Formalizing Multimedia QoS Constraints Using Actors

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

Abstract

The vision of future information systems is that different forms of information are potentially accessible at anytime through the Internet. We describe challenges in the modeling and specification of timing related multimedia (MM) services in open distributed systems. Management of multimedia services in an open system is complicated by the heterogeneity of application requirements, multimedia information, and system components. Services and systems in this environment evolve dynamically and their components interact with an environment that is not under their control. We propose a formal specification of timing related Quality-of-Service (QoS) attributes in an actor-based distributed system and describe some techniques for informally reasoning about quantitative QoS properties.

Keywords

QoS, Real-time, Multimedia, Actors, Concurrency semantics, Sessions

1 MOTIVATION

Services and systems in a global information environment like the WWW 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 the specification and semantics of timing based Quality-of-Service (QoS) parameters in multimedia applications. Our goal is to provide formal specifications of QoS in a distributed MM system and thereby reason about the formal val-

idation of quantitative properties. QoS statements may specify constraints on timing, availability, security, and resource utilization. However, we will only address the specification of timing based QoS constraints.

Multimedia applications are characterized by the presence of QoS requirements for resolution, tolerable jitter, delays etc. With multimedia applications, correctness and semantics of the application depends heavily on the timeliness of the interactions among different system components. We focus on timing based QoS requirements and their specification in the actor model. Consider the following timing-based QoS properties:

- **End-to-end Delay (EED):** represents the sum total of the delays experienced between the service-provider and end-user . Simplistically, it can be stated as the difference in time between the sending of a message at the server sv (t^{sv}_{send}) and arrival of a message at a client cl (t^{cl}_{arr}) , assuming that the sender and receiver clocks are synchronized.

$$EED = t^{cl}_{arr} - t^{sv}_{send} \tag{1}$$

- **Jitter:** quantifies the perceived deviation from expected bit-rate, and measures the expected inter-arrival times of the frames in a stream. It is defined as the variation in delays experienced by two consecutive packets of a session transmitted across a network. Violation of jitter bounds portrays itself as flickering video, distorted audio, blanks and frozen images. The jitter j^i is defined as

$$j^i = (\delta^i - \delta^{i-1}) \tag{2}$$

  where δ^i is the difference between arrival times of packets i and $i+1$; δ^{i-1} is the difference between arrival times of packet $i-1$ and i.
- **Synchronization Skew:** quantifies the skew i.e. the difference in arrival rates between 2 multimedia streams, for example audio and video. Let $t^{i,a}$ and $t^{i,v}$ be time points of two packets from different media which should be synchronized in the same time interval. The synchronization skew σ is defined as:

$$\sigma = t^{i,a} - t^{i,v} \tag{3}$$

For acceptable QoS in a distributed multimedia system, these parameters are bounded.

1.1 Actors and Multimedia

A drawback with the traditional method of QoS specification is that the specification of QoS requirements is intermixed with the service specification [6, 7]. 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. 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 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. 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 be formalized [2, 11].

Specifying QoS in the Actor based model is essentially a problem of specifying coordination constraints between distributed objects. Synchronizers [5] allow us to express these coordination constraints, i.e. local synchronization constraints or multi-actor coordination constraints [4]. Synchronizers allow us to qualitatively control the semantics of message delivery. 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]. Objects in this system are defined using a real-time variant of the Actor model. A high-level programming language construct called *RTsynchronizer*, specifies a collection of temporal constraints between actors. This approach helps separate what an object does from when it does it and facilitates the ability to dynamically modify real-time constraints.

1.2 QoS Synchronizers

The structure of a unified system is depicted in Figure 1. We can visualize MM applications as a collection of autonomous, concurrent information processing actors called *media-actors* involved in a multimedia session.

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. To incorporate the notion of a session in our model, we propose a special entity called the *QoS synchronizer* that manages QoS constraints and specifications, and verifies invariants between a group of

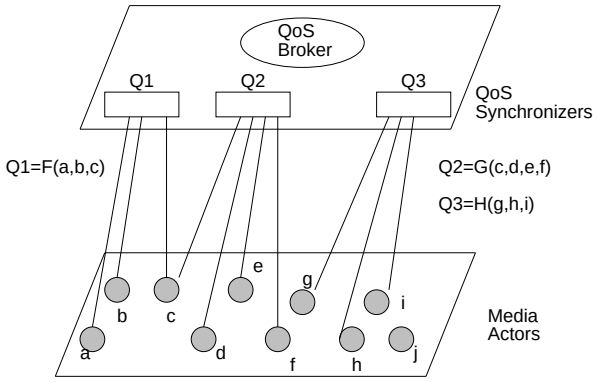

Figure 1 Separation of concerns - QoS requirements specified by system level QoS actors and application functionality encapsulated in media-actors.

media-actors. *QoS synchronizers* are special actors with QoS attributes, e.g. synchronization or timing-related information. Using the concept of QoS Synchronizers permits the modularization of QoS requirements and encourages separation of concerns. 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 gives us the ability to modify one without impacting the other. A QoS synchronizer can observe transitions on the set of media-actors it controls. It cannot send messages to or receive messages from media-actors but can affect their execution.

In general, a multimedia session consists of

- a collection of actors (media-actors) that are involved in that session (components at the server and client ends),
- a session synchronizer that holds and enforces QoS constraints between media-actors in a session.
- a notion of time that is uniform among media-actors of that session.

For the sake of simplicity, we assume that there is only one ongoing session in the system at a time. In a system with multiple sessions, we need to address the issue of satisfying QoS constraints for each session. For this, we propose an agent called the *QoS broker* [8] that acts as a coordinator for all the ongoing sessions and performs admission control for new incoming sessions. Design criteria for an actor-based QoS broker, its components and interactions in a distributed multimedia environment is beyond the scope of this paper and is a future area of research [12].

We define a QoS Synchronizer `QosSync` encoded in RTsynchronizer syntax to express timing-based QoS requirements by an example illustrated in Figure 2. An RTsynchronizer specification consists of

- a declaration of state variables which may be changed by observing certain events on the controlled media-actors;
- a set of constraints that need to be enforced; and
- rules for state change.

In this example, we consider a multimedia session with 2 streams of information. The streams have the following QoS requirements: (1) Jitter less than `maxJitter` (2) End-to-end delay `EED` less than `maxEED`; (3) Interstream synchronization skew less than `maxSkew`. These constraint values may be passed in as parameters to the RTsynchronizer, for example, the desired synchronization skew may be 80 milliseconds.
The group of media-actors which `QosSync` controls are specified in the formals. As shown in Figure 2, the **Declare** section defines and initializes the local state variables, such as the array indices, `ind1, ind2`, arrival times `arrtime1, arrtime2`, etc.. The **Constrain** section specifies a set of QoS constraints, which is of the form `condition : action`. The statements in the **Constrain** section ensure that messages for which `EED >= maxEED` or `jitter >= maxJitter` or `skew >= maxSkew` are never delivered. This infinite delay is specified by the timing condition `(inf, inf)` and is logically equivalent to dropping the packet. We will further explain this notation and its formal semantics in Section 2.2.

The **Update** section defines how the `QosSync` changes its state upon observing certain events which are defined by message patterns. Here, `stream1ptrn` is the pattern that identifies the stream corresponding to mediaActor1 (e.g. a videostream) and `stream2ptrn` is the pattern that identifies the stream corresponding to mediaActor2 (e.g. the associated audiostream).

When a packet arrives at one of the media-actors, local state variables are updated, e.g., `arrtime1` is updated to be the current time. The last statement indicates that under any condition, `skew` is defined as `arrtime1[i] - arrtime2[i]`, the difference in arrival time between two packets corresponding to the two different streams, specified as `true:arrtime1[i] - arrtime2[i]`. Structures describing the RTsynchronizer, the distribution of state information between an RTsynchronizer and its domain of control and strategies for maintaining interobject consistency are described in [10].

2 FORMALIZING MM SESSIONS

The basic Actor model captures the fundamental properties of general purpose distributed computing in which only logical time is concerned. Individ-

```
RTsynchronizer QoSSync( actor: Mediaactor1, Mediaactor2;
              real: maxEED, maxJitter, maxSkew) {

 Declare
  int count;
  int ind = ind1 = ind2 = 0; /* array indices */
  real arrtime1= arrtime2 = 0;
  ...
  /* declaration of other local variables */

 Constrain
  true:((*,*),(msg),(EED >= maxEED))(inf,inf);
         /* DropPacket if EED requirement is not satisfied */
  true:((*,*),(msg),(jitter >= maxJitter))(inf,inf);
         /* DropPacket if Jitter requirement is not satisfied */
  true:((*,*),(msg),(skew >= maxSkew))(inf,inf);
         /* DropPacket if skew requirement is not satisfied */


 Update
 stream1ptrn: /* stream corresponding to Mediaactor1 */
              arrtime1[ind1++] = currentTime;
              sendtime1[ind1++] = sendTime;
              /** defining End-to-end delay */
              EED1[ind1]         = sendtime1[ind1] - arrtime1[ind1];
              /** defining jitter */
              if (time11 == NULL) time11 = currentTime;
               else if (time12 == NULL) time12 = currentTime;
                 else if (time13 == NULL)
                    {
                      time13  = currentTime;
                      delta11 = time12 - time11;
                      delta12 = time13 - time12;
                      jitter1 = mod(delta12-delta11);
                      time11 = time12; time12 = time13; time13 = NULL;
                    }

 stream2ptrn: /* stream corresponding to Mediaactor2 */
          /* code similar to stream1 */


 true: /* defining sync skew between stream1 and stream2 */
      skew = mod(arrtime1[ind]-arrtime2[ind]);
```

Figure 2 Declaring a QoS Service using RTsynchronizer

ual 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. Extensions to the actor model, e.g. RTsynchronizers, allow us to specify time related QoS constraints modularly and verify the correctness of constraint satisfaction. In this section, we first state the semantics of RTsynchronizer. We then apply these semantics to a session-based multimedia system and provide a mapping for the semantic constructs. Finally, we informally reason about the correctness of this mapping.

2.1 RTsynchronizer Semantics

RTsynchronizer semantics is based on the ART model, which is an extension of the Actor model. The ART model takes the quantitative real-time aspect into consideration. In particular, statically specified constraints (which are encapsulated by RTsynchronizers), and dynamically created *expectation instance*s on message invocations become the new basic integral components of ART systems configuration.

RTsynchronizers are special actors which may be in one of the following two different states:

rs	::=	$\otimes$	unenforced
	|	(s)	enforced with state s

Figure 3 Two Possible RTsynchronizer States

Unlike ordinary computational actors, RTsynchronizers may not send or receive messages. The functionality of RTsynchronizers is to impose timing constraints on message invocations. An RTsynchronizer's control may be turned off by an actor performing `unenforce` operation. Thereafter, the RTsynchronizer reaches `unenforced` state $\otimes$. Before a message is invoked at its target actor, the controlling RTsynchronizer must evaluate the validity of invocation of the message. The state in which RTsynchronizers evaluate the constraints c is called `evaluating` state and denoted as (s). Another component in an ART system configuration is the *expectation instance set* (ς), which is a collection of expectation instances. An expectation instance is defined as follows:

Definition 1 (Expectation Instance)
The structured unit $s : p \triangleright \delta$ is called an expectation instance, where p is a message pattern defined in the enclosing RTsynchronizer s; δ specifies the time interval during which a message of pattern p is expected to be invoked.

δ is of the form $[d1, d2]$, where $d2 \geq d1 \geq 0$, denoting the starting and ending time points.

Formally, an ART configuration is a structured entity representing the system state at a given time instance. Such time instance is global with respect to the configuration. An ART system configuration is defined as:

Definition 2 (ART Configuration)

$$\langle\!\langle\ \alpha,\ (\tau)_{\mathtt{Timer}}\ \mid\ \mu\ \mid\ \langle\ \sigma \parallel \varsigma\ \rangle\ \rangle\!\rangle$$

where

- α is an actor map, which maps a finite set of actor names to their behavior;
- `Timer` is a special actor whose state ($\tau \in \Re^+$) indicates the current global time with respect to the configuration when the snapshot is taken.
- μ is a finite multi-set of messages, yet to be processed;
- σ is an RTsynchronizer map, which maps a finite set of RTsynchronizer names to their states;
- ς is a multi-set of expectation instances on message invocations needed to be satisfied.

Before we present operational semantics for ART systems, we first define a group of notations and a group of functions based on these notations. These definitions will simplify further explanations and future formulae.

Definition 3 (Notations)

- $\mathcal{AN}$: the set of all possible actor names
- $\mathcal{AS}$: the set of all possible actor states, which contains the following information:
  - execution state (*unknown*, *idle* or *busy*)
  - the values of its state variables
  - the current expression it is reducing if in *busy* state
- $\mathcal{RN}$: the set of all possible RTsynchronizer names
- $\mathcal{RS}$: the set of all possible RTsynchronizer states, which contains the following information:
  - the enforcing state (unenforced, idle or validating)
  - the values of its state variables (including *ctime*())
  - the static constraint expressions

- the dynamic expectation instances
- the current constraint expression it is validating (i.e. in `validating` state).

- Π: the set of all possible message patterns
- $\mathcal{M}$: the set of all possible messages
- $\mathcal{D}$: the set of all possible time intervals
- $\mathcal{E}$: the set of all possible expectation instances

Definition 4 (Functions)

- α, a map which maps actor names to actor states.
  $\alpha: \mathcal{AN} \rightarrow \mathcal{AS}$
- σ, a map which maps RTsynchronizer names to RTsynchronizer states.
  $\sigma: \mathcal{RN} \rightarrow \mathcal{RS}$
- Sat_σ, a boolean function which returns T if message m satisfies pattern p under the enclosing RTsynchronizer s.
  $Sat_\sigma: \mathcal{M} \times \Pi \times \mathcal{RN} \rightarrow \{\mathtt{T}, \mathtt{F}\}$

$$Sat_\sigma(m,p,s) = \begin{cases} \mathtt{T} & \text{if the evaluation of pattern matching returns T} \\ \mathtt{F} & \text{if the evaluation of pattern matching returns F} \end{cases}$$

- $Safe_\sigma$, a boolean function which returns T if message m is safe for invoking at time τ with respect to an expectation instance $(s : p \rhd [t, t1])$.
  $Safe_\sigma: \mathcal{E} \times \mathcal{M} \times \Re_{\geq 0} \rightarrow \{\mathtt{T}, \mathtt{F}\}$

$$Safe_\sigma((s : p \rhd [t,t1]), m, \tau) = \begin{cases} \mathtt{F} & \text{if } (\sigma(s) \neq \otimes) \ \wedge (Sat_\sigma(m,p,s) = \mathtt{T}) \wedge \\ & \quad ((\tau < t) \vee (\tau > t1)) \\ \mathtt{T} & \text{otherwise} \end{cases}$$

- Dlv_σ, a function which returns T if message is deliverable with respect to the current expectation instance set.
  $Dlv_\sigma : \mathcal{M} \times 2^{\mathcal{E}} \times \Re_{\geq 0} \rightarrow \{\mathtt{T}, \mathtt{F}\}$

$$Dlv_\sigma(m, \varsigma, \tau) = \begin{cases} \mathtt{T} & \text{if } (\forall (s : p \rhd [t1,t2]) \in \varsigma \ (Sat_\sigma(m,p,s) = \mathtt{F})) \vee \\ & \quad (\exists (s : p \rhd [t1,t2]) \in \varsigma \ ((Sat_\sigma(m,p,s) = \mathtt{T}) \wedge \\ & \quad\quad (Safe_\sigma((s : p \rhd [t1,t2]), m, \tau) = \mathtt{T}))) \\ \mathtt{F} & \text{if } (\exists (s : p \rhd [t1,t2]) \in \varsigma \ (Sat_\sigma(m,p,s) = \mathtt{T}) \wedge \\ & \quad \forall (s : p \rhd [t1,t2]) \in \varsigma \ (((Sat_\sigma(m,p,s) = \mathtt{F}) \vee \\ & \quad\quad (Safe_\sigma((s : p \rhd [t1,t2]), m, \tau) = \mathtt{F}))) \end{cases}$$

- $NewEI_\sigma$, a function which returns a set of new expectation instances cre-

ated by invoking a message. *
$NewEI_\sigma$: $\mathcal{M} \times \Re_{\geq 0} \rightarrow 2^{\mathcal{E}}$
$NewEI_\sigma(m, \tau) = \cup_{s \in \mathrm{Dom}(\sigma)} \{(s : p' \ \triangleright [t1 + \tau, t2 + \tau]) \mid (Sat_\sigma(m, p, s) = \mathtt{T}) \wedge ((p : p' \ [t1, t2]) \text{ is in } \sigma(s))\ \}$

- *RemEI*, a function which returns a set of expectation instances satisfiable by invoking a message.
  $RemEI$: $\mathcal{M} \times 2^{\mathcal{E}} \times \Re_{\geq 0} \rightarrow 2^{\mathcal{E}}$

  $RemEI_\sigma(m, \varsigma, \tau) = \{e \mid (e \in \varsigma) \ \wedge \ (Safe_\sigma(e, m, \tau) = \mathtt{T})\}$

- *Update*, a function which changes the RTsynchronizer to new state.
  $Update$: $\mathcal{M} \times \Re_{\geq 0} \times \mathcal{RN} \rightarrow \mathcal{RS}$

$$Update_\sigma(m, \tau, s) = \begin{cases} newState & \text{if } \exists p Sat_\sigma(m, p, s), \text{and} \\ & p \text{ is an update pattern in } s \\ \sigma(s) & \text{otherwise} \end{cases}$$

The standard operational semantics of ART systems is build on top of the basic actor semantics [3] with two extensions: (1) there is a special actor `Timer` whose state τ represents the current time in the system, (2) constraints are imposed on the invocation of messages.

In the standard operational semantics for actors there are two kinds of transitions: execution steps local to an individual actor and those dealing with message receipt. Execution transitions not affect other objects states in the system, — thereby the RTsynchronizer map (σ) and the expectation set (ς) remain the same as before and after the transitions. We skip the discussion of these transition rules in this paper *.

However, different from its corresponding part in the standard actor semantics, the meaning of transition rule <`rcv` : m> may be 3-fold:

- The prerequisite for taking the transition at the current configuration is that the message m is deliverable, namely the function $Dlv_\sigma(m, \varsigma, \tau)$ must evaluate to `true`.
- The same as the standard `rcv` rule, the receiving actor binds the actual parameters carried in the message to its formal parameters and applies its behavior to the message contents, i.e., carry out the underlying computation.
- If the message m is captured by a pattern in an RTsynchronizer, the invocation of such message may cause:

---

*No need to do the `safe` testing because the function `NewEI` is called only when the message is safe to be delivered at the destination. The same reason applies for `RemEI` and `Update` functions

*The details of these transition rules can be found in [10]

- RTsynchronizers state change, formulated as $Update_\sigma(m, \tau, \sigma)$;
- If m is an expected message, invoking it will satisfy an expectation instance in the system; hence, a satisfied expectation instance will be removed from the expectation instance set $(RemEI_\sigma(m, \varsigma, \tau))$;
- the message m may also be a "causal" message, thereby may create new expectation instances. Hence new expectation instances $(NewEI_\sigma(m, \tau))$ are added into the expectation set.

Because constraints are added into the ART systems, there exists the case that the system fails to satisfy the required constraints and causes system failure. In addition, time as an independent entity progresses concurrently with the underlying system activities. Two new transitions are added into ART systems to express system failure and time progression. The short version of ART system operational semantics using transition rules are given in Figure 4. The detailed semantics can be found in citeren-phd.

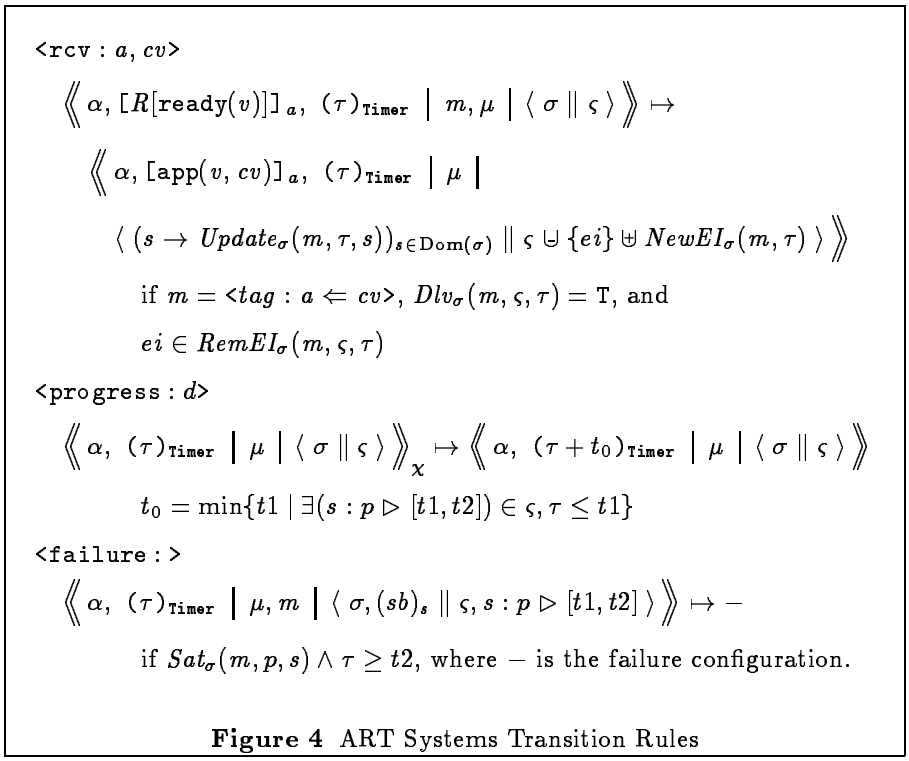

Figure 4 ART Systems Transition Rules

We formally define a session configuration as a structured entity representing the system state at a given time instance similar to an ART configuration. The time actor in this case gives us a virtual time with respect to the ongoing

session in the system. Formally, a session configuration is mapped from an ART configuration as:

Definition 5 (SessionConfiguration)

$$\langle\!\langle\ \alpha,\ (\tau)_{\texttt{Session}}\ \mid\ \mu\ \mid\ \langle\ \sigma_s \parallel \varsigma\ \rangle\ \rangle\!\rangle$$

where

- α is an actor map that represents the media-actors in the session
- $\tau_{Session}$ is a session based virtual time, which is a special case of global time τ and
- σ_s is a mapping from a single QoS synchronizer to its state $(S)_{QoS}$. The QoS synchronizer encodes the required QoS constraints.

We apply the same mapping to the rest of the RTsynchronizer semantics to obtain the session based QoS Synchronizer semantics.

2.2 The Correctness of Session Based Semantics

In this section, we provide informal reasoning that shows that the RTsynchronizer semantics for the QoS specification provides the desired behavior, i.e satisfies the desired time related QoS requirements. We need to show that the transition rules described in the previous section implement the specified QoS requirements.

Consider the example discussed in Figure 2. There are two media streams both of which obey: (a) EED constraints (b) Jitter constraints and (c) synchronization skew constraints. Constraints in MM systems can be specified using two different formulations:

- `A : B[t1,t2]`: In this representation, delivery of a message satisfying pattern `A` at `t0` creates an expectation instance `B[t0+t1, t0+t2]` indicating that a message of pattern `B` must be delivered in the interval `[t0+t1, t0+t2]`.
- `true: B[t1,t2]`: In this representation, there is always an expectation instance `B[t1,t2]` i.e. a message of pattern `B` that must be delivered in the time interval `[t1, t2]`. Specifically if `t1` and `t2` are `inf`, i.e `B[inf, inf]`, the message delivery is postponed indefinitely.

We use the second format to encode the jitter, EED and synchronization skew constraints by stating that if, for any message to be processed, delivering this message causes the EED, jitter and skew values to be greater than `maxEED`,

`maxJitter` or `maxSkew` respectively, the message is indefinitely postponed (i.e. logically dropped).

In general, the arrival of a message causes the receive transition <`rcv` : m> to be triggered. By the transition semantics discussed in Section 2.1, the packet is considered deliverable only when the delivery clause Dlv_{σ_s} evaluates to true. σ_s is an encoding of QoS constraints within a media actor, hence jitter and end-to-end delay constraints must be satisfied before the message is considered deliverable. i.e. `EED` and `jitter` are calculated for the incoming packet. If the value of `EED` is greater than the maxEED value specified in the constraint, or the jitter value is greater than maxJitter, that message packet is indefinitely postponed(i.e dropped).

σ_s also encodes synchronization skew constraints between the two media-actors controlled by the QoS synchronizer. Since both actors share the same notion of time ($\tau_{session}$), synchronization skew is obtained by measuring the interarrival times of messages to the two actors in the configuration. Hence the delivery of a message at one actor can occur only if the corresponding message to the other media-actor has also been received within a time period specified by the synchronization skew constraint `maxSkew`. Since this is also a clause for the satisfaction of the Dlv_{σ_s} constraint, inter-actor constraints must also be met before the messages are considered deliverable.

Compositionality of the session configurations described can be achieved by synchronizing the session virtual times $\tau_{session}$ of the composing configurations. Let

$$K_1 = \left\langle\!\!\left\langle\ \alpha_1,\ (\tau)_{\texttt{Session1}}\ \middle|\ \mu_1\ \middle|\ \langle\ \sigma_{s1} \parallel \varsigma_1\ \rangle\ \right\rangle\!\!\right\rangle$$

and

$$K_2 = \left\langle\!\!\left\langle\ \alpha_2,\ (\tau)_{\texttt{Session2}}\ \middle|\ \mu_2\ \middle|\ \langle\ \sigma_{s2} \parallel \varsigma_2\ \rangle\ \right\rangle\!\!\right\rangle$$

be two independent session configurations. the composition of these 2 sessions $K_1||K_2$ is defined as:

$$K_1||K_2 \ \textit{iff}\ \tau_{\texttt{Session1}} = \tau_{\texttt{Session2}}$$

3 CONCLUSIONS

We have proposed a formal specification of timing related QoS attributes in an actor-based distributed system and described some techniques for informally reasoning about quantitative QoS properties. When two QoS synchronizers operate on the same media actor, there may be inconsistencies that must be resolved. We are developing multisession semantics to reason about the behavior of an actor based realtime multimedia system. In practice, multiple system and application activities will need to occur concurrently, for example, scheduling, monitoring, inter and intra stream synchronization all of which may need to satisfy QoS constraints. One implementation of QoS synchronizers is through a meta-architecture framework [13]. 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 addressing the composability of QoS constraints in this 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] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7:1–72, 1997.

[4] S. Frølund and G. Agha. A language framework for multi-object coordination. In *Proceedings of ECOOP 1993*, volume 707 of *Lecture Notes in Computer Science*. Springer Verlag, 1993.

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

[6] 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.

[7] 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.

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

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

[13] N. Venkatasubramanian and C. L. Talcott. Reasoning about Meta Level Activities in Open Distributed Systems. In *Principles of Distributed Computation*, 1995.

BIOGRAPHICAL SKETCH

Shangping Ren is a doctoral candidate and research assistant in the Open Systems Laboratory at the University of Illinois at Urbana-Champaign. Her research interests include programming languages and parallel distributed real-time computing. She received her BS and MS from Department of Computer Engineering at Hefei Polytechnic University, China.

Nalini Venkatasubramanian is currently a doctoral candidate at the Open Systems Laboratory at the University of Illinois at Urbana-Champaign. She is also a member of technical staff working on interactive multimedia at Hewlett-Packard Laboratories in Palo Alto, California. Her research interests include multimedia and real-time systems, parallel and distributed technology, resource management, distributed operating systems and concurrent object programming languages. She has been working on software architectures for multimedia servers, multimedia networking protocols, real-time issues and load-balancing on distributed multimedia servers. Prior to this she has worked on various database management systems and on programming languages/compilers for high performance machines. She received an MS in Computer Science from the University of Illinois, Urbana-Champaign in 1992.

Gul Agha is Director of the Open Systems Laboratory at the University of Illinois at Urbana-Champaign. His research interests include models, languages, and tools for computing in open distributed systems. His book, *Actors: A Model of Concurrent Computing in Distributed Systems* stimulated considerable research in concurrent objects. Professor Agha is an ACM International Lecturer, Editor-in-Chief of *IEEE Concurrency*, and Associate Editor of *Theory and Practice of Object Systems* and *ACM Computing Surveys*. He is a recipient of the Incentives for Excellence Award from Digital Equipment Corporation, Naval Young Investigator Award from the US Office of Naval Research, and of a Fellowship at the University of Illinois Center for Advanced Study. Agha has served as Consulting Scientist to Microelectronics and Computer Technology Consortium and as a Visiting Professor at the University of Grenoble, France.