IEEE distributed systems ONLINE
Expert-authored articles and resources

IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922 © 2004 Published by the IEEE Computer Society, Vol. 5, No. 7; July 2004

Adaptive Resource Management in Middleware: A Survey

Hector A. Duran-Limon, **Tecnológico de Monterrey**
Gordon S. Blair, **Lancaster University**
Geoff Coulson, **Lancaster University**

Current middleware technologies cannot meet the demands of new application areas, such as embedded and mobile systems, that require mechanisms for dealing with a changing environment. This article reviews several approaches for providing adaptive resource management for middleware.

Current middleware technologies, such as the Common Object Request Broker Architecture (CORBA)[1] and .NET (http://msdn.microsoft.com/net), mask system and network heterogeneity problems and alleviate the inherent complexity of distributed systems in many application areas. However, the recent emergence of new application areas for middleware, such as embedded systems, real-time systems, and multimedia, imposes challenges that few existing middleware platforms can meet. In particular, because they impose greater resource-sharing and dynamism demands, these application areas require more complex and sophisticated middleware. Resource sharing must be controlled and predictable to ensure that activities running on the same middleware instance have adequate resources.

Dynamism is most obvious in mobile computing systems, which are inherently dynamic. Consider, for instance, a mobile computer that's initially connected to a fixed network. At this point, the machine has plenty of network resources. However, when the user moves it to another place and replaces the network connection with a wireless one, the mobile computer experiences a lack of network bandwidth. Furthermore, many applications (distributed multimedia, for example) are inherently dynamic—for instance, the number of participants in a videoconference system can change at any time. So, the application's resource requirements fluctuate over time. Other application areas and types have dynamism requirements as well. Even embedded real-time applications can benefit from middleware support for dynamism when just-in-time component loading and automated system evolution are desirable goals.

Adaptive resource management helps systems meet these requirements, especially when unexpected perturbations lead to resource scarcity. Dynamically redistributing resources within application activities can address such problems—that is, systems should be able to change resource reservations over time on a per-activity basis. This approach is only possible, however, when the system knows both the resource requirements and the resource availability. Selecting the most appropriate replacement for a component requires that the system also know the resource demands of the components running the application. Such replacements usually involve a trade-off between resource types. For instance, replacing a Global System for Mobile Communications (GSM)[2] compressor with a mixed-excitation linear predictive (MELP) coder[3] compressor trades network bandwidth for CPU demand.

Some researchers have introduced adaptive resource management support in middleware platforms and as a result have produced approaches offering facilities for configuring and reconfiguring resources. This article reviews the most important work in adaptive resource management in middleware. Our goal is to help practitioners in general and next-generation middleware architects in particular evaluate the strengths, limitations, and drawbacks of current and new approaches and to define some guidelines for future efforts. Relevant research not covered in our review include Realize,[4] Darwin,[5] Agilos,[6] Globus,[7] and FlexiNet.[8]

RESOURCE MANAGEMENT IN MIDDLEWARE

Resource management has two purposes:

- Enhance system performance by maximizing use of system resources, such as CPU, memory, and network bandwidth
- Distribute and allocate system resources according to application resource requirements

Developers must sometimes choose one goal at the other's expense. For instance, *hard real-time applications* (critical applications that can't tolerate missed deadlines) usually overbook resources to support worst-case execution times.

Resource management also plays an important role in the adaptation process in terms of both resource awareness and dynamic resource reallocation. That is, a middleware platform can provide facilities that inform a system of computational resource availability, current management policies, and resource allocation among the system's activities. Such platforms also provide facilities to dynamically reconfigure allocated resources and replace management policies when perceiving environmental changes. For example, a platform could redistribute CPU time and memory to meet the needs of the system's activities.

This article assumes that the most adequate and natural locus for applying adaptation is at the middleware level. Adaptation at the operating-system level is platform-dependent and requires deep knowledge of the operating system's internals. In addition, unwanted changes at this level could be catastrophic because they might affect every application running in a node. At the other extreme, some research in operating systems[9] and networking[10] has advocated leaving as much flexibility and functionality to applications as possible to satisfy their many requirements. Application-level adaptation imposes an extra burden on the application developer, however. In addition, because they're application-specific, adaptation mechanisms developed at this level can't be reused.

Two aspects—openness and ease of use—are essential to achieving adaptive resource management in middleware. Although other aspects are also important, a system is adaptive and practical if it meets at least these two aspects. For example, consistency support ensures that the system is always in a valid state; however, consistency check mechanisms aren't essential if developers take care to maintain the system in a safe state.

Openness and flexibility

Current middleware follows the traditional software engineering approach—that is, it hides the implementation details from the user. Encapsulating the implementation details results in a "black box" that's difficult to inspect and modify. To achieve the resource management adaptation that middleware requires, developers must introduce open resource configuration and reconfiguration.

Resource management in middleware should adapt to the deployment platform's specific requirements. Therefore, both resource configurations (that is, resource allocation) and resource management policies should be configurable. For example, in embedded systems, devices such as PDAs and sensors, which have limited battery life, need power management to use energy efficiently. Because memory and CPU resources are also limited, the resource system should conform to the constraints imposed by the deployment platform. Resource configuration usually occurs at compile or load time. In either case, it inevitably involves the use of a description language (or languages) to specify the configuration.

Resource management in middleware should also dynamically adapt to resource availability and other contextual changes. Such resource management systems should support both *runtime reallocation of system resources* (also termed *runtime resource reconfiguration*)—that is, redistribution of the resources used by a set of executing tasks—and dynamic changes to the resource management policies. A mobile application is one type of highly dynamic system requiring resource adaptation. Communication delays between nodes can vary unexpectedly as the number of hops to reach the destination changes; conditions in the geographical area can cause random periods of disconnection; and an area can unexpectedly become congested, resulting in a lack of communication resources.

Openness should be flexible. Flexibility relates to the magnitude of the changes that are allowed. A system is flexible if it supports both coarse- and fine-grained resource management adaptation, which it achieves by representing resources at multiple abstraction levels. Coarse-grained resource management occurs among several applications and can involve node clusters. The middleware inspects an application's resources and, if other, more important applications require additional resources, lowers its allocation on their behalf. Fine-grained resource management involves inspecting and reconfiguring the resources used by a single application's sessions (that is, stream connections). Resource abstractions involved in this case include thread and memory pools.

Finally, resource management can also occur at the operation (that is, function) level, whereby the system manages lower-level resource abstractions, such as a single thread and chunks of memory buffer.

Ease of use

Performing adaptive resource management of applications involving multiple resource types can be difficult. Managing the resources of large-scale applications can introduce further complexity. For example, distributed multimedia usually involves a number of heterogeneous resources, including network bandwidth, CPU cycles, and memory buffers and storage. Resource management in environments with heterogeneous resources typically requires abstract resource models to alleviate the complexity of coordinating and adapting diverse resources. Such models should offer uniform resource abstractions, which enable consistent resource management over various resource types. In addition, uniform resource abstractions let systems evolve easily because they can incorporate new resource types and resource management policies as they become available.

Modeling resource management for single client-server interactions isn't difficult for small-scale applications. For large-scale applications involving hundreds or thousands of interactions, however, code complexity increases considerably. Consider, for example, an automatic car control system that lets hundreds of cars cooperate to avoid collisions and traffic congestions, or a video-on-demand system that allows thousands of simultaneous sessions. Hence, modeling resource management of more than one interaction—from a single operation invocation to all the interactions in an application or group of applications—also requires abstractions.

MIDDLEWARE APPROACHES

Standardization efforts are ongoing in several areas of middleware.

- *Remote procedure calls* extend traditional procedure calls to a distributed paradigm, allowing procedures in heterogeneous distributed platforms to be called as if they were local. The Open Group incorporated its RPC standard into the Distributed Computing Environment standard. Most Unix-based operating systems and many non-Unix systems support DCE's basic core of services.

- *Transaction-oriented middleware* aims to interconnect heterogeneous database systems. It supports the processing of distributed transactions across platforms, offers high performance and availability, and ensures data integrity.

- *Message-oriented middleware* provides asynchronous rather than synchronous interactions. That is, the client need not be synchronized with the server because the client deposits messages in a message container, such as a queue, which the server can collect at any time.

- *Object-oriented middleware* supports the remote invocation of object methods. CORBA, Java remote method invocation (Java RMI, http://java.sun.com/j2se/1.3/docs/guide/rmi/index.html), the Distributed Component Object Model (DCOM, www.microsoft.com/com/dcom.asp), and .NET are important OO middleware platforms.

- *Component-oriented middleware* is a recent evolution of OO middleware. It allows for the large reuse of software and introduces more reconfigurable capabilities into distributed applications. That is, component-based technology lets developers dynamically insert and replace pieces of binary code. Examples of these technologies include Enterprise JavaBeans (EJB, http://java.sun.com/products/ejb) and the CORBA Component Model (CCM).[11]

Often a middleware platform involves more than one middleware type. For instance, the Java Message Service (JMS) specification (http://java.sun.com/products/jms/docs.html) combines the message and object paradigms. Another example, the CORBA Notification Service,[12] is an OO platform featuring event communication services.

Object- and component-oriented middleware are fundamental to resource management. The encapsulation and abstraction properties of the object and component approaches are attractive for modeling and managing middleware resources.[13] Transaction-oriented middleware is suitable for managing storage and communication resources, but it offers little resource-processing support. Message-oriented middleware deals well with communication resources but it achieves limited integration with processing and storage resources. In addition, encapsulating storage resources with RPC-oriented middleware is difficult. Object- and component-oriented middleware, on the other hand, provide a natural way for cleanly encapsulating and integrating these resources. For instance, both approaches can easily hide the complexity of resource management algorithms.

Crucially, the flexibility criterion demands resource representation at multiple abstraction levels. In addition, the ease-of-use criterion requires the use of abstract resource models. The object and component paradigms are clearly best suited for representing these abstractions.

Our review of approaches to adaptive resource management at the middleware level uses openness, flexibility, and ease of use as evaluation criteria.

Real-time CORBA and Dynamic Scheduling

Real-time CORBA (RT-CORBA)[14] offers facilities for end-to-end predictability of operations in fixed-priority CORBA applications. The standard provides support for processing, communication, and memory resources. It represents processing resources as *thread pools* (that is, group of threads). Users configure communication resources by selecting an object request broker (ORB)/transport protocol and establish connections in advance using the explicit binding mechanism. Memory resources are configured within thread pools. Users specify these resources through CORBA Interface Definition Language (IDL) interfaces.

The standard presents a platform-independent priority scheme and defines a priority propagation mechanism that propagates thread priorities across computing nodes. RT-CORBA supports priority transforms, which it implements as "hooks" that intercept requests and can change invocation priority.

The OMG's Dynamic Scheduling adopted standard[15] aims to overcome RT-CORBA's limitations in terms of dynamic scheduling. Dynamic Scheduling CORBA extends the concept of *activity* (an analysis or design concept describing a sequence of control flow that can traverse system boundaries), introduced in RT-CORBA, to an implementation entity, or *distributable thread*. A distributable thread carries its scheduling parameters across system boundaries and can encompass one or more threads. A distributable thread can be partitioned into a (potentially nested) set of *scheduling segments*, each representing a control flow associated with particular scheduling parameters that can span node boundaries. Scheduling segments can be useful, for example, when independently developed software components define their own scheduling segments.

Openness and flexibility. Priority transforms let the system change priorities during resource reconfiguration to accommodate external factors such as server load. The Dynamic Scheduling CORBA standard lets the system modify scheduling parameters during operation, giving dynamic applications more flexibility to adapt. In addition, Dynamic Scheduling provides language support (IDL definitions) for configuring processing, communication, and memory resources. Because it has no resource model for representing resources at multiple abstraction levels, its ability to support resource configuration and reconfiguration is limited.

Ease of use. RT-CORBA's distributable threads handle the complexity of large-scale applications. A distributable thread represents the processing resources of a coarse-grained distributed interaction. Expanding these threads to encompass communication and memory resources as well would improve the approach. In addition, this approach does not support the uniform representation of heterogeneous resources.

UML scheduling profile

The OMG bases its standard for a Unified Modeling Language scheduling profile[16] on a generic framework for modeling resources.[17]

In the UML world, a *profile* is the specialization of UML's general semantics for a particular domain. Within this resource framework, a *resource* is a generic abstraction denoting physical (processors, memory, networks, and so on) or logical (such as buffers, queues, and semaphores) devices. The framework views a resource as a server with associated quality-of-service attributes that attends client demands. A *QoS contract* captures both the client's QoS requirements and the QoS offered by the server.

The UML scheduling profile provides a layered interpretation of the relationship between clients and resources. In a layered interpretation, a two-viewpoint model represents the distributed system. The client (application) side of the relationship relates to the logical viewpoint, whereas the resource (platform) side is the engineering viewpoint. The logical viewpoint represents the logical interactions of the distributed-system entities. It abstracts away how the system implements its entities as well as the interaction mechanism details. The engineering viewpoint then establishes how the system realizes the logical viewpoint. More concretely, this viewpoint describes how a specific technology implements the logical-viewpoint elements. A *realization relationship* is the relationship between the two models whereby elements of the logical model are mapped to elements of the engineering model. Recursively applying the realization relationship can provide different levels of abstraction. At one level of abstraction, the engineering model can be viewed as a logical model and mapped to its own engineering model. The recursion finishes when it reaches hardware resources.

Openness and flexibility. The logical model is related to the Reference Model for Open Distributed Processing (RM-ODP)[18] computational viewpoint, which provides representation for a system's elements along with their logical interactions. In addition, the layered interpretation engineering model, which realizes the logical model, resembles both the RM-ODP engineering and technology viewpoints. Because we can apply the models recursively, they can represent various levels of abstraction. The UML scheduling profile also fully supports resource configuration at multiple abstraction levels. However, the standard focuses more on support for modeling real-time systems' QoS requirements and does not address resource reconfiguration issues directly.

Ease of use. The approach supports the separation of concerns, thus simplifying complex design problems by separating logical aspects from implementation concerns. Furthermore, because the logical and engineering models are detached, the logical model can be realized on several different platform configurations without modification. Hence, the standard provides tools to address the complexity of large-scale systems and presents a comprehensive resource model for representing heterogeneous resources uniformly.

Real-Time Adaptive Resource Management

Honeywell's Real-Time Adaptive Resource Management platform[19] provides middleware mechanisms for QoS negotiation and adaptation. In the RTARM resource model, *service managers*, representing resource management components, manage both specific resources and computing nodes. As Figure 1 shows, SMs are in recursive hierarchy, with higher-level SMs constructed on top of lower-level SMs. At the top level, SMs coordinate end-to-end resource negotiation and adaptation. At the lowest level, SMs model individual resources, such as CPU and network resources, within a node. Interestingly, lower-level SMs provide an adapter mechanism, which supports the incorporation of current and future components implementing scheduling algorithms or protocols. A plug-and-play feature lets RTARM dynamically load SM components, allowing users to implement a different SM by replacing the set of components that realizes a particular SM component.

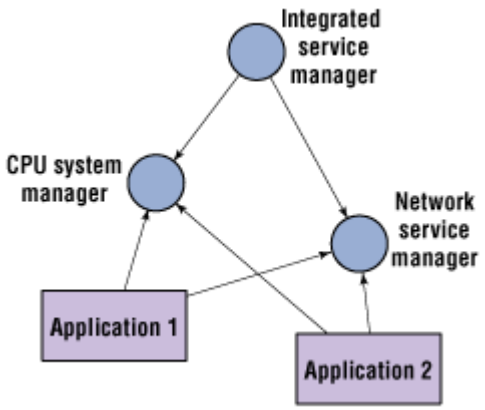

Figure 1. Hierarchy of system managers (SMs) in the Real-Time Adaptive Resource Management (RTARM) platform.

A *distributed session* is the unit of resource negotiation, allocation, and scheduling. The system creates a distributed session for each application program. *Subsessions* are sessions running on individual computing nodes that encompass the set of resources, such as threads and buffer, used for execution. A *task* is a unit of resource management for a given resource agent. For instance, a CPU task represents the list of threads belonging to a subsession.

Finally, the resource framework provides the *ripple scheduling algorithm*, consisting of two major elements:

- A *scheduling spanning tree* describes the execution path across multiple nodes of an application.
- A *two-phase negotiation and adaptation protocol* performs an admission control test on the associated spanning tree and propagates either a commit or an abort command along the tree.

The protocol also shrinks the executing sessions' QoS if the amount of resources available to attend the demands of all sessions is insufficient. After QoS shrinking, it preempts low-criticality sessions if sufficient resources aren't available. At the end of the negotiation phase, the protocol expands the executing sessions' QoS to maximize the application QoS.

Openness and flexibility. The ripple scheduling algorithm fully supports resource reconfiguration. RTARM offers both coarse- and fine-grained adaptation. The distributed session—the unit of resource management encompassing all the resources required for executing an application—supports coarse-grained adaptation. In case of resource contention, for instance, the system might suspend an entire application on behalf of a higher-priority one. Low-level SMs can achieve fine-grained reconfiguration. Although the framework provides mechanisms for resource reconfiguration, its support for static resource configuration seems limited.

Ease of use. A main strength of the RTARM framework is its generic resource model, in which SMs model different resource management components uniformly. Hence, it fully supports the management of heterogeneous resources. Using distributed sessions tackles the complexity of large-scale applications.

The ERDOS QoS architecture

The ERDOS (end-to-end resource management of distributed systems) project[20] offers a generic and comprehensive resource framework providing a middleware architecture with QoS-driven resource management capabilities. The framework offers facilities for admission control, negotiation, and graceful degradation.

Three models—resource, system, and application—describe a distributed system as a graph whose nodes are subsystems or resources and whose edges are connections. A subsystem then includes either a set of resources or a set of subsystems

governed by a single resource management scheme.

The *resource* model provides a uniform abstraction of the resources system. The *system* model defines a hierarchical structure in which the resources are at the bottom layer. The *application* model captures application information such as the application component graph and its associated QoS properties. The application component graph is reconfigurable to allow for the dynamic replacement of components. The application model also lets users recursively encapsulate object interactions. ERDOS defines an extended CORBA IDL to capture this model.

Openness and flexibility. In the ERDOS framework, systems reconfigure resources by performing QoS degradation and modifying the application graph structure. A hierarchical resource model allows both coarse- and fine-grained adaptation. Users can configure resources using an extended CORBA IDL; however, the extensions only partially capture the resource framework.

Ease of use. In large-scale applications, the application model lets users recursively encapsulate object interactions. A service's granularity can range from that provided by a single component to that of the entire application. Finally, users can model multiple types of resources uniformly; thus, the approach fully supports the representation of multiple resource types.

QuO and TAO projects

A joint effort between the Quality Objects (QuO)[21] and ACE Orb (TAO) projects[22] aims to study adaptive middleware for real-time systems.[23] QuO provides a framework for specifying the QoS of CORBA object interactions; TAO is an RT-CORBA implementation focusing on optimization.

One approach integrates the middleware with resource management at the operating system and network levels. More specifically, it layers QuO on top of TAO and incorporates integrated-service (IntServ) mechanisms[24] and the differentiated-service (DiffServ) architecture.[25] TAO provides a mechanism that maps RT-CORBA priorities[14] to DiffServ network priorities. QuO's adaptive mechanisms then change these priorities dynamically according to network traffic conditions. In addition, the approach reserves CPU resources using the TimeSys Linux operating system,[26] which supports CPU process reserves based on parameters such as period and computation time.

QuO provides the QoS Description Language (QDL) for specifying QoS aspects. QDL's designers followed aspect-oriented programming techniques,[27] decomposing programs into different aspects of concern. Developers use a different language to program each of these aspects separately: a *contract description language* to specify QoS contracts, an *adaptation specification language*[28] to specify delegates' adaptive behavior, and a *connector setup language* to define how the QuO objects are associated with the client and object.

Openness and flexibility. QuO's adaptive mechanisms let users introduce dynamic changes according to monitored conditions. However, QuO offers no resource model for uniformly representing resources at different levels of abstraction, as do some of the approaches presented earlier. Thus, its flexibility for resource management is limited. Because the approach is RT-CORBA–compliant, it provides some support for both resource configuration and resource reconfiguration.

Ease of use. QDL removes some of the complexity of programming QoS. However, the approach provides no resource description language. In addition, the QuO/TAO approach focuses on single client-server interactions and doesn't directly support resource management for a large number of interactions. However, as an RT-CORBA–compliant ORB, it does provide support for dealing with large-scale applications. Finally, it gives no support for the treatment of heterogeneous resources.

DynamicTAO

DynamicTAO[29] is a CORBA-compliant reflective ORB supporting runtime distributed reconfiguration. Its implementation is a more flexible extension of the TAO ORB. A system can use reflection to inspect and change its internals in a principled way[30]; thus, a reflective system performs both self-inspection and self-adaptation. To accomplish this, the system has a representation

of itself that is causally connected to its domain—that is, any change in the domain must affect the system, and vice versa.

Figure 2 shows the dynamicTAO architecture. DynamicTAO specifies component dependencies and component resource requirements (hardware and software) in a simple prerequisite description format. Examples of hardware requirements are machine type, operating system, percentage of CPU time, and minimal RAM. Software requirements can include a file system, a window manager, and a Java virtual machine. The 2K resource management service[31] offers a hierarchy of resource managers, with global resource managers at the top and local resource managers at the bottom. Local resource managers are present in each node of the distributed system. Global resource managers constitute a cluster of either local or global managers.

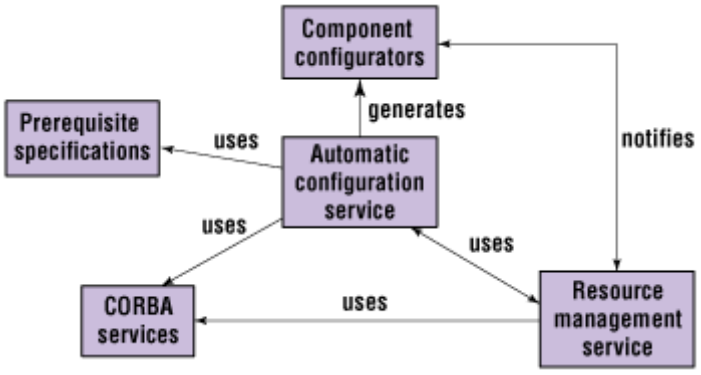

Figure 2. DynamicTAO architectural framework.

A collection of *component configurators* performs dynamic customization. These configurators maintain information about the dependencies between the components they manage. The *DomainConfigurator* holds references to ORB instances and to servants running within an address space. In addition, *TAOConfigurators* attach and detach components and implement strategies such as scheduling, concurrency, security, and monitoring. Components are implemented as dynamically loadable libraries that can be linked to the system at runtime.

Openness and flexibility. The finest granularity for a resource manager is constrained to a node-wide scope, hence limiting the possibilities for finer-grained resource reconfiguration. For instance, users can inspect a given node's CPU availability. Nevertheless, dynamicTAO provides no specific facilities for reconfiguring the resource usage of a single application or a particular application task running on a single node. Moreover, the approach's main focus is application adaptation, such as changing a video application's frame rate and dynamic customization of the ORB (core ORB component replacements) rather than resource adaptation. Although the prerequisite description format supports resource configuration, only coarse-grained configuration is feasible because a node is the finest granularity for a resource abstraction. So, partial support is offered for both configuration and reconfiguration of resources.

Ease of use. Resource managers partially support the representation of heterogeneous resources. The resources modeled by a resource manager are limited to clusters of resource managers (that is, collections of resource managers managed by higher-level resource managers) and a node's hardware resources. For instance, resource managers don't support separate modeling of communication resources. Although using hierarchical managers to cross node boundaries helps diminish the complexity of managing large-scale applications, dynamicTAO presents no abstractions for managing resources on a per-application or -session basis (as supported by RTARM and ERDOS). In this respect, it provides partial support.

Open ORB

The Open ORB framework[32] is a componentized reflective middleware that includes task and resource models.[33] The task model allows high-level analysis and design of the resources subsystem, which we can use to model resource management of both coarse- and fine-grained interactions. Higher-level tasks can represent an application or group of applications. At the task

hierarchy's lowest level, a task can represent the activity of a single object operation.

A close relationship exists between the task and resource models. Tasks have an associated pool of resources defined by the resource model. More specifically, the resource model lets us model different resource types at multiple levels of abstraction. An object running one task can invoke another object concerned with a different task. Such a method invocation represents a *task switching point*. Thus, a task switching point corresponds to a change in the underlying resource pool to support execution of the task in play. Figure 3 shows the various levels of resource abstraction in which the user constructs higher-level resources on top of lower-level resources. *Virtual task machines* are top-level resource abstractions and can encompass several resource types (such as CPU, memory, and network resources) allocated to a particular task.

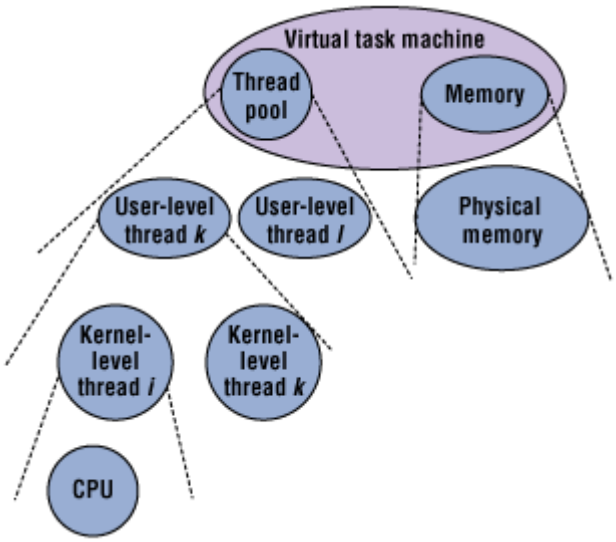

Figure 3. A resource hierarchy in Open ORB.

The resource manager hierarchy complements the resource hierarchy. Open ORB includes operations to traverse both hierarchies: getHL() retrieves the entity located at a higher abstraction level whereas getLL() provides the entity located at a lower abstraction level. The user can dynamically change a processing resource's scheduling parameters at any abstraction level and can modify the amount of resources contained by a passive resource. For instance, it can increment the amount of memory buffer in an abstract resource. We can dynamically replace a resource manager's resource management policy at any abstraction level. In case of resource contention, the user can suspend less important tasks on behalf of more important tasks.

Open ORB uses Xelha,[34] an architecture description language[35] (a formal notation for describing software architectures in terms of coarse-grained components and connectors), and a *resource configuration description language* (RCDL)[34] to specify configuration and reconfiguration of the resources system. Xelha is concerned with the high-level design of QoS management issues. RCDL, in contrast, is a set of aspect languages supporting the low-level specification of the system resources. (Aspect-oriented programming[27] lets us decompose programs into aspects that cross-cut each other.) More specifically, service description language descriptions give information about the QoS level, task, and object class associated with a service. The Task Switch Description Language defines task switching points, whereas the Task Description Language describes the resources assigned to tasks. The Resource Description Language defines the specific resource requirements of a task instance for a particular deployment platform. The QoS Management Graph Description Language describes the QoS management structure.

Openness and flexibility. The Open ORB framework provides language support for high-level QoS management specification and fine-level system resource tuning. It also provides facilities for traversing the resource hierarchy and performing changes at any abstraction level. Because the framework supports resource representation at multiple abstraction levels, coarse- and fine-grained resource management are feasible. It thus fully supports resource configuration and reconfiguration.

Ease of use. By modeling multiple interactions as a single entity, the task model reduces the complexity of large-scale applications. Reflection, however, introduces greater flexibility at the expense of some complexity. Because fully reflective systems are usually difficult to implement, Open ORB achieves partial support in this respect. On the other hand, the framework fully supports the uniform representation of different resource types.

Table 1 summarizes our evaluation of the reviewed approaches. (F indicates that the approach provides full support, P that the approach offers partial support, and X that the approach gives little or no support.) None of the revised approaches provides a complete solution for all evaluation points. However, RTARM, ERDOS, and Open ORB have the most comprehensive resource frameworks, offering full support in three and partial support in one of the evaluated aspects. CORBA, the main middleware standard, still requires further support for adaptive resource management. That is, it offers only partial support for resource configuration and reconfiguration. In addition, it gives no support for the uniform representation of multiple resource types at different abstraction levels.

Table 1. Evaluation of resource management in middleware platforms. (F indicates full support; P, partial support; and X, little or no support.)

Middleware platform		Real-Time CORBA and Dynamic Scheduling	UML scheduling profile	Real-Time Adaptive Resource Management	ERDOS	QuO and TAO	Dynamic TAO	Open ORB
Openness	Configuration	P	F	P	P	P	P	F
	Reconfiguration	P	X	F	F	P	P	F
Ease of use	Support for heterogeneous resources	X	F	F	F	X	P	F
	Support for large-scale applications	F	F	F	F	F	P	P

All the reviewed approaches provide at least partial support for openness and flexibility. Clearly the research community has acknowledged the need for addressing both issues. Half of the approaches fully support either the configuration or reconfiguration of resources; only Open ORB fully supports both aspects. Achieving the desirable level of openness will require further work. Regarding ease of use, nearly half the approaches fully support uniform and consistent representation of heterogeneous resources. Hence, the community has also acknowledged the importance of dealing with different resource types. More efforts are still required in this area, however.

Finally, the evaluation results show that most researchers recognize the need for support for dealing with the complexity of large-scale applications—that is, most of the approaches fully support this aspect.

More openness and flexibility also introduce more complexity. For instance, mechanisms should be provided to maintain a system in a consistent state after reconfigurations are carried out. Further research is needed for managing such a complexity. Self-managing or automatic systems are possible approaches, which would remove the burden from developers and users.

Acknowledgments

We thank the reviewers for their valuable comments, which have notably improved this article's content.

References

1. *Common Object Request Broker: Core Specification, Revision 3.0.3* (CORBA v3.0.3), Object Management Group, 2004; www.omg.org/technology/documents/formal/corba_iiop.htm.

2. A.S. Spanias , "Speech Coding: A Tutorial Review,"*Proc. IEEE,* vol. 82, no. 10, 1994,pp. 1441-1582; http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=326413.

3. A. Spanias , "Speech Coding for Mobile and Multimedia Applications,"*Digital Signal Processing Technologies-Critical Technology Reviews* (CR57), P. Papamichalis and R. Kerwin, eds., SPIE Press, 1995,pp. 115–144.

4. V. Kalogeraki , P.M. Melliar-Smith, and L.E. Moser , "Dynamic Scheduling for Soft Real-Time Distributed Object Systems,"*Proc. 3rd IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing* (ISORC 2K), IEEE CS Press, 2000,pp. 114–121; http://csdl.computer.org/comp/proceedings/isorc/2000/0607/00/06070114abs.htm.

5. P. Chandra , et al., "Darwin: Customizable Resource Management for Value-Added Network Services,"*IEEE Network,* vol. 15, no. 1, 2001,pp. 22–35.

6. B. Li and K. Nahrstedt , "A Control-Based Middleware Framework for Quality of Service Adaptations,"*IEEE J. Selected Areas in Comm.,* vol. 17, no. 9, 1999, pp. 1632-1650.

7. I. Foster , A. Roy, and V. Sander , "A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation,"*Proc. 8th Int'l Workshop Quality of Service* (IWQOS), IEEE Press, 2000, pp. 181–188.

8. R. Hayton , *FlexiNet Open ORB Framework,* tech. report, APM Ltd., 1997.

9. D. Engler , M. Kaaashoek, and J. O'Toole , "Exokernel: An Operating System Architecture for Application-Level Resource Management,"*Proc. 15th ACM Symp. Operating System Principles,* ACM Press, 1995.

10. S. Floyd , et al., "A Reliable Multicast Framework for Light-Weight Session and Application Level Framing," *Proc. Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm.,* ACM Press, 1995,pp. 342-356.

11. J. Mischkinsky , ed. "CORBA 3.0 New Components Chapters," CCM FTF Draft ptc/99-10-04, Object Management Group, 1999; http://CORBAweb.lifl.fr/OpenCCM/docs/ptc_99-10-04.pdf.

12. *Notification Service Specification, Version 1.0.1,* Object Management Group, 2002; www.omg.org/technology/documents/formal/notification_service.htm.

13. D.E. Bakken , "Middleware,"*Encyclopedia of Distributed Computing,* J. Dasgupta, ed., Kluwer Academic Publishers, 2003.

14. *Real-time CORBA 1.0,* Adopted Specification, ptc/99-06-02, Object Management Group, 1999.

15. *Dynamic Scheduling,* final adopted specification, ptc/01-08-34, Object Management Group, 2001.

16. *UML Profile for Schedulability, Performance, and Time, Version 1.0,* Object Management Group, 2002; www.omg.org/technology/documents/formal/schedulability.htm.

17. B. Selic , "Generic Framework for Modeling Resources with UML,"*Computer,* vol. 33, no. 6, 2000,pp. 64–69.

18. *Reference Model for Open Distributed Processing, Parts 1, 2, 3,* ITU-T Rec. X.901 | ISO/IEC 10746-1,2,3, Int'l Standards Organization/Int'l Electrotechnical Commission, 1995.

19. I. Cardei , et al., "Hierarchical Architecture for Real-Time Adaptive Resource Management,"*Proc. IFIP/ACM Middleware Conf.,* Springer-Verlag, 2000,pp. 415–434.

20. S. Chatterjee , B. Sabata, and M. Brown , "Adaptive QoS Support for Distributed, Java-Based Applications,"*Proc. IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing* (ISORC 99), IEEE CS Press, 1999,pp. 203–212; http://csdl.computer.org/comp/proceedings/isorc/1999/0207/00/02070203abs.htm.

21. P. Pal , et al., "Using QDL to Specify QoS-Aware Distributed (QuO) Application Configuration,"*Proc. 3rd IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing* (ISORC 2K), 2000.

22. D.C. Schmidt , D.L. Levine, and S. Mungee , "The Design of the TAO Real-Time Object Request Broker,"*Computer Comm.,* vol. 21, no. 4, 1998, pp. 294–324.

23. R. Schantz , et al., "Flexible and Adaptive QoS Control for Distributed Real-Time and Embedded Middleware,"*Proc. 4th IFIP/ACM/Usenix Int'l Conf. Distributed Systems Platforms,* LNCS 2672, Springer-Verlag, 2003,pp. 374–393.

24. R. Braden , D. Clark, and S. Shenker , "Integrated Services in the Internet Architecture: An Overview," Internet Eng. Task Force tech. report #RFC 1633, 1994; www.ifla.org/documents/rfcs/rfc1633.txt.

25. M. Carson , et al., "An Architecture for Differentiated Services," Internet Eng. Task Force tech. report RFC 2475,

1998; www.ietf.org/rfc/rfc2475.txt.

26. *TimeSys Linux/RT User's Manual,* TimeSys Corp., 2001.

27. G. Kiczales , et al., "Aspect-Oriented Programming,"*Proc. 11th European Conf. Object-Oriented Programming* (ECOOP 97), Springer-Verlag, 1997,pp. 220-242.

28. R. Schantz , et al., "Packaging Quality of Service Control Behaviors for Reuse,"*Proc. 5th IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing* (ISORC), IEEE CS Press, 2002; http://csdl.computer.org/comp/proceedings/isorc/2002/1558/00/15580375abs.htm.

29. F. Kon , et al., "Monitoring, Security, and Dynamic Configuration with the Dynamic TAO Reflective ORB,"*Proc. IFIP Int'l Conf. Distributed Systems Platforms and Open Distributed Processing* (Middleware 00), Springer-Verlag, 2000.

30. P. Maes , "Concepts and Experiments in Computational Reflection,"*Proc. ACM Conf. Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA 87), ACM Press, 1987.

31. F. Kon , et al., "Dynamic Resource Management and Automatic Configuration of Distributed Component Systems,"*Proc. 6th Usenix Conf. Object-Oriented Technologies and Systems* (COOTS 01), Usenix, 2001.

32. G.S. Blair , et al., "The Design and Implementation of Open ORB version 2," *IEEE Distributed Systems Online,* vol. 2, no. 6, 2001; http://csdl.computer.org/comp/mags/ds/2001/06/o6001abs.htm.

33. H.A. Duran-Limon and G.S. Blair , "Reconfiguration of Resources in Middleware,"*Proc. 7th IEEE Int'l Workshop Object-Oriented Real-Time Dependable Systems* (WORDS 2002), IEEE CS Press, 2002,pp. 219–226; http://csdl.computer.org/comp/proceedings/words/2002/1576/00/15760219abs.htm.

34. H.A. Duran-Limon and G.S. Blair , "QoS Management Specification Support for Multimedia Middleware,"*J. Systems and Software,* vol. 72, no. 1, 2004,pp. 1–23.

35. N. Medvidovic and R.N. Taylor , "A Classification and Comparison Framework for Software Architecture Description Languages,"*IEEE Trans. Software Eng.,* vol. 26, no. 1, 2000,pp. 70–93; http://csdl.computer.org/comp/trans/ts/2000/01/e0070abs.htm.

Hector Duran-Limon is a lecturer in the Department of Computing Science at the Tecnológico de Monterrey. His research interests include resource management in adaptable middleware platforms and the role of reflection on such platforms. He is also interested in the use of ADLs and component-oriented techniques for constructing distributed real-time systems. He has a PhD from Lancaster University. Contact him at the Dept. of Computing Science, Tecnológico de Monterrey, Campus Guadalajara, Mexico; hduran@itesm.mx. Web: http://academia.gda.itesm.mx/~hduran.

Gordon Blair is a professor of distributed systems at Lancaster University, an adjunct professor at the University of Tromsø, and a visiting researcher at the Simula Research Laboratory in Oslo. He has a PhD from Strathclyde University. Contact him at Dept. of Computing, Lancaster Univ., Bailrigg, Lancaster LA1 4YR, UK; gordon@comp.lancs.ac.uk.

Geoff Coulson is a reader in distributed computing at Lancaster University. His research interests include middleware architecture, programmable networks, and network and operating system support for continuous media. He received his PhD in distributed multimedia systems from Lancaster University. He is a member of the ACM and the British Computer Society. Contact him at the Dept. of Computing, Lancaster Univ., Bailrigg, Lancaster LA1 4YR, UK; geoff@comp.lancs.ac.uk.