# CompOSE|Q - A QoS-enabled Customizable Middleware Framework for Distributed Computing

Nalini Venkatasubramanian
Dept. of Information and Computer Science, Univ. of California Irvine,
Email: nalini@ics.uci.edu

## Abstract

*Advances in networking, communication, storage and computing technologies coupled with emerging novel application areas is enabling the widespread use of large scale distributed computing systems. These systems exhibit constant evolution as new applications place specialized requirements from the computing and communication infrastructure. Many applications provide QoS (Quality of Service) parameters that define the extent to which performance specifications such as responsiveness, reliability, availability, security and cost-effectiveness may be violated. These requirements are often implemented via resource management mechanisms in the middleware. In this paper, we develop a Qos-enabled customizable middleware framework called* CompOSE|Q *that can safely and effectively manage change in large scale distributed systems. We illustrate how to achieve flexible, safe and efficient composability of resource management services in the middleware layer while ensuring QoS to the application.*

In the next decade,distributed computing will be the default mode of operation. Large scale distributed systems handle applications that are open and interactive; such systems evolve dynamically and their components interact with an environment that is not under their control. Increasing levels of distribution in information access, openness and dynamism results in increasing complexity. Distributed middleware enables the modular connection of software components to manage the resources of an open distributed system (ODS); it can be used to constrain the global behavior of the distributed system to ensure safety while providing cost-effective utilization of resources. Distributed applications have varying requirements, often stated as QoS (Quality of Service) parameters that define the extent to which performance specifications such as responsiveness, reliability, availability, cost-effective utilization and security may be violated. These application requirements can be satisfied using appropriate resource management policies,e.g. replication, migration, checkpointing. The role of middleware is to abstract over the low level *mechanisms* which are required to implement these policies. Building a QoS-enabled customizable middleware framework requires characterizing and reasoning about the interactions between multiple resource management activities in ODSs, their dynamic installation and customization in the presence of QoS requirements.

In this paper, we describe a framework for QoS-enabled customizable middleware called CompOSE|Q, currently being developed at the University of California, Irvine. CompOSE|Q is based on a metaarchitectural model that facilitates specifying and reasoning about the composability of multiple resource management services in ODSs. The rest of this paper is organized as follows. Section 1 briefly describes the two level metaarchitecture, a formal semantic model for reasoning about middleware. Section 2 discusses the basic CompOSE|Q framework and Section 3 describes how application requirements such as security and mobility can be incorporated into CompOSE|Q. Section 4 addresses the implementation of the CompOSE|Q environment. We discuss related work and future research directions in Section 5.

## 1 The Two Level MetaArchitectural Model

To simplify development of applications, we use Actors [1], a model of concurrent active objects that has a built-in notion of encapsulation and interaction among the concurrent components of an ODS. In the actor paradigm, the universe contains computational agents called *actors*, distributed over a network. Traditional passive objects encapsulate state and a set of procedures that manipulate the state; actors extend this by encapsulating a thread of control as well. Each actor potentially executes in parallel with other actors and may communicate with other actors via asynchronous message passing.

In [24], we presented the TLAM(Two Level Actor Machine) model as a first step towards providing a formal semantics for specifying and reasoning about properties of and interactions between components of ODSs. 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 behavior of the base level. Meta-actors communicate with each other via message passing as do base level actors, but they may also examine and modify the state of the base actors located on the same node.

Meta-level controllers define protocols and mechanisms that customize various aspects of distributed systems management. In practice, multiple system and application activities occur concurrently in a distributed system, e.g. scheduling, protocol processing, stream synchronization and can therefore interfere with each other. Composing multiple resource management mechanisms leads to complex interactions. Consider the following example of a system where distributed garbage collection and process migration can proceed concurrently. If processes in migration (transit) are not accounted for, the garbage collection process can potentially destroy accessible information. Similarly, if the process is continuously migrating, the garbage collection process runs the risk of potential nontermination. In general, risks that arise due to mechanism composition include loss of information, possible nonterminations that cause deadlocks and livelocks, dangling resources, inconsistencies, and incorrect execution semantics.

One approach to deal with interference during mechanism composition in an open system is to serialize or delay activities to ensure overall safety of the system. However, this can result in over-serialization of resource management activities causing performance degradation. Furthermore, global delays and halts may cause violations of timing based QoS constraints. We also argue that composability of resource management activities is not just desirable, but *essential* to ensure cost-effective QoS in distributed systems. For instance, system protocols and activities must not enforce arbitrary delays in the presence of timing based QoS constraints.

To ensure non-interference and manage the complexity of reasoning about components of ODSs in general, our strategy is to identify key system services where non-trivial interactions between the application and system occur, i.e. base-meta interactions. We refer to these key services as *core services*. Core services are then used in specifying and implementing more complex activities within the framework as purely meta-level interactions. The development of suitable non-interference requirements allows us to reason about composition of multiple system services; these services have constraints that must be obeyed to maintain composability. We have identified three core activities:

- **Remote Creation:** - Recreation of services/data at a remote site. Remote creation can be used as the basis for designing algorithms for activities such as migration, replication and load balancing.
- **Distributed Snapshot:** Capturing information at multiple nodes/sites used as a basis for distributed garbage collection (cf. [21]).
- **Directory Services:** Interactions with a global repository. Directory services can be used to provide access control and implement group communication protocols.

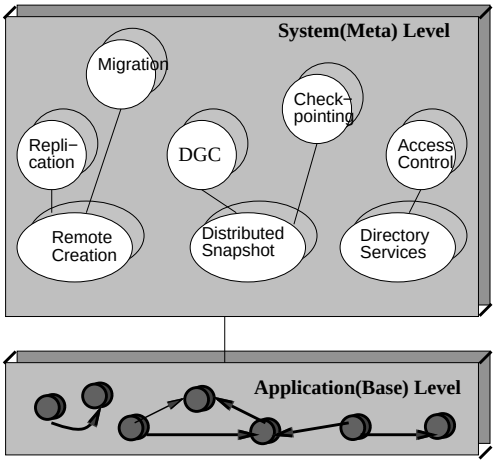


**Figure 1. Classification of Core Services**

## 2 The CompOSE|Q Framework

Based on the two level metaarchitecture, we are developing a customizable and safe distributed systems middleware infrastructure, called CompOSE|Q (Composable Open Software Environment with QoS), that has the ability to provide cost-effective QoS-based distributed resource management. The primary distinguishing feature of CompOSE|Q, is that it provides *composable distributed resource management*, i.e.,it allows the concurrent execution of multiple resource management policies in a distributed system in a safe and correct manner. This will allow safe integration of mechanisms for services such as mobility, load balancing, fault tolerance and end-to-end QoS management. Within CompOSE|Q, we develop algorithms and mechanisms for services such as scheduling, coordination, load management and resource discovery.

CompOSE|Q will contain:

- Modules that implement the 3 basic composable core services - remote creation, distributed snapshot and directory services.
- Interfaces and interaction requirements (i.e. usage constraints) for core services.
- Common services built using core services - actor migration, replication of services and data, actor scheduling, distributed garbage collection, checkpointing,

name services etc. Each of these services have their own interface definitions and interaction constraints.
- QoS specification and enforcement mechanisms.

Providing support for common services in CompOSE|Q enables the implementation of sophisticated policies and mechanisms for resource management. For instance, scheduling mechanisms will be implemented using the basic remote creation core service to address two levels of scheduling - (1) object scheduling that assigns newly created objects/actors on nodes and (2) message scheduling that addresses scheduling of messages destined for existing actors. The basic scheduling techniques will then be extended to support policies for priority and constraint-based scheduling. 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. The directory core service is used 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.

**Enhancing Middleware to Support QoS:** The following aspects are currently being addressed to enhance the basic framework to support QoS. (1) QoS Specification: Development of mechanisms for specifying QoS. (2) QoS Enforcement: Design and implementation of services and basic functionalities required to enforce QoS at runtime.

**QoS Specification:** QoS statements may specify constraints on timing , availability, security and resource utilization at various levels of abstraction. Abstract properties such as correct data delivery and uninterrupted service can be translated into concrete parameters such as jitter, end-to-end delay, synchronization skew and/or concrete resource requirements such as network and disk bandwidth and buffer requirements. We have proposed a special entity called the *QoS Synchronizer* [13] that incorporates the notion of a QoS-enabled session. *QoS Synchronizers* encapsulate and manage QoS constraints and verify that constraints within the session are feasible. Independently specifying the QoS constraints and the service gives us the ability to modify one without impacting the other. Dynamic changes in QoS specifications (cost and service level tradeoffs) or changes in system configuration and parameters (change in resource availability, addition/removal of resources while a request is ongoing) can be expressed concisely at a level independent of the application semantics.

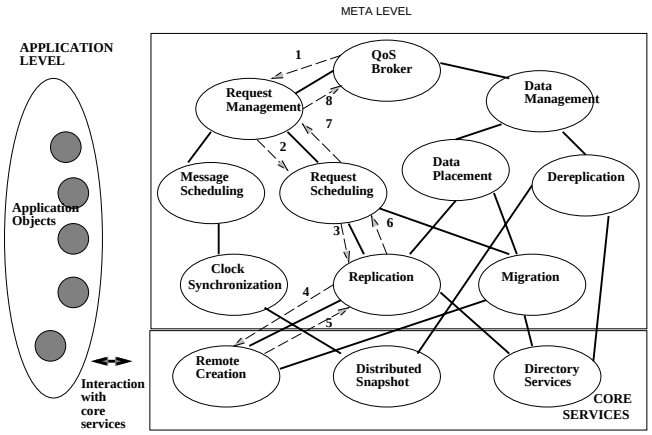


**Figure 2. Architecture of the CompOSE|Q System. The dotted lines indicate the flow of an incoming request through the middleware modules.**

**QoS Enforcement in the Middleware Framework:** In this section, we extend the basic metaarchitectural framework to provide QoS based services to applications. The base level component of the meta-architecture implements the functionality of the distributed session and deals with (a) data which includes objects of varying media types, e.g., 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 sharing of existing resources among multiple requests. To provide coordination at the highest level and perform admission control for new incoming sessions, a meta-level entity called the *QoS broker* [12] is being developed. The organization of meta-level services in CompOSE|Q is illustrated in Figure 2; the services are mapped to a specific distributed video server architecture [23, 19]. Each of the services described above can be based on one or more of the core services - remote creation, distributed snapshot and the directory service. Formal reasoning of QoS properties within a meta-architecture model in the presence of other activities has been discussed in [19].

The two main functions of the QoS broker are (a) data management and (b) request management. The *data management* component decides the placement of data in the distributed system, i.e., it decides when and where to create additional replicas of data. It also determines when additional replicas of data actors are no longer needed and can be garbage collected/dereplicated. The *request management* component performs the task of admission control for incoming requests and ensures the satisfaction of QoS constraints for requests that are ongoing in the system. We have developed adaptive admission control mechanisms [23] in the *request scheduling* module that assigns

requests to servers and ensures cost-effective utilization of resources. The *message scheduling* module ensures QoS constraint satisfaction of requests that have already been initiated. The QoS constraints are specified in declarative entities - QoS Synchronizers discussed earlier. The data and request management functions in turn require basic services such as:

•*Clock Synchronization:* in order to maintain a uniform notion of time among multiple nodes and thereby among actors in the session that are distributed across multiple nodes.

•*Replication:* to replicate data and request actors using adaptive and predictive techniques for selecting where, when and how fast replication should proceed.

•*Dereplication:* to dereplicate/garbage-collect data or request actors and optimize utilization of storage space in the distributed system based on current load in the system as well as expected future demands for the object.

• *Migration:* to migrates data or requests for load balancing, availability and locality. The interaction of migration with timing based QoS constraints is an interesting issue. For instance, in MM applications, significant playback jitter that can be introduced by explicit tear-down and re-establishment of network connections must be minimized.

## 3 Providing Application Requirements in $\mathrm{CompOSE|Q}$

We are investigating mechanisms for hierarchical metalevel control that composes modules implementing application requirements such as security, fault tolerance and mobility in a modular and composable manner.

**Security:** Traditional security mechanisms were rooted in a security kernel (within the OS) that enforced a single security policy throughout the system [14]. Future systems must enforce security at multiple levels ranging from implementing access control for legitimate users of a system to cryptographic protocols that guard against illegitimate users. Protection to users within a system can be provided via access control techniques using protection domains, access control matrices and capabilities. Securing the system from illegitimate users requires further investigation of possible security attacks, intrusion detection policies, key-based encryption algorithms, techniques such as digital signatures and watermarking. In general, it is paramount that resource allocation for any application is robust and free from security attacks. The generalized directory service in $\mathrm{CompOSE|Q}$ can be used to define access control and security mechanisms that provide global accessibility while ensuring protection of data from use in unwarranted ways. This will require an investigation of the tradeoffs between extensibility and security since the ability to arbitrarily customize a system can result in security breaches. Developing access control mechanisms and capabilities within the TLAM framework has been discussed in [19]; these mechanisms will be implemented in $\mathrm{CompOSE|Q}$.

**Mobility:** We are already observing a trend toward migratory and completely self-contained, portable applications via programming languages such as Java. A metalevel mechanism can be used for managing mobility and for communicating between the different mobile components. $\mathrm{CompOSE|Q}$ will incorporate services that enable the development of mobile applications by providing support for the creation, execution and migration of objects and deal with the composition of migratory agents with other system customizations. Reflection is a useful concept in developing mobility mechanisms since it provides higher level abstractions that reify executing code and treat the code representation as data that can be relocated. In $\mathrm{CompOSE|Q}$, the migrating computations are modeled as base-actors and the metaarchitecture is the reflective framework that coordinates base-actor execution in a distributed system. We have addressed composability issues of actor migration with a reachability snapshot service [20]. Further work is required to generalize these mechanisms, embed mobility information into the directory service and understand the impact of system management activities on migrating components.

**Secure Mobility:** Understanding complex interactions between mechanisms that facilitate mobile code and those that implement security policies requires a modular representation of the services implementing secure mobility. Current techniques for secure mobility implement a series of low-level checks; there is a lack of formal models for expressing the interaction between security and mobility using higher level abstractions. We are currently investigating techniques for specifying and enforcing non-interference of mechanisms that implement security and mobility in $\mathrm{CompOSE|Q}$ using an existing logic framework based on concurrent rewriting semantics [10].

## 4 Implementation Issues and Development environment for $\mathrm{CompOSE|Q}$

We have taken an evolutionary approach to developing the full-fledged customizable middleware infrastructure, $\mathrm{CompOSE|Q}$. The $\mathrm{CompOSE|Q}$ environment consists of the following components: (1) a programming environment based on concurrent objects (2) a middleware library that manages execution of applications on nodes, coordinates distribution and provides services. (3) a compact runtime component that resides on the nodes of the distributed system which interacts with the distributed component of the middleware.

For the initial implementation of $\mathrm{CompOSE|Q}$, we derive components from the actor-based runtime system - THAL [6]; these components implement basic services such as actor scheduling, message communication and actor creation. The middleware layer of $\mathrm{CompOSE|Q}$ cur-

rently supports remote creation, a basic name service and a rudimentary state capture(distributed snapshot) mechanism. We are concurrently developing modules for QoS management such as the QoS broker, data and request management modules etc. to be integrated into $\mathbf{CompOSE|Q}$. We are in the process of implementing adaptive and predictive policies for data placement and measuring the performance overheads of these policies in the presence of QoS requirements. $\mathbf{CompOSE|Q}$ will contain a meta-level implementation of QoS Synchronizers to satisfy application specific QoS constraints. The middleware must deal with application specific constraint evaluation techniques and failure handling strategies that will need to take effect when the constraints are not satisfied. The runtime system consists of a front-end that runs on a controller node and a set of runtime kernels that run on the individual nodes in the network. Each kernel contains a meta-level node manager actor. The kernel is a passive substrate on which actors execute. The kernel interfaces with the underlying machine via a communication module.

The application interface consists of a set of Java classes through which application programs access the runtime library. Powerful constructs for efficiently controlling synchronization and coordination of distributed entities is required. The use of the *synchronized* keyword in Java for thread coordination leads to over-serialization resulting in a loss of performance. The ability to deal with the management of thread priorities for real-time thread management in Java is dependent on the underlying threads implementation, making QoS support complicated to achieve. For these reasons, we will implement a language with Java-like syntax but with actor semantics through an additional library. The language will be appropriately extended with real-time constraints and abstractions for the specification of QoS requirements. The developed environment will then be used as a framework for developing more sophisticated services in $\mathbf{CompOSE|Q}$ and for prototyping applications to evaluate the performance of distributed algorithms. The hardware platform used for the prototype implementation is a network of workstations.

## 5 Related work and Future Research Directions

Commercially available object-based middleware infrastructures including CORBA and DCOM represent a step toward compositional software architectures but do not deal with interactions of multiple object services executing at the same time, and the implication of composing object services. The Electra framework [9] extends CORBA to provide support for fault tolerance using group-communication facilities; real-time extensions to CORBA [15, 25] to support timing-based QoS requirements have been proposed. Systems such as Infospheres [2] and Globe [18] use a distributed object model to construct large scale distributed systems. Globus, a metacomputing framework defines a QoS component called Qualis [8] where low level QoS mechanisms can be integrated and tested. Traditional reflective languages and systems aim at providing a customizable execution of concurrent systems [16]. The Aspect Oriented Programming paradigm [5] makes it possible to express programs where design decisions(aspects) can be appropriately isolated permitting composition and re-use of code. Reflective systems being developed include Apertos [4], 2K [7] and Broadway [17]. QoS enforcement has been a topic of considerable research in the multimedia community. The Omega architecture [12] developed real-time communication protocols and QoS brokers at the endpoints to supply end-to-end QoS. *QualMan* [11] is a QoS aware resource management platform that provides negotiation, admission and reservation capabilities for sharing end-system resources. The *EPIQ* project [3] provides interfaces, mechanisms, and protocols to support QoS management of flexible applications.

We are actively working on extending $\mathbf{CompOSE|Q}$ to support more distributed services for security, mobility and fault-tolerance. The basic $\mathbf{CompOSE|Q}$ infrastructure can also be used to provide a prototypical network-centric computing paradigm that requires the integration of network and system policies with applications. We will address techniques that provide decisions about the degree of network awareness that applications and middleware must possess to ensure performance under varying network conditions. Modeling client interaction requires a notion of session and resources within a session. Further work is required to provide a generalized model that captures the architectural resources required in the server and network to support the session connection. We are also developing techniques for translating QoS requirements for each session into resource constraints and executing admission control so that the overall system safety and correctness is not compromised. To provide end-to-end QoS, it is necessary to determine how *real-time scheduling* strategies interact with strategies for tasks requiring CPU intensive calculations, or network communication with clients.

Prior work on defining unified metrics of QoS addresses the price-performance tradeoff [22] using an economic framework and develops negotiation protocols that express and implement resource tradeoffs. Specification mechanisms for QoS that permit the definition of these tradeoffs and tie in with mechanisms in the middleware that will execute these tradeoffs safely and correctly is an interesting area of future research. The implication of dynamic negotiation on distributed resource allocation mechanisms and the impact on QoS caused by enforcing the negotiated changes requires further investigation. The request management component of $\mathbf{CompOSE|Q}$ must deal with ne-

gotiation protocols and enforce negotiation changes in a request without impacting QoS of other requests in a cost-effective manner. In general, the dynamic nature of applications under varying network conditions, request traffic, etc. imply that resource management policies must be dynamic and customizable. We believe that composable middleware frameworks that implement cleanly defined meta-architectures enable customization of applications, protocols and system services; this will provide a foundation for the evolution of large scale distributed computing.

**Acknowledgements:** The author would like to thank Carolyn Talcott and Gul Agha with whom the background work for this framework was developed and for insightful discussions during the course of this work.

## References

[1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, Mass., 1986.

[2] K. Chandy, A. Rifkin, P. A. Sivilotti, J. Mandelson, M. Richardson, W. Tanaka, and L. Weisman. A world-wide distributed system using java and the internet. In *Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-5), Syracuse, New York*, Aug. 1996.

[3] D. Hull, A. Shankar, K. Nahrstedt, and J. W. Li. An end-to-end qos model and management architecture. In *Proceedings of IEEE Workshop on Middleware for Distributed Real Time Systems and Services, San Francisco*, Dec. 1997.

[4] J. ichiro Itoh, R. Lea, and Y. Yokote. Using meta-objects to support optimization in the Apertos operating system. In *USENIX COOTS (Conference on Object-Oriented Technologies*, June 1995.

[5] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings of ECOOP'97 European Conference on Object-Oriented Programming*, June 1997.

[6] W. Kim. *Thal: An Actor System for Efficient and Scalable Concurrent Computing*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[7] F. Kon, A. Singhai, R. H. Campbell, D. Carvalho, R. Moore, and F. J. Ballesteros. 2K: A Reflective, Component-Based Operating System for Rapidly Changing Environments . In *Proceedings of ECOOP'98 Workshop on Reflective Object-Oriented Programming and Systems*, Brussels, Belgium, July 1998.

[8] C. Lee, C. Kesselman, J. Stepanek, R. Lindell, S. Hwang, B. S. Michel, J. Bannister, I. Foster, and A. Roy. The quality of service component for the globus metacomputing system. *Proceedings of IWQoS '98*, pages 140–142, 1998.

[9] S. Maffeis and D. Schmidt. Constructing reliable distributed communication systems with corba. *IEEE Communications*, 14(2), February 1997.

[10] J. Meseguer and C. Talcott. Rewriting logic and secure mobility. In *Proceedings of the NPS Workshop*, 1997.

[11] K. Nahrstedt, H.-H. Chu, and S. Narayan. Qos-aware resource management for distributed multimedia applications.

[12] K. Nahrstedt and J. M. Smith. The qos broker. *IEEE Multimedia*, 2:53–67, 1995.

[13] S. Ren, N. Venkatasubramanian, and G. Agha. Formalizing qos constraints using actors. In *Proceedings of Second IFIP International Conference on Formal Methods for Open Object Based Distributed Systems,FMOODS'97*, July 1997.

[14] J. Rushby. Design and Verification of Secure Systems . In *Proceedings of the 8th ACM Symposium on Operating System Principles*, 1981.

[15] D. C. Schmidt, D. Levine, and S. Mungee. The design of the tao real-time object request broker. *Computer Communications Special Issue on Building Quality of Service into Distributed System*, 1997.

[16] B. C. Smith. *Reflection and Semantics in a Procedural Langauge*. PhD thesis, Massahusetts Institute of Technology, Jan. 1982.

[17] D. Sturman. *Modular Specification of Interaction Policies in Distributed Computing*. PhD thesis, University of Illinois at Urbana-Champaign, May 1996. TR UIUCDCS-R-96-1950.

[18] M. van Steen, A. Tanenbaum, I. Kuz, and H. Sip. A scalable middleware solution for advanced wide-area web services. In *Proc.Middleware '98, The Lake District, UK*, 1998.

[19] N. Venkatasubramanian. *An Adaptive Resource Management Architecture for Global Distributed Computing*. PhD thesis, University of Illinois, Urbana-Champaign, 1998.

[20] N. Venkatasubramanian. *An Adaptive Resource Management Architecture for Global Distributed Computing*. PhD thesis, University of Illinois, Urbana-Champaign, 1998.

[21] N. Venkatasubramanian, G. Agha, and C. L. Talcott. Scalable distributed garbage collection for systems of active objects. In *International Workshop on Memory Management, IWMM92, Saint-Malo*, LNCS, 1992.

[22] N. Venkatasubramanian and K. Nahrstedt. An integrated metric for video qos. In *Proceedings of ACM Multimedia '97, Seattle, Washington*, Nov. 1997.

[23] N. Venkatasubramanian and S. Ramanathan. Effective load management for scalable video servers. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS97)*, May 1997.

[24] 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.

[25] V. F. Wolfe, J. K. Black, B. Thuraisingham, and P. Krupp. Real-time method invocations in distributed environments. In *Proceedings of the HiPC'95 Intl. Conference on High Performance COmputing*, 1995.