A Reflective Framework for Providing Safe QoS-enabled Customizable Middleware

Nalini Venkatasubramanian and Carolyn Talcott
Univ. of California Irvine and Stanford University,
Email: nalini@ics.uci.edu, clt@cs.stanford.edu

Open Distributed Systems (ODS) evolve dynamically and components of ODS interact with an environment that is not under their control. Managing change efficiently is critical to the effective deployment of applications executing in such environments; reflection helps deal with the need for flexibility. A wide range of protocols and activities must execute concurrently and non-disruptively, and must share resources. In order to avoid resource conflicts, deadlocks, inconsistencies and incorrect execution semantics, the underlying resource management system must ensure that the concurrent system activities compose in a correct manner. In this position paper, we discuss our experiences in developing system level behaviors in the context of $\mathbf{CompOSE|Q}$, a QoS-enabled composable middleware framework and outline key challenges in the development of formally verifiable middleware for next generation highly distributed and customizable applications.

Ensuring correctness in a purely reflective environment involves reasoning about system level interactions by characterizing the semantics of shared distributed resources, and the understanding of what correctness of the overall system means. We have earlier developed the TLAM [11, 12], a two-level model of distributed computation based on Actors, a model of concurrent objects; the TLAM supports dynamic customizability and separation of concerns in designing and reasoning about components of ODS. In the TLAM, a system is composed of two kinds of actors, base actors and meta actors, distributed over a network of processing nodes. Base level actors carry out application level computation, while meta-actors are part of the runtime system which manages system resources and controls the runtime semantics of the base level. Meta-actors communicate with each other via message passing as do base level actors, but meta-actors may also examine and modify the state of the base actors located on the same node. The model abstracts from the choice of a specific programming language or system architecture, providing a framework for reasoning about heterogeneous systems. The framework has a very natural representation in rewriting logic [4, 6]. A one level framework (called actor theories), restricted to purely base-level systems has been developed and applied to specification and reasoning about actor systems and languages [5, 9].

The two-level architecture naturally extends to multiple levels, with each level manipulating the level below while being protected from manipulation by lower levels. In practice, however, expressing a computation in terms of multiple meta-levels becomes unwieldy. A purely reflective architecture provides an unbounded number of meta-levels with a single basic mechanism. The formal verification of interaction semantics between the different layers in the reflective hierarchy can be quite complex. The challenge here is to develop easy to use principles for harnessing the power of reflection and avoiding the potential chaos that is possible with its unrestricted use.

In other reflective models for distributed object computation [7, 2, 1], an object is represented by multiple models allowing behavior to be described at different levels of abstraction and from different points of view. In each model the behavior of an object is described by a metaspace that consists of meta objects representing the different models/aspects (e.g. encapsulation, environment) and each meta object sees and acts on one base level object. In the TLAM, each meta actor can examine and modify the behavior of a group of base level actors – namely

those located on the same node. While some instances of the TLAM may have the many-to-one organization of the multiple model frameworks, the more flexible relation seems appropriate when considering resource management facilities that involve manipulation of collections of base level actors. Other work on using reflective ORBs to customize resource management behavior, e.g. scheduling is reported in [8].

The TLAM uses reification (base object state as data at the meta object level) and reflection (modification of base object state by meta objects) with support for implicit invocation of meta objects in response to changes of base level state. This provides for debugging, monitoring, and other hooks. For instance, the TLAM framework has been used to formally specify and reason about an event-based logging service for distributed messaging. The TLAM provides for full actor-style interaction of meta level objects; we are currently looking into providing restricted support for the *explicit* invocation of meta objects by base objects. Our earlier work has focused on defining a rigorous mathematical semantics and on using this semantics to develop concepts and methods for expressing and reasoning about properties of ODS and their components. To reason about middleware, we give TLAM specifications of the resource management algorithms at different levels of abstraction. Service level specifications express requirements on global system behavior, while behavior level specifications express requirements on individual actors or collections of actors. The two levels are related by the notion of a group of actors providing a service. Each form of specification itself may be refined in various ways. Such refinements generally correspond to design decisions related to the choice of a class of distributed algorithms for implementing the service.

To manage the complexity of reasoning about multiple middleware components and services, we identify three key *core services* provided by meta-actors where non-trivial base-meta interactions occur - remote creation, distributed snapshots and directory services. Using the core services, we define other metalevel services and specify properties such as functionality, quality-of-service and non-interference requirements in terms of purely meta-level interactions, which are more easily expressed and understood, although still non-trivial. The $\mathbf{CompOSE|Q}$ framework (See Fig 1), currently being developed at the University of California, Irvine implements the basic TLAM metaarchitecture with additional features to support a reflective communication layer for actor interaction. Within $\mathbf{CompOSE|Q}$, we have developed (from formally verifiable specifications) resource management services such as remote creation, migration and reachability snapshot and ensured their safe composition.

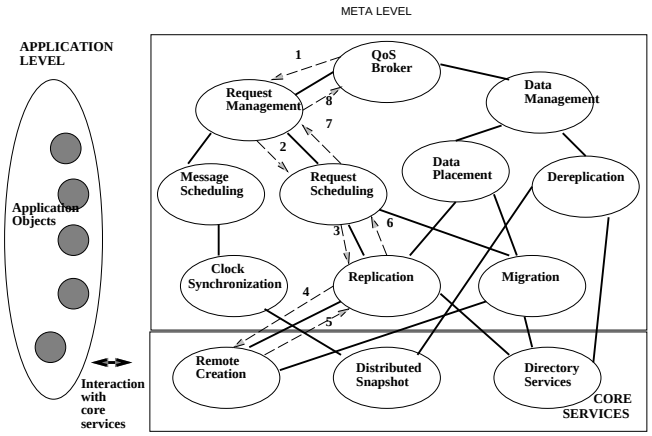

Figure 1: Architecture of the $\mathbf{CompOSE|Q}$ System. The dotted lines indicate the flow of an incoming request through the middleware modules.

The remote creation core service allows a meta-actor to create a base-level subsystem on a remote node. It has been used as the basis for services such as migration and replication in a distributed system. Scheduling mechanisms use the basic remote creation core service for object scheduling(that assigns newly created objects/actors on nodes) and message scheduling (i.e. messages destined for existing actors). We have also defined a reachability snapshot service that provides the capability to record a snapshot of the base-level reachability relation. The reachability snapshot specializes the notion of global snapshot of a distributed computation which is an important tool for distributed programming. For instance, we have earlier developed a distributed garbage collection algorithm for concurrent objects based on the reachability snapshot. Using generalized state capture facilities, we are developing a checkpointing service for capturing causal orders of executions in the system that can be used for monitoring and debugging distributed computations. A state broadcast mechanism is used to implement a

clock synchronization service, which informs nodes about a global time value that can be used for time related services. We are currently exploring the use of a directory core service to develop (1) Naming and namespace management techniques used for defining routing policies and group based communication. (2) Dynamic discovery and location of objects using name based and attribute based object discovery. (3) Access control and security mechanisms.

In order to ensure safe and cost-effective QoS in distributed multimedia environments, composability of resource management services is essential. For instance, system level protocols must not cause arbitrary delays in the presence of timing based QoS constraints. The $\mathbf{CompOSE|Q}$ framework implements QoS-based resource management services for distributed multimedia systems as metalevel services in the reflective architecture. To ensure QoS in highly adaptive environments, metalevel policies must provide efficient implementations of resource discovery protocols used for admission control, resource reservation and scheduling. We apply the QoS broker system (See Figure 1) to implement scheduling and routing of MM requests as well as placement of MM data using a collection of multimedia resource management meta-actors providing these services and show how to reason about their interaction [10]. We are experimenting with reflective mechanisms to provide dynamic resource adaptation while ensuring QoS using the notion of events. We are currently working on using $\mathbf{CompOSE|Q}$ to design reflective MM systems that enable the execution of dynamic QoS negotiation and renegotiation policies. This implies the integration of economic models into the metalevel infrastructure to manipulate cost-quality parameters/tradeoffs on the fly. Effective implementations of such commercial applications will require the incorporation of flexible transactional semantics into the reflective middleware layer that can be used to ensure consistency properties and express weaker transaction models [3].

References

[1] G. Blair, G. Coulson, P.Robin, and M. Papathomas. An architecture for next generation middleware. In *Middleware '98*, 1998.

[2] F. Costa, G. Blair, and G. Coulson. Experiments with reflective middleware. In *European Workshop on Reflective Object-Oriented Programming and Systems, ECOOP'98*. Springer-Verlag, 1998.

[3] A. Elmagarmid. *Database Transaction Models for Advanced Applications*. Morgan-Kaufmann Publishers.

[4] P. Lincoln, N. Martí-Oliet, and J. Meseguer. Specification, transformation, and programming of concurrent systems in rewriting logic. In G. Blelloch, K. Chandy, and S. Jagannathan, editors, *Specification of Parallel Algorithms*, pages 309–339. DIMACS Series, Vol. 18, American Mathematical Society, 1994.

[5] I. A. Mason and C. L. Talcott. Actor languages: Their syntax, semantics, translation, and equivalence. *Theoretical Computer Science*, 228(1), 1999.

[6] J. Meseguer and C. Talcott. A partial order event model for concurrent objects. In *The International Conference on Concurrency Theory (CONCUR '99)*, Lecture Notes in Computer Science. Springer Verlag, 1999.

[7] H. Okamura, Y. Ishikawa, and M. Tokoro. Al-1/d: A distributed programming system with multi-model reflection framework. In A. Yonezawa and B. C. Smith, editors, *Reflection and Meta-Level Architetures*, pages 36–47. ACM SIGPLAN, 1992.

[8] A. Singhai, A. Sane, and R. Campbell. Reflective ORBs: Support for Robust, Time-Critical Distribution. In *Proceedings of the ECOOP'97 Workshop on Reflective Real-Time Object-Oriented Programming and Systems*, June 1997.

[9] S. F. Smith and C. L. Talcott. Modular reasoning for actor specification diagrams. In P. Ciancariani, A. Fantechi, and R. Gorrieri, editors, *Formal Methods for Open Object-based Distributed Systems*, pages 313–330. Kluwer, 1999.

[10] N. Venkatasubramanian. Compose—q - a qos-enabled customizable middleware framework for distributed computing. In *Proceedings of the Middleware Workshop, International Conference on Distributed Computing Systems (ICDCS99)*, June 1999.

[11] N. Venkatasubramanian and C. L. Talcott. Reasoning about Meta Level Activities in Open Distributed Systems. In *14th ACM Symposium on Principles of Distributed Computing*, pages 144–152, 1995.

[12] N. Venkatasubramanian and C. L. Talcott. Integration of resource management activities in distributed systems. Technical report, Stanford University Computer Science Department, 1999.