or end-users.

- ***More universal adoption of standard middleware.*** Today, it is too often the case that a substantial percentage of the effort expended to develop DRE systems goes into building *ad hoc* and proprietary middleware, or additions for missing middleware functionality. As a result, subsequent composition of these *ad hoc* capabilities is either infeasible or prohibitively expensive. One reason why rede-

velopment persists is that it is still often relatively easy to pull together a minimalist *ad hoc* solution, which remains largely invisible to all except the developers. Unfortunately, this approach can yield substantial recurring life-cycle costs, particularly for complex and long-lived DRE systems. One of the most immediate challenges is therefore to establish and eventually standardize middleware interfaces that support QoS attributes. It is important to have a clear understanding of the QoS information so that it becomes possible to identify the users' requirements at any particular point in time and understand whether or not these requirements are being (or even can be) met.

It is also essential to aggregate these requirements, making it possible to form decisions, policies, and mechanisms that begin to address a more global information management organization. Meeting these requirements will require flexibility on the parts of both the application components and the middleware resource management strategies used across heterogeneous systems of systems. A key direction for addressing these needs is through the concepts associated with managing adaptive behavior, recognizing that not all requirements can be met all of the time, yet still ensuring predictable and controllable end-to-end behavior.

- ***Leveraging and extending the installed base.*** In addition to the R&D challenges described above, there are also pragmatic considerations, including incorporating new QoS-enabled middleware interfaces and implementations to various building blocks that are already in place for the networks, operating systems, security, and data management infrastructure, all of which continue to evolve independently. Ultimately, there are two different types of resources that must be considered: (1) those that will be fabricated as part of application development and (2) those that are provided and can be considered part of the substrate currently available.

  While not much can be done in the short-term to change the direction of the hardware and software substrate that's installed today, a reasonable approach is to provide the needed services at higher levels of middleware-based abstraction. This architecture will enable new components to have properties that can be more easily included into the controllable applications and integrated with each other, leaving less lower-level complexity for application developers to address and thereby reducing system development and ownership costs. Consequently, the goal of next-generation middleware for DRE systems is not simply to build a better network, better resource manager, or better security service in isolation, but rather to pull these capabilities together and deliver them to applications in ways that enable them to realize this model of adaptive behavior with tradeoffs between the various QoS attributes. As the evolution of the underlying system components change to become more controllable, we can expect a refactoring of the implementations underlying the enforcement of adaptive control.

Middleware Research Areas for DRE Systems

The following concepts are central to addressing the R&D challenges described above:

- ***Contracts via meta-programming.*** Information must be gathered for particular applications or application families regarding user requirements, resource requirements, and system conditions. Multiple system behaviors must be made available based on what is best under the various conditions. This information provides the basis for the contracts between users and the underlying system substrate. These contracts provide not only the means to specify the degree of assurance of a certain level of service, but also provide a well-defined, high-level middleware abstraction to improve the visibility of adaptive changes in the mandated behavior. Model-based engineering techniques can provide the means to model these contracts while providing the ability to analyze and verify them for system correctness. Generative techniques can then be used to synthesize appropriate middleware artifacts that applications can use for their adaptive behavior.
- ***Adaptive control and graceful degradation.*** Well-established theory and practice on control engineering can be applied to monitor DRE systems and enforce contracts via feedback or feedforward techniques so that application services can adapt their behavior or degrade gracefully (or augment) as conditions change, according to a prearranged contract governing that activity. The initial challenge here is to establish the idea in the minds of developers and users that multiple behaviors are both feasible and desirable. The next step is to put into place the additional middleware support – including connecting to lower-level network and operating system enforcement mechanisms – necessary to provide the right behavior effectively and efficiently given current system conditions.
- ***Prioritization and physical world constrained load invariant performance.*** Some systems are highly correlated with physical constraints and have little flexibility in some of their requirements for computing assets, including QoS. Deviation from requirements beyond a narrowly defined error tolerance can sometimes result in catastrophic failure of the system. The challenge is in meeting these *invariants* under varying load conditions. This often means guaranteeing access to some resources, while other resources may need to be diverted to insure proper operation. Generally collections of such components will need to be resource managed from a system (aggregate) perspective in addition to a component (individual) perspective.

Although it is possible to satisfy contracts, achieve graceful degradation, and globally manage some resources to a limited degree in a limited range of systems today, much R&D work remains. The strategies needed to deliver these goals can be divided into the seven research areas described below:

1. ***Individual QoS requirements***, which deals with developing mechanisms related to the end-to-end QoS needs from

the perspective of a single user or application. The specification requirements include multiple contracts, negotiation, and domain specificity. Multiple contracts are needed to handle requirements that change over time and to associate several contracts with a single perspective, each governing a portion of an activity. This capability is particular important for mobile applications, where the bandwidth restrictions of wireless connectivity and interferences may require a significant number of coordinated contracts. Different application operation modes or users running the same application in different configurations may have different QoS requirements emphasizing different benefits and tradeoffs, often depending on current configurations. Such dynamic behavior must be taken into account and introduced seamlessly into next-generation DRE systems.

General negotiation capabilities that offer convenient mechanisms to enter into and control a negotiated behavior (as contrasted with the service being negotiated) need to be available as COTS middleware packages. The most effective way for such negotiation-based adaptation mechanisms to become an integral part of QoS is for them to be "user friendly," *e.g.*, requiring a user or administrator to simply provide a list of preferences via higher level abstractions, such as modeling tools.. This area is likely to become domain-specific and even user-specific, but can be enabled by general purpose, common tools. Other challenges that must be addressed as part of delivering QoS to individual applications include: (1) translation of requests for service among and between the various entities on the distributed end-to-end path, (2) managing the definition and selection of appropriate application functionality and system resource tradeoffs within a "fuzzy" environment, and (3) maintaining the appropriate behavior under composability.

Translation addresses the fact that complex DRE systems are being built in layers. At various levels in a layered architecture the user-oriented QoS must be translated into requests for other resources at a lower level. A key R&D challenge is how to accomplish this translation from user requirements to system services. A logical place to begin is at the application/middleware boundary, which closely relates to the problem of matching application resources to appropriate distributed system resources. As system resources change in significant ways, either due to anomalies or load, tradeoffs between QoS attributes (such as timeliness, precision, and accuracy) may need to be (re)evaluated to ensure an effective level of QoS, given the circumstances. Mechanisms need to be developed to identify and perform these tradeoffs at the appropriate time. Last, but certainly not least, a theory of effectively composing systems from individual components in a way that maintains application-centric end-to-end properties needs to be developed, along with efficient realizations of the theory in the form of working middleware.

2. ***Run-time requirements***. From a system lifecycle perspective, decisions for managing QoS are made at design time, at configuration/deployment time, and/or at run-time. Of these, the run-time requirements are the most challenging since they have the shortest time scales for decision-making, and collectively we have the least experience with developing appropriate solutions. They are also the area most closely related to advanced middleware concepts. This area of research addresses the need for run-time monitoring, feedback, and transition mechanisms to change application and system behavior, *e.g.*, through dynamic reconfiguration, orchestrating degraded behavior, or even off-line recompilation. The primary requirements here are *measurement*, *reporting*, *control*, *feedback*, and *stability*. Each of these plays a significant role in delivering end-to-end QoS, not only for an individual application, but also for an aggregate system. A key part of a run-time environment centers on a permanent and highly tunable measurement and resource status services as a common middleware capabilities, oriented toward various granularities for different time epochs and with abstractions and aggregations appropriate to its use for run-time adaptation.

   In addition to providing the capabilities for enabling graceful degradation, these same underlying mechanisms also hold the promise to provide flexibility that supports a variety of possible behaviors, without changing the basic implementation structure of DRE applications. This reflective flexibility reduces the dependence on diminishes the importance of many initial design decisions by offering late- and run-time-binding options to accommodate actual operating environments at the time of deployment, instead of only anticipated operating environments at design time. In addition, it anticipates changes in these bindings to accommodate new behavior.

3. ***Aggregate requirements***. This area of research deals with the system view of collecting necessary information over the set of resources across the system, and providing resource management mechanisms and policies that are aligned with the goals of the system as a whole. While middleware itself cannot manage system-level resources directly (except through interfaces provided by lower-level resource management and enforcement mechanisms), it can provide the coordinating mechanisms and policies that drive the individual resource managers into domain-wide coherence. With regards to such resource management, policies need to be in place to guide the decision-making process and the mechanisms to carry out these policy decisions.

   Areas of particular R&D interest include: (1) *reservations*, which allow resources to be reserved to assure certain levels of service such as maintaining end-to-end priorities, (2) *admission control mechanisms*, which allow or reject certain users access to system resources, (3) *enforcement mechanisms* with appropriate scale, granularity and performance, and (4) *coordinated strategies and policies* to allocate distributed resources that optimize various properties. Moreover, policy decisions need to be made to allow for varying levels of QoS, including whether

each application receives guaranteed, best-effort, conditional, or statistical levels of service. Managing property composition is essential for delivering individual QoS for component based applications, and is of even greater concern in the aggregate case, particularly in the form of layered resource management within and across domains.

4. ***Integration requirements***. Integration requirements address the need to develop interfaces with key building blocks used for system construction, including the OS, network management, security, and data management. Many of these areas have partial QoS solutions underway from their individual perspectives. The problem today is that these partial results must be integrated into a common interface so that users and application developers can tap into each, identify which viewpoint will be dominant under which conditions, and support the tradeoff management across the boundaries to get the right mix of attributes. Currently, tools working with component middleware provide end-to-end syntactic interoperation and relatively seamless linkage across the networks and subsystems. There is no *managed* QoS, however, making these tools and middleware useful largely for resource rich, best-effort environments.

   To meet varying requirements for integrated behavior, advanced tools and mechanisms are needed that permit requests for *different* levels of attributes with different tradeoffs governing this interoperation. Model-based analysis, verification, and generative tools can provide the system (re)configuration and integration needed to deliver the requested end-to-end QoS. Likewise, they can indicate the inability to deliver that level of service, perhaps offering to support an alternative QoS, or triggering application-level adaptation. For all of this to work together properly, multiple dimensions of the QoS requests must be understood within a common framework to translate and communicate those requests and services at each relevant interface. Advanced integration middleware provides this common framework to enable the right mix of underlying capabilities.

5. ***Adaptivity requirements***. Many of the advanced capabilities in next-generation information environments will require adaptive behavior to meet user expectations and smooth the imbalances between demands and changing environments. Adaptive behavior can be enabled through the appropriate organization and interoperation of the capabilities of the four research areas described above. There are two fundamental types of adaptation required: (1) changes beneath the applications to continue to meet the required service levels despite changes in resource availability and (2) changes at the application level to either react to currently available levels of service or request new ones under changed circumstances. In both instances, the system must determine if it needs to (or can) reallocate resources or change strategies to achieve the desired QoS. Applications need to be built in such a way that they can change their QoS demands as the conditions under which they operate change. Mechanisms for reconfiguration need to be put into place to implement new levels of QoS as required, mindful of both the individual and the aggregate points of view, and the conflicts that they may represent.

   Part of the effort required to achieve these goals involves continuously gathering and instantaneously analyzing pertinent resource information collected as mentioned above. A complementary part is providing the algorithms and control mechanisms needed to deal with rapidly changing demands and resource availability profiles and configuring these mechanisms with varying service strategies and policies tuned for different environments. Control theoretic techniques, in particular hybrid systems modeling, have gained significant importance for controlling the behavior of large-scale complex physical systems. Many of these techniques can be encapsulated as middleware- or application-level controllers that can provide verifiable QoS adaptation. Ideally, such changes can be dynamic and flexible in handling a wide range of conditions, occur intelligently in an automated manner, and can handle complex issues arising from composition of adaptable components. Coordinating the tools and methodologies for these capabilities into an effective adaptive middleware for DRE systems should be a high R&D priority.

6. ***System engineering methodologies and tools***. Advanced middleware by itself will not deliver the capabilities envisioned for next-generation embedded environments. We must also advance the state of the system engineering discipline and tools that come with these advanced environments used to build large-scale DRE systems. This area of research specifically addresses the immediate need for system engineering approaches and tools to augment advanced middleware solutions. These include:
   - *View-oriented or aspect-oriented programming techniques*, to support the isolation (for specialization and focus) and the composition (to mesh the isolates into a whole) of different projections or views of the properties the system must have. The ability to isolate, and subsequently integrate, the implementation of different, interacting features will be needed to support adapting to changing requirements.
   - *Design time tools and models*, to assist system developers in understanding their designs, in an effort to avoid costly changes after systems are already in place (this is partially obviated by the late binding for some QoS decisions referenced earlier [1]).
   - *Generative tools*, which use higher level models to synthesize (1) middleware-specific glue code, (2) customized middleware that satisfies QoS and resource constraints, and (3) controllers for QoS adaptation.
   - *Interactive tuning tools*, to overcome the challenges associated with the need for individual pieces of the system to work together in a seamless manner [2]
   - *Composability tools*, to analyze resulting QoS from combining two or more individual components
   - *Modeling tools for developing system performance models* as adjunct means (both online and offline) to

monitor and understand resource management, in order to reduce the costs associated with trial and error
- *Debugging tools*, to address inevitable problems.

7. ***Reliability, trust, validation, and certifiability***. The dynamically changing behaviors we envision for next-generation large-scale, DRE systems are quite different from what we currently build, use, and have gained some degrees of confidence in. Before they can be deployed, considerable effort must therefore be focused on validating, and certifying for operational safety, the correct functioning of the adaptive behavior, and on understanding the properties of large-scale systems that try to change their behavior according to their own assessment of current conditions, before they can be deployed. But even before that, longstanding issues of adequate reliability and trust factored into our methodologies and designs using off-the-shelf components have not reached full maturity and common usage, and must therefore continue to improve.

   Conventional strategies organized around anticipation of long life cycles with minimal change and exhaustive test case analyses are clearly inadequate for next-generation dynamic DRE systems with stringent QoS requirements. An integrated approach involving model-based tools and middleware can therefore help alleviate stringent certifiability and reliability requirements. For example, model-based formalisms are easily amenable to verification for system correctness and empirical benchmark generation for verifying the configured middleware properties.

Concluding Remarks

Over the past decade, middleware has emerged as a set of reusable software layers that help resolve problems associated with heterogeneity and interoperability. It has also contributed considerably to better environments for building distributed real-time and embedded (DRE) systems and managing their distributed resources effectively. A major trend driving researchers and practitioners involves moving toward a multi-layered architecture (i.e., applications, middleware, network and operating system infrastructure) that is oriented around application composition from reusable components, and away from traditional architectures where DRE applications were developed directly atop the network and operating system abstractions. This middleware-centric, multi-layered architecture descends directly from the adoption of a network-centric viewpoint brought about by the emergence of the Internet and the componentization and commoditization of hardware and software.

This article has presented a research agenda to meet the R&D challenges associated with developing middleware, aided by modeling, analysis, and synthesis tools, to support DRE systems. Since challenges facing the embedded systems community change constantly, in terms of resources and expectations, we do not have the luxury of being able to design DRE systems to perform highly specific functions and then expect them to have life cycles of 20 years with minimal change. In fact, we more routinely expect DRE systems to behave differently under different conditions, and complain when they just as routinely do not. These changes have raised a number of issues, such as end-to-end oriented adaptive QoS, and construction of DRE systems by composing off-the-shelf parts that have promising solutions involving significant new middleware-based capabilities and services.

The ultimate goals of the research agenda described in this article are to (1) reliably and repeatably construct and compose DRE systems that can meet and adapt to more diverse, changing requirements/environments and (2) enable the affordable construction and composition of the large numbers of these systems that society will demand, each precisely tailored to specific domains. To accomplish these goals, we must overcome not only the technical challenges, but also the educational and transitional challenges, and eventually master and simplify the immense complexity associated with large-scale DRE system environments, as we integrate an ever growing number of hardware and software components together via middleware.

Acknowledgements

The middleware research agenda presented in this paper is based in part on discussions that took place in conjunction with the Software Design and Productivity Coordinating Group Workshop on New Visions for Software Design and Productivity: Research and Applications, December 13-14, 2002 Nashville, TN. Participants in this workshop included Betty Cheng (Michigan State University), Joe Cross (DARPA), Gary Daugherty (Rockwell Collins), Prem Devanbu (UC Davis), Cordell Green (Kestrel), Thuc Hoang (DOE), Sally Howe (NCO), Jim Hugunin (PARC), Kane Kim (UC Irvine), Joe Loyall (BBN Technologies), Mike Masters (NSWC), Priya Narasimhan (CMU), Martin Rinard (MIT), Richard Schantz (BBN Technologies), Douglas Schmidt (Vanderbilt University), and David Sharp (Boeing).