NALINI VENKATASUBRAMANIAN

Safe 'Composability' of Middleware Services

Look to composable middleware frameworks to ensure safe middleware interactions for ubiquitous computing applications.

In the coming decade, distributed computing will establish itself as the default mode of operation for a wide variety of applications. With the proliferation of mobile devices and ubiquitous computing environments today, applications are beginning to place specialized requirements on the computing and communication infrastructure. Interactive applications, including personal multimedia services accessible through handheld devices, will become commonplace. These applications require continuous service quality despite frequent interruptions in network connectivity and degradation of device power levels. Additional requirements, including secure communication in mobile environments, further complicate system management.

Reflective middleware promises dynamic customizability to satisfy application requirements, including quality-of-service (QoS), security, mobility, and fault tolerance, in the presence of changing system conditions. However, a number of technical challenges complicate deployment of safe plug-and-play middleware environments where protocols and services are installed and uninstalled on the fly to accommodate dynamicity. Distributed applications have varying requirements, often stated as QoS parameters defining the extent of violation of performance specifications, including responsiveness, reliability, availability, cost-effective utilization, and security. These requirements are satisfied through appropriate resource management policies, such as replication, migration, and checkpointing. Satisfying application QoS in a dynamic environment implies the dynamic customization of the underlying management mechanisms to meet QoS requirements.

The safe "composability," or simultaneous operation, of the executing middleware entities is a prime concern. Fundamental problems arise in the concurrent execution of multiple distributed resource management services and protocols; further complications arise in the dynamic customization of these services and protocols. The changing, often complex, interactions among distributed policies and services within the middleware layer can violate correctness, destroy validity, and alter the semantics of applications built above generic distributed management infrastructures. Much of the need to ensure safe interactions between middleware services has been ignored in existing commercial frameworks and distributed middleware infrastructures. Frameworks like the Object Management Group's Common Object Request Broker (CORBA) and Microsoft's Distributed Component Object Model (DCOM) represent a step toward compositional software architectures but do not deal with the interactions of multiple object services executing at the same time or with the implication of having to compose object services.

This article explores principles and practices for developing safe customizable middleware, specifi-

TERRY MIURA

cally for ensuring composable distributed resource management, or the concurrent execution of multiple resource-management policies in distributed systems. The use of these principles and practices will allow the safe integration of mechanisms for such services as mobility, load balancing, fault tolerance, and end-to-end QoS management. The principles have been used in, for example, the Composable Open Software Environment with QoS framework, or CompOSE|Q (see www.ics.uci.edu/~dsm/compose), to manage change in large-scale distributed systems while ensuring application QoS requirements [3].

A Meta-Architectural Framework for Safe Composability

The Two Level Actor Machine (TLAM) model [4] is a first step toward providing a formal semantics for specifying and reasoning about the properties of and interactions among middleware components. A model of concurrent active objects, called Actors [1], provides a uniform representation of concurrent system- and application-level entities distributed across a network. In the TLAM model, meta-level controllers define protocols [2] and the mechanisms customizing various aspects of distributed systems management.

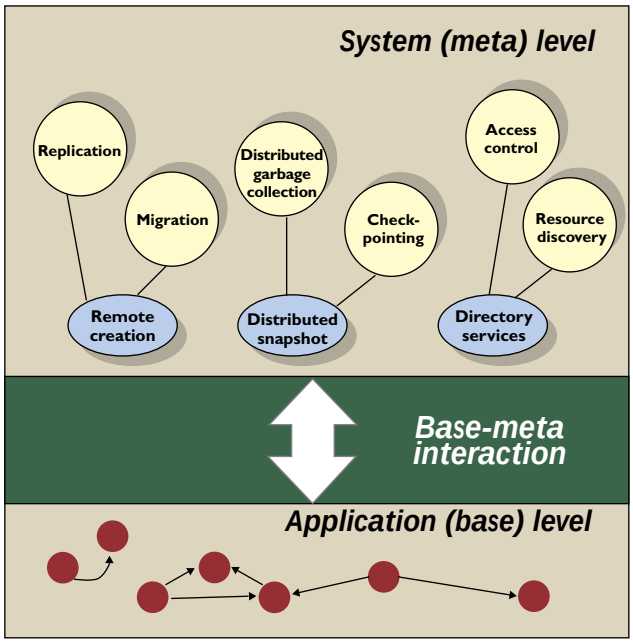

Figure 1. Classification of core services.

In practice, because they occur concurrently in distributed systems, multiple system and application activities (such as scheduling, protocol processing, and stream synchronization) can interfere with each other. Composing multiple resource-management mechanisms leads to complex interactions. Consider the following example of a system in which distributed garbage collection and process migration proceed concurrently. If processes in migration (transit) are not accounted for, the garbage collection process could destroy accessible information. Similarly, if the process migrates continuously, the garbage collection process risks nontermination. In general, risks due to mechanism composition include loss of information, nonterminations causing deadlocks and livelocks, dangling resources, inconsistencies, and incorrect execution semantics.

One approach to dealing with interference during mechanism composition in an open system is to serialize or delay activities to ensure the system's overall safety. However, it can also result in the overserialization of resource-management activities, causing performance degradation. Global delays and halts may cause violations of timing-based QoS constraints. An alternative approach is a closed system in which handcrafted mechanisms are tied to applications. But a handcrafted system results in the loss of openness, flexibility, and modularity and reduces portability, leading to reengineering effort. A system designer might argue that composability of resource management activities is not only desirable but essential for cost-effective QoS in distributed systems. For instance, system protocols and activities should not be able to enforce arbitrary delays in the presence of timing-based QoS constraints (see Figure 1).

To ensure noninterference and manage the complexity of reasoning about components of open distributed systems in general, our strategy is to identify key system services involving nontrivial interactions between the application (base-level objects) and the system (meta-level objects). These core services are then used in specifying and implementing more complex activities within the framework as purely meta-level interactions. Development of suitable noninterference requirements allows us to reason about the composition of multiple system services with constraints that must be obeyed to maintain composability. Note, too, that a formal specification of the core service approach has to include interfaces and interaction requirements (usage constraints) for the core services. We have identified three such services:

Remote creation. The re-creation of services/data at a remote site;

Distributed snapshots. The capture of approximations of state information at multiple nodes/sites; and

Directory services. Interactions with a global repository.

Using these core services, a system designer can develop a library of commonly used services and protocols that are safe and composable. For instance, scheduling mechanisms use the basic remote-creation core service to address two levels of scheduling: (a) object scheduling, which assigns newly created objects on nodes in the interest of load-balancing, locality, fault tolerance, and other criteria; and (b) message scheduling, which addresses the scheduling of messages destined for existing objects. The basic scheduling techniques can be extended to support policies involving priority- and constraint-based scheduling. The remote creation core service can also be used in the implementation of object migration, as well as in the replication of data and services. A checkpointing service can be developed using generalized state-capture facilities to obtain causal orders of executions in the system and can be used for monitoring and debugging distributed computations. A state broadcast mechanism is used to implement a clock synchronization service that informs nodes about a global time value that in turn can be used for time-related services. The directory core service is used to develop other services, including:

- Naming and namespace management techniques for defining resource provisioning and group-based communication;
- Dynamic discovery and location of objects using name-based and attribute-based object discovery; and
- Access control and security mechanisms.

Each is accompanied by specific interface definitions and interaction constraints.

Application Requirements Via Composable Middleware

Meta-level mechanisms can be used to provide application requirements, including QoS, security, mobility, and fault tolerance, in a modular and composable manner.

QoS. The basic meta-architectural framework can be extended to provide QoS-based services to applications. Figure 2 outlines the modular use of a meta-architecture in facilitating application QoS. The application (base)-level component of the meta-architecture implements the functionality of the distributed session and deals with: (a) data, including objects of varying media types, such as video and audio files; and (b) requests to access this data via sessions. The meta-level component deals with the coordination of multiple requests and the sharing of existing resources among multiple requests. To provide coordination at the highest level and perform admission control for new incoming sessions, we define a meta-level entity called the QoS broker. Figure 2 outlines meta-level services obtained through

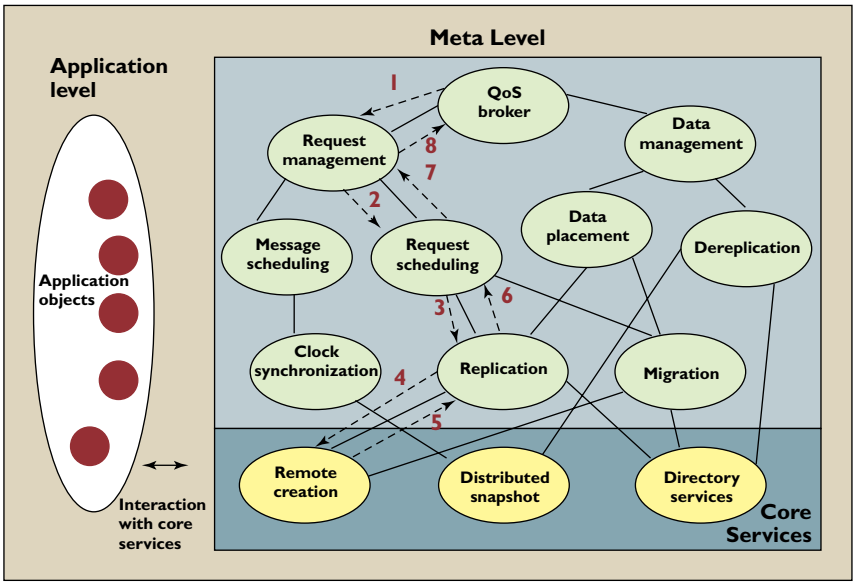

Figure 2. CompOSEIQ system architecture. Dotted lines indicate the flow of an incoming request through the middleware modules.

an analysis of module functionality in commercial multimedia servers. Each of these services can be based on one or more of the core services: remote creation, distributed snapshot, and the directory service. A meta-architecture makes it possible to formally specify and reason about the satisfaction of QoS properties in the presence of other activities.

Security and mobility. Future systems will have to enforce security at multiple levels, from access control for legitimate users, to cryptographic protocols guarding against illegitimate users. Securing the system from illegitimate users requires further investigation of possible security attacks, intrusion-detection policies, key-based encryption algorithms, and a variety of techniques like digital signatures and watermarking. The generalized direc-

tory service can be extended to define access control and security mechanisms that provide global accessibility while ensuring the protection of data from unwarranted use. Such an approach allows flexibility in the degree of coupling between security policies and naming services. Further work is needed to understand the trade-offs between extensibility and security, as the ability to arbitrarily customize a system can result in security breaches. Meta-level mechanisms can also be used for managing mobility and for communicating among the different mobile components in a safe and correct manner. Further work is also required to generalize these mechanisms, embed mobility information into the directory service, and understand the effect of system management activities on migrating components.

FUNDAMENTAL PROBLEMS ARISE IN THE CONCURRENT EXECUTION OF MULTIPLE DISTRIBUTED RESOURCE MANAGEMENT SERVICES AND PROTOCOLS.

Yet another challenge involves developing middleware that allows flexible composition of these requirements, including secure mobility and mobile QoS. For application developers, it is increasingly obvious that security is a critical issue in making full-fledged mobile computing a reality. Understanding complex interactions among the mechanisms facilitating mobile code and the mechanisms implementing security policies requires a modular representation of the services behind secure mobility. Current techniques for secure mobility implement a series of low-level checks; there is a lack of formal models for expressing the interactions among security and mobility components using higher-level abstractions. Similarly, ensuring QoS to users in mobile and ad-hoc communication environments is complicated by frequent handoffs that trigger the rerouting and rescheduling of network and server resources. Composability issues in delivering secure mobility and mobile QoS include time-sensitive reliable message delivery to migrating objects, as well as the delegation and revocation of access rights in the presence of object mobility and failures.

Conclusion

System developers often underestimate the need for formal semantics and formal reasoning methodologies. With the growth in ubiquitous computing environments, middleware frameworks are starting to employ techniques, including computational reflection, to manage resources and services under constantly changing conditions. Although reflection is a powerful tool for customizability, its unconstrained use can cause unexpected behavior by applications and violate desired application semantics; the application designer is often totally unaware of the effects caused by the underlying middleware infrastructure. New software design methodologies are required to enable the translucent design of middleware frameworks where both interfaces and interaction constraints are exposed to system designers using the framework.

The dynamic nature of applications under varying network conditions and traffic requests generally implies resource-management policies should be dynamic and customizable. Composable middleware frameworks implementing cleanly defined meta-architectures enable the customization of applications, protocols, and system services. The result is a foundation for the evolution of large-scale ubiquitous computing. C

References

1. Agha, G. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, Cambridge, MA, 1986.
2. Agha, G., Frølund, S., Panwar, R., and Sturman, D. A linguistic framework for dynamic composition of dependability protocols. In *Proceedings of the IFIP Third Working Conference on Dependable Computing for Critical Applications*, C. Landwehr, B. Randell, and L. Simoncini, Eds. Springer-Verlag, 1993, 345–363.
3. Venkatasubramanian, N., Deshpande, M., Mohapatra, S., Gutierrez-Nolasco, S., and Wickramasuriya, J. Design and implementation of a composable reflective middleware framework. In *Proceedings of the IEEE International Conference on Distributed Computer Systems* (Mesa, AZ, Apr. 16–19). IEEE Computer Society Press, Los Alamitos, CA, 2001, 644–653.
4. Venkatasubramanian, N. and Talcott, C. Meta-architectures for resource management in open distributed systems. In *Proceedings of the ACM Symposium on Principles of Distributed Computing* (Ottawa, Ontario, Canada, Aug. 20–23). ACM Press, New York, 1995, 144–153.

NALINI VENKATASUBRAMANIAN (nalini@ics.uci.edu) is an assistant professor in the Department of Information and Computer Science of the University of California, Irvine.