An Actor Based Framework for Managing Multimedia QoS

Nalini Venkatasubramanian and Gul A. Agha
Department of Computer Science,
1304 W. Springfield Avenue,
University of Illinois at Urbana-Champaign,
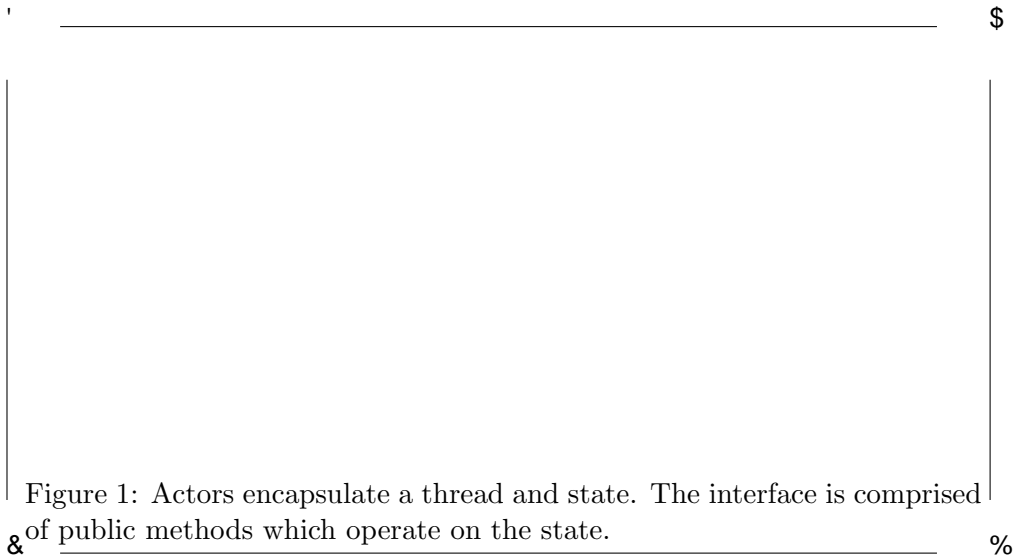Urbana, IL 61801, USA,
Email: `nalini/agha@cs.uiuc.edu`, Web: `http://www-osl.cs.uiuc.edu`

1   Motivation

Services and systems in a global multimedia information infrastructure evolve dynamically and their components interact with an environment that is not under their control. Many multimedia (MM) applications can tolerate relatively minor and infrequent violations of their performance requirements; the extent to which such violations are permissible is specified as a quality-of-service (QoS) parameter. In this paper, we discuss preliminary work on an actor-based framework for the specification and management of open distributed multimedia systems. QoS statements may specify constraints on timing, availability, security, and resource utilization. By developing a formal theory that models QoS features and captures the dynamicity of evolution, we facilitate the design of open systems. A formal model also allows us to reason about the composition of objects with varying QoS constraints and restrict interactions for safe compositionality.

1.1   Actors and Multimedia

A drawback with some implementation driven methods of QoS specification is that the specification of QoS requirements is intermixed with the service specification [5, 6]. While the correctness of program execution, e.g. data delivery, relies on meeting the QoS requirements, merging the specifications results in loss of modularity and complicates the correctness validation process. Other approaches address formal description methodologies for specifying QoS via dual language techniques that specify functional behavior and

Figure 1: Actors encapsulate a thread and state. The interface is comprised of public methods which operate on the state.

QoS constraints distinctly using two different languages [3]. These methodologies are based on synchronous communication and therefore hide the overhead of synchronization and communication. The synchronous communication hypothesis is hard to realize because real systems cannot run infinitely fast. Furthermore, these systems do not account for the overheads and delays introduced by the presence of multiple activities sharing global system resources. Hence, the translation of these formal descriptions into executable code and the subsequent performance tuning capabilities are not clear. In this paper, we present a modular, customizable, system-driven approach to managing multimedia QoS in distributed systems. Our approach is based on a formal framework that gives us the ability to reason about interactions among activities in a distributed multimedia system.

In general, models and methodologies that have been developed for sequential programming are inadequate for creating correct distributed applications: to simplify distributed programming, a model of open concurrent computation that provides facilities for the modular specification of distributed interactive applications is needed. The *Actor* model of computation [1] has a built-in notion of encapsulation and interaction, and thus it is a natural model to use as a basis for interactive applications. Actors can be viewed as a model of coordination between autonomous interacting components (See Figure 1).

The local computation carried out by the components may be specified in any sequential language. Actors can be dynamically created. They

communicate via message passing which is asynchronous and fair. The communication topology of an actor system is called the *acquaintance relation* and can change dynamically. Semantics of actor interactions are relatively well understood and reasoning about systems of actors has been formalized [2, 12].

Specifying QoS in the Actor based model is essentially a problem of specifying coordination constraints between distributed objects. Synchronizers [4] allow us to express these coordination constraints, i.e. local synchronization constraints or multi-actor coordination constraints.Synchronizers also allow us to qualitatively control the semantics of message delivery. However, Synchronizers do not incorporate the notion of quantitative time. The basic Actor model captures the fundamental properties of general purpose distributed computing in which only logical time is concerned. Individual objects are constrained only by the computational causal order. However, in distributed multimedia systems, in addition to computational causal order that ensures the computational correctness, quantitative precedence orderings among different (possibly independent) computing units have to be enforced in order to achieve the QoS requirements. Earlier work separates real-time constraints from the computational aspects of an application; real-time constraints are described by synchronization code between the interfaces of objects [9]. A high-level programming language construct called *RTsynchronizer*, a real-time variant of actors, specifies a collection of temporal constraints between actors. This approach separates what an object does from when it does it allowing us to specify timing based QoS constraints modularly, dynamically modify them and verify the correctness of constraint satisfaction.

2   QoS Synchronizers

The structure of a unified system is depicted in Figure 2. We can visualize MM applications as a collection of autonomous, concurrent information processing actors called *media-actors* involved in multimedia sessions. A session consists of a set of media-actors that interact with each other to achieve some common goal, e.g. a video-on-demand session or a multimedia conferencing session. QoS requirements are typically associated with a multimedia session, e.g., the audio-video synchronization skew must be bounded to 80ms. To incorporate the notion of a session in the Actor model, we propose a special entity called the *QoS Synchronizer* that manages QoS constraints and specifications for a particular session, and verifies

Figure 2: Separation of concerns - QoS requirements specified by system level QoS actors and application functionality encapsulated in media-actors.

that constraints within a session are feasible. *QoS Synchronizers* are special entities that abstract and encapsulate the real-time QoS requirements for a session, e.g.synchronization or timing-related information. A QoS Synchronizer can control the execution of media-actors by observing transitions on the media-actors in the session. It is different from an actor in that it does not interfere with the behavior of the underlying computational actors that it observes. In particular, the QoS Synchronizer cannot send messages to or receive messages from the underlying media actors. It impacts execution by constraining when the media-actors in the session react and respond to events and thereby restricts when a particular computational behavior can occur. In addition, the internal state of a QoS Synchronizer may be updated as a result of observing events on media-actors.

Using the concept of QoS Synchronizers permits the modularization of QoS requirements and encourages separation of orthogonal concerns - functionality and execution constraints. By specifying program behavior and QoS constraints separately, we reason about program and system correctness independent of the constraints. We can also separately reason about the validity and satisfiability of the QoS constraints, then embed the QoS constraints in the system in a modular way and reason about the composite correctness. Independently specifying the QoS constraints and the service also gives us the ability to modify one without impacting the other. Dynamic changes in QoS specifications, e.g. cost and service level tradeoffs or changes in system configuration and parameters e.g. change in resource availability, addition/removal of resources while a request is ongoing etc.,

can be expressed concisely at a level independent of the application semantics. Since a multimedia session often involves the coordination of multiple media-actors (for e.g. audio/video related), QoS Synchronizers allow the programmer to treat the session as a group of actors instead of having to deal with each component independently. In [11], we illustrate examples of programming with QoS Synchronizers.

In general, a multimedia session consists of (1) a collection of actors (media-actors) that are involved in that session (components at the server and client ends), (2) a session Synchronizer that holds and enforces QoS constraints between media-actors in a session, and (3) a notion of time that is uniform among media-actors of that session.

There are two kinds of timing constraints expressed within a session - absolute timing constraints and relative timing constraints. Relative timing constraints are expressed relative to the occurrence of another event in the system. Absolute timing constraints refer to given points in time with respect to some frame of reference, the accuracy of the constraint is based on the granularity of time progress in that frame of reference. We assume that each QoS Synchronizer has an implicit notion of time and time progress within a session that accurately reflects a chosen standard, e.g. wall clock time. This is encoded as part of the state of a QoS Synchronizer - the virtual time for the session and is used as the clock for all activities in that session.

In a system with multiple sessions, we need to address the issue of satisfying QoS constraints for each session. To facilitate this, we propose an agent called the *QoS broker* [7] that acts as a coordinator for all the ongoing sessions and performs admission control for new incoming sessions. Since the overall system QoS cannot be violated, every QoS Synchronizer must interact with the QoS broker. Design criteria for an actor-based QoS broker, its components and interactions in a distributed multimedia environment is a future area of research [13].

2.1 Formalizing MM Sessions

We use the notion of a configuration to provide an instantaneous representation of an actor system with respect to some idealized observer.

We formally define a session configuration as a structured entity representing the system state at a given time instance (similar to [10]) and use it to express session-based quantitative time. An explicit time actor in this case gives us a virtual time with respect to an ongoing session in the system ($\tau_{session}$).

Definition 1 (SessionConfiguration)

$$\Big\langle\!\Big\langle\ \alpha,\ (\tau)_{\texttt{Session}}\ \ \Big|\ \ \mu\ \ \Big|\ \ \langle\ \sigma_s \parallel \varsigma\ \rangle\ \Big\rangle\!\Big\rangle_{\chi}^{\rho}$$

where

- α is an actor map, which maps a finite set of actor names to their behavior, including media-actors;

- μ is a finite multi-set of messages, yet to be processed;

- ρ and χ are finite sets of actor addresses; χ represents the external actors, i.e. the environment external to the configuration and ρ represents the receptionists, actors that act as an interface between the session and the external environment.

- σ_s is an session map, which maps a finite set of QoS Synchronizer names to their states; In the case of a single session, we reduce this to a mapping between the QoS Synchronizer and its state $(S)_{QoS}$. This mapping holds the static QoS constraint expressions that must be satisfied.

- ς is a multi-set of dynamic expectation instances on which QoS constraints must be satisfied - it represents dynamic messages that need to be invoked and the invocation period of these messages.

- $\tau_{Session}$ is the current virtual time with respect to the session. Any point in time is represented by a nonnegative real number, i.e., $\tau_{Session} \in \Re^{+}$.

Execution in the actor system is then represented by a transition relation on the set of configurations. If the co-existence of multiple sessions is permitted in the system, then each session has its own notion of time and time progress. However, if the sessions need to interact, for instance, when two QoS Synchronizers operate on the same media actor, there may be inconsistencies that must be resolved. In this case, a mechanism for synchronizing the clocks of the co-existing sessions must be provided. Discussions on the correctness of session based semantics and the compositionality of sessions can be found in [11].

3   A Meta-level Distribution Infrastructure

A meta-level distribution infrastructure bridges the gap between the specification of QoS constraints and the realization of the specified QoS requirements in a distributed multimedia system. End-to-end solutions to QoS imply that the constraints must be obeyed by all the elements in the application path [8]. In practice, multiple system and application activities will need to occur concurrently in a distributed multimedia system, e.g., scheduling, monitoring, inter and intra stream synchronization, protocol processing. We need to ensure that the enforcement of QoS constraints does not violate the correctness of other activities present in the system and vice-versa. In order to compose services and allow multiple sessions to co-exist and satisfy QoS constraints, we need to ensure the following:

- non-interference of services within a session, since one service may impact the timing(QoS) or correctness requirements of another service required for the session,

- non-interference of services that span multiple sessions, i.e. since multiple sessions may share the same resources, and,

- non-interference of multiple sessions that share media-actors, i.e. since each session may have a different QoS requirement from the underlying media actor.

What we require is a framework that will permit tasks to interact with each other satisfying the specified QoS constraints in non-interfering ways while obeying the interfaces and invariants of the composite system.

In [14], we propose a two-level meta-architectural model of distributed computation based on actors. We use this model to specify distributed system services at various levels, prove properties of these specifications, combine the services in various ways, and reason about their interactions. This model provides a formal semantic framework that provides support for dynamic customizability and separation of concerns in designing and reasoning about components of open distributed systems (ODS). We use this framework to isolate orthogonal issues such as: functional behavior of an application; QoS constraints that must be satisfied by an application and resource management issues such as memory management, load balancing, and scheduling.

In the two-level metarchitectural model, we distinguish between two kinds of subsystems - application subsystems and meta-level subsystems.

Application subsystems are encapsulated modules invoked directly by the application programmer, e.g. video/audio subsystems. A meta-level subsystem is designed by a systems architect and holds the internal details of the architecture and management algorithms employed to deliver the service. Correspondingly, 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 by explicitly representing some of the runtime structures, resources and protocols. 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 they control.

QoS Synchronizers are essentially declarative entities that hold the QoS constraints for that session. There are therefore many ways of implementing the enforcement of the constraints captured by QoS Synchronizers on the media actors being constrained. One implementation would be to distribute a part of the QoS Synchronizer state to each media-actor being controlled. This implies that the media-actors themselves directly communicate with each other to enforce constraint satisfaction. A more centralized implementation consists of a single entity that represents the synchronizer through which the media actors are controlled. The implementation of Synchronizers in [4] uses a hybrid approach where some constraints are implemented using a centralized entity and some constraints are distributed among the controlled actors. Constraint evaluation techniques may be (a) pessimistic where all constraints are evaluated before scheduling a message or (b)optimisticwhere messages may be scheduled for execution before the satisfiability of all constraints has been determined. The choice of pessimistic or optimistic constraint evaluation is dependent on the application requirements. If the timing constraints expressed in the synchronizer are hard real-time constraints, then a pessimistic evaluation of constraints is required prior to message scheduling.

We propose a meta-level implementation of QoS Synchronizers that observe events on media-actors and control the invocation of messages on media actors. In this case, media actors constitute the application or base-level subsystem. In order to reason about the implementation, we propose meta-level implementation entities that represent the QoS Synchronizer known as QoS meta-actors (similar to constraint servers in [4]) that monitor events on base-level media-actors. The QoS meta-actors evaluate timing constraints and communicate them to a message scheduler which is itself another meta-level actor controlling the mailqueue orderings of media actors. The con-

Figure 3: An Actor Based Distributed Multimedia System

straint evaluation technique for multimedia information is based on the type of information and possibly on the characteristics of the packet as well. For instance, the optimistic evaluation technique would schedule the display of a video frame or the playing of an audio packet even if timing constraints were violated. Unfortunately, rollback in the case of failure is not a feasible option if the event has occured, i.e. video frame has already been displayed. If a video frame or audio packet is waiting in a play queue, however, it will be possible to drop the frame or packet. However, this degree of micro-scheduling, if permitted, is expensive on general purpose architectures. Note that timing violations on audio packets are more disruptive than QoS violations on video frames, hence one may choose optimistic evaluation of video constraints and pessimistic evaluation of audio constraints. This implies additional overhead in examining the contents and type of the message before evaluating constraints for message scheduling. Given the complications and overheads associated with optimistic constraint evaluation, we assume pessimistic evaluation of all multimedia constraints in our design.

We also extend meta-level architectures to coordinate multiple resource management activities in distributed multimedia systems by introducing one or more distinct meta-actors, known as *QoS brokers*, as illustrated in Figure 3. QoS brokers deal with (a) interactions between multiple QoS sessions that share media-actors through the corresponding QoS Synchronizers and (b) interactions between the QoS Synchronizers and other system services.

The *QoS broker* has two types of functions:

- Static functions: *QoS brokers* define protocols and mechanisms that can customize the placement,scheduling and management of media actors. They communicate among the different entities in the system executing protocols for negotiation and renegotiation of system resources, connection establishment, resource reservation and admission control. QoS brokers also provide the necessary interfaces to translate end-to-end QoS specifications into resource related metrics that can then be used by the OS for resource allocation.

- Dynamic functions: QoS brokers can be used to monitor multimedia sessions to ensure that specified guarantees are met during execution. To perform this, the broker behaves as a dynamic automated performance agent that assimilates, transports and correlates predetermined performance metrics and verifies that performance guarantees are met.

The availability of QoS brokers also provides customizable services and adds a new dimension of control, flexibility and openness to system management. For instance, using the meta-level broker, QoS Synchronizer implementations can be augmented for dynamic installation of protocols to interpret MM data. A preliminary specification for a QoS broker that facilitates object placement, management, request scheduling and admission control in a distributed multimedia system is discussed in [13].

4 Conclusions

In this position statement, we have proposed an actor-based approach to specifying and managing QoS in open distributed multimedia systems via QoS Synchronizers and brokers. We propose one implementation of QoS Synchronizers through a meta-architecture framework [14]. Meta-architectures can form the basis for adaptive environments that provide customizable services needed to ensure end-to-end guarantees to multimedia traffic. We are currently exploring the specification of QoS in the presence of multiple sessions and the composability of QoS constraints in the meta-architectural framework.

References

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

[2] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. Towards a theory of actor computation. In *The Third International Conference on Concurrency Theory (CONCUR '92)*, volume 630 of *Lecture Notes in Computer Science*, pages 565–579. Springer Verlag, August 1992.

[3] Howard Bowman, Gordon Blair, Lynne Blair, and Amanda Chetwynd. Formal description of distributed multimedia systems: An assessment of potential techniques. *Computer Communications*, 18(12), December 1995.

[4] Svend Frølund. *Coordinating Distributed Objects: An Actor-Based Approach to Synchronization*. MIT Press, 1996.

[5] Peter Leydekkers and Valerie Gay. Odp view on qos for open distributed mm environments. In Jan de Meer and Andreas Vogel, editors, *4th International IFIP Workshop on Quality of Service, IwQos96 Paris, France*, pages 45–55, March 1996.

[6] Flavio Henrique de Souza Lima and Edmundo Roberto Mauro Madeira. Odp based qos specification for the multiware platform. In Jan de Meer and Andreas Vogel, editors, *4th International IFIP Workshop on Quality of Service, IwQos96 Paris, France*, pages 45–55, March 1996.

[7] Klara Nahrstedt and Jonathan M. Smith. The qos broker. *IEEE Multimedia*, 2:53–67, 1995.

[8] Klara Nahrstedt and Ralf Steinmetz. Resource management in networked multimedia systems. *IEEE Computer*, 28(5):52–65, May 1995.

[9] S. Ren, G. Agha, and M. Saito. A modular approach for programming distributed real-time systems. *Journal of Parallel and Distributed Computing*, 36(1), July 1996.

[10] Shangping Ren. *Modularization of Time Constraint Specification in Real-time Distributed Computing (to be published)*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1997.

[11] Shangping Ren, Nalini Venkatasubramanian, and Gul Agha. Formalizing multimedia qos constraints using actors. In *IFIP Workshop on Formal Methods for Open Object-based Distributed Systems, FMOODS'97*, 1997.

[12] C. L. Talcott. Interaction semantics for components of distributed systems. In *1st IFIP Workshop on Formal Methods for Open Object-based Distributed Systems, FMOODS'96*, 1996.

[13] N. Venkatasubramanian. *Composing Distributed Resource Management Activities (to be published)*. PhD thesis, University of Illinois, Urbana-Champaign, 1997.

[14] N. Venkatasubramanian and C. L. Talcott. Reasoning about Meta Level Activities in Open Distributed Systems. In *Principles of Distributed Computing (PODC95)*, 1995.