A Formal Model for Reasoning about Adaptive QoS-Enabled Middleware

Nalini Venkatasubramanian[1], Carolyn Talcott[2], and Gul Agha[3]

[1] Univ. of California Irvine, Irvine, CA 92697-3425, USA,
nalini@ics.uci.edu
[2] Stanford University, Stanford, CA 94305, USA,
clt@cs.stanford.edu
[3] University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA,
agha@cs.uiuc.edu

Abstract. Systems that provide QoS-enabled services such as multimedia are subject to constant evolution - customizable middleware is required to effectively manage this change. Middleware services for resource management such as scheduling, protocols providing security and reliability, load balancing and stream synchronization, execute concurrently with each other and with application activities and can therefore potentially interfere with each other. To ensure cost-effective QoS in distributed systems, safe composability of resource management services is essential. In this paper we present a meta-architectural framework for customizable QoS-based middleware based on the actor model of concurrent active objects. Using TLAM, a semantic model for specifying and reasoning about components of open distributed systems, we show how a QoS brokerage service can be used to coordinate multimedia resource management services in a safe, flexible and efficient manner. In particular, we show that a system in which the multimedia actor behaviors satisfy the specified requirements, provides the required multimedia service. The behavior specification leaves open the possibility of a variety of algorithms for resource management as well as adding additional resource management activities by providing constraints to ensure their non-interference. *Keywords: meta-object models, distributed systems, theoretical foundations, object-oriented applications, multimedia*

1 Introduction

In the coming years, QoS-enabled distributed servers will be deployed to deliver a variety of interactive services. Applications such as telemedicine, distance learning and electronic commerce exhibit varying requirements such as timeliness, security, reliability and availability. The set of servers, clients, user requirements, network and system conditions, in a wide area infrastructure are changing continuously. Future applications will require dynamic invocation and revocation of services distributed in the network without violating QoS constraints of ongoing applications. To assure safe adaptation to dynamically changing requirements it is important to have a rigorous semantic model of the system: the resources, middleware that provides system management, the application activities, and the sharing and interactions among these. Using such a model, designs can be analyzed to clarify assumptions that must be met for correct operation, and to establish criteria for non-interference. In [29, 25], we presented the TLAM(Two Level Actor Machine) semantic framework for specifying, composing and reasoning about resource management services in open distributed systems. The TLAM is based on the actor model of computation, a natural model for dynamic object-based distributed systems. In the TLAM, a system is composed of two kinds of actors (active objects), base actors and meta actors, distributed over a network of processing nodes. *Base-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. The two levels provide a clean separation of concerns and a natural basis for modeling and reasoning about customizable middleware and its integration with application activity. Based on the two-level architecture, a customizable and safe distributed systems middleware infrastructure, called $\mathbf{CompOSE|Q}$ (Composable Open Software Environment with QoS) [26] is being developed at the University of California, Irvine, that has the ability to provide cost-effective and safe QoS-based distributed resource management.

Our general approach to modeling systems using a TLAM framework is to develop a family of specifications at from different points of view and at different levels of abstraction. We begin with the abstract notion of end-to-end service provided by a system in response to a request. This high-level view can be refined (here we use the word informally) by expressing system wide properties in terms of of abstract properties of the underlying network. Another point of view specifies constraints on the behavior and distribution of a group of actors. This local behavior point of view can be further refined by specifying protocols and algorithms for the actions of individual actors. The two points of view are related by the notion of a group of meta actors providing a service, in a system satisfying suitable initialization and non-interference conditions. The staging and refinement of specifications provides a form of modularity, scalability, and reusability by reducing the task of implementation to that of implementing individual abstract behaviors. Behavior level specifications can be used to guide or check implementations or even serve as executable prototypes.

In previous applications of the TLAM [28, 29] we have focused on defining core services such as remote creation and snapshots, understanding their potential interactions, and composing services built up such core services. Of particular concern were constraints on meta-level behavior needed to maintain a consistent view of the actor acquaintance topology. In this paper, we use the TLAM framework to develop a formal model for customizable, cost-effective middleware to enforce QoS requirements in multimedia applications. Here we explicitly model resources of the network infrastructure and constraints on the proper management of these resources. We begin by informally describing the notion of a system providing *QoS-based MM Service*.We then map QoS requirements to resource requirements and focus on modeling and reasoning about the resource management underlying a QoS-based service. For this purpose we define, in a rigorous manner, the notions of a system providing *Resource-based MM Service*, of a system having *Resource-based MM Behavior*, and finally refining the system with an *Adaptive Request Scheduling Policy*. The Resource-based MM Service specification reflects the chosen system resource architecture and allows us to reason about the availability and use of resources. The Resource-based MM Behavior specification models the QoS broker software architecture presented in [25, 27] and places constraints on the actions of the QoS meta actors. Such a behavior specification can serve as a first stage in refining a service specification into an implementation. The *Adaptive Request Scheduling Policy* illustrates such refinement. It specifies one of the resource management policies developed in [25] by giving high-level algorithms for determining meta actor behavior. The main results are:

(1) if a system provides Resource-based MM Service, then (under the assumptions on the mapping from QoS requirements to Resource requirements) it provides QoS-based MM Service;
(2) if a system has Resource-based MM Behavior, and meets certain initialization and non-interference conditions, it provides Resource-based MM Service;
(3) if a system is refined with the Adaptive Request Scheduling Policy, then it implements Resource-based MM Behavior.

A consequence of (2) is that new broker policies can be safely installed as long as they satisfy the behavior constraints. (3) is an example of such a policy.

The rest of this paper is organized as follows. Section 2 reviews the TLAM framework concepts needed to understand the formal model of the QoS Broker. Section 3 recalls the QoS Broker multimedia server meta-architecture of [25]: its physical and software architectures, and a mapping of the software architecture onto the physical architecture. Section 4 describes several resource management policies that we have used in realizing the software architecture. Section 5 shows how we use the TLAM framework to formally model and reason about systems such as QoS Broker. Section 6 discusses related work and outlines areas for future research.

2 The Two Level MetaArchitecture

In this section we briefly summarize the main concepts of the TLAM semantic framework. More detail can be found in [29, 25]. The Actor model is a model of concurrent active objects that has a built-in notion of encapsulation and interaction and is thus well-suited to represent evolution and co-ordination among interacting components in distributed multimedia applications. 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 interacts only by sending and receiving messages. (See $[1,2]$ for more discussion of the actor model, and for many examples of programming with actors.)

As mentioned earlier, in a TLAM model, a system is composed of two kinds of actors, base actors and meta actors, distributed over a network of processing nodes. *Base-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. A TLAM model provides an abstract characterization of actor identity, state, messages, and computation, and of the connection between base- and meta-level computation. Meta-actors communicate with each other via message passing as do base-actors, and meta-actors may also examine and modify the state of the base actors located on the same node. Base-level actors and messages have associated runtime annotations that can be set and read by meta actors, but are invisible to base-level computation. Actions which result in a change of base-level state are called events. Meta-actors may react to events occurring on their node.

A TLAM is a structure of the form

$$TLAM = \langle Net, TLAS, loc \rangle$$

where Net is the underlying computer network with processor nodes and communication links; and $TLAS$ is a two-level actor system distributed over the network by the map, loc. The semantics of a TLAM model is given by a labeled transition relation on configurations. A TLAM *configuration*, C, has sets of base- and meta-level actors, and a set of undelivered messages. Each actor has a unique name (address) and the configuration associates a current state $getS(C, a)$ to each actor name, a. $Cast(C)$ is the set of names of actors in C. $getA(C, a, t)$ is the value, in C, of the annotation of base-actor a with tag t; $setA(C, a, t, v)$ sets the value of the annotation of a with tag t, returning the updated configuration. Thus $getA(setA(C, a, t, v), a, t) = v$. The undelivered messages are distributed over the network – some are traveling along communication links and others are held in node buffers. There are two kinds of transition: communication and execution. Communication transitions move undelivered messages from node buffers to links and links to node buffers and are the same in every TLAM. An execution transition consists of a computation step taken by a base- or meta-level actor, by applying an enabled *step rule*, followed by application of all enabled *event handling rules* (in some order). *Step rules* specify messages delivered to and sent by the stepping actor, change of the stepping actors state, and possibly newly created actors. In addition meta-level step rules may specify changes in the state of base-level actors, base-level messages to be sent, and base-level actors to be created. A step that delivers or sends base-level messages, changes base-level state, or creates new base-level actors signals corresponding events. *Event handling rules* specify the response of meta-level actors to signalled events. Note that no message is delivered and the only base-level modifications that can be specified are annotation modifications. In particular no events are signaled. All actors modified or created in an execution transition reside on the node of the stepping actor.

A *computation path* for initial configuration C_0 is an infinite sequence of labeled transitions:

$$\pi = [\ C_i \xrightarrow{l_i} C_{i+1}\ \Big|\ i \in \mathbf{Nat}\ ]$$

where l_i is the transition, indicating the transition rule applied. The semantics of a configuration is the set of fair computation paths starting with that configuration, where fairness means that any enabled communication transition will eventually happen, and enabled reaction rules will either fire or become permanently disabled.

A *system* is a set of configurations closed under the transition relation. We say that π is a computation path of a system S if it is a computation path with initial configuration some configuration C of S. Properties of a system modeled in the TLAM are specified as properties of computation paths. A property can be a simple invariant that must hold for all configurations of a path, a requirement that a configuration satisfying some condition eventually arise, or a requirement involving the transitions themselves. Properties are checked using the properties of the building blocks for configurations – message contents and actor state descriptions – and of the TLAM reaction rules that determine the behavior of actors in the system.

3 The QoS Broker Meta-Architecture for QoS-based Services

Using the TLAM framework, we develop a meta-architectural model of a multimedia server that provides QoS based services to applications. The physical architecture of the MM server consists of:

- a set of data sources (DS) that provide high bandwidth streaming MM access to multiple clients. Each independent data source includes high capacity storage devices (e.g. hard-disks), a processor, buffer memory, and high-speed network interfaces for real-time multimedia retrieval and transmission;
- a specific node designated as the distribution controller (DC) that coordinates the execution of requests on the data sources; and
- a tertiary storage server that contains the MM objects—replicas of these MM objects are placed on the data source nodes.

All the above components are interconnected via an external distribution network that also transports multimedia information to clients.

The software architecture of a multimedia server consists of two subsystems - the base level and meta-level subsystems corresponding to the application and system level resource management components respectively. The base level component represents the functionality of the MM application: *replica actors* models both MM data objects and their replicas (e.g. video and audio files), and *request actors* model MM requests to access this data. The meta-level component deals with the coordination of multiple requests, sharing of existing resources among multiple requests, and ensuring that the resources needed by requests being serviced at any given time do not exceed the system capacity. To provide coordination at the highest level we introduce, the *QoS Broker* meta-actor, QB. The two main functions of the QoS Broker are data management and 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 replicas of data actors are no longer needed and can be dereplicated. The *request management* component performs the task of admission control for incoming requests, i.e., it decides whether or not a request can be admitted for service. It must ensure the satisfaction of QoS constraints for requests that are ongoing in the system. The multimedia data and request management functions of the QoS broker in turn require a number of services. The organization of the QoS Broker services is shown in Figure 1. Each of these services in turn can be based on one or more of the core services integrated into the metaarchitecture - remote creation, distributed snapshot and directory services.

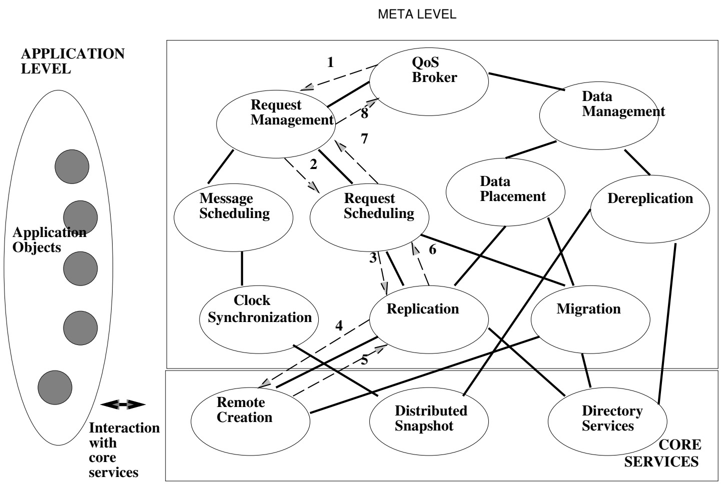

Fig. 1. Detailed architecture of the QoS meta-architecture system.

We model both *adaptive* and *predictive* resource management services. An adaptive service makes decisions based on the state of the system at the time a service is requested. A predictive service has a model of expected request arrival times and attempts to arrange system parameters (e.g. replication state) to maximize some measure, e.g. number of requests served. In this paper we restrict attention to the services related to replication and dereplication. Issues such as message synchronization and migration are topics for future work.

•*Replication:* creates copies of media objects on data sources. Issues to consider include where and when to replicate and object. The rate at which replication proceeds also has a direct impact on system performance.

•*Dereplication:* marks some replicas as removable – the objective being to optimize utilization of storage space by removing replicas that are not needed to make room for ones that are. A dereplication service may base its decisions on current load in the system as well as on expected future demands for an object. Dereplication does not immediately remove a marked copy, this can only happen after all requests that are currently being serviced by that copy have completed.

In order to map the QoS meta-architecture to the physical system architecture, we distinguish between local and global components and define interactions between local resource managers on nodes and the global resource management component. The global component includes the QoS broker and associated meta-actors which reside on the distribution controller node. The node local components include, for each DS node:

• a DS meta-actor for load-management on that node. The DS meta-actor contains state information regarding the current state of the node in terms of available resources, replicas, ongoing requests and replication processes etc.

• Request base actors corresponding to the requests assigned to that node.

• Replica base actors that correspond to the replicas of data objects currently available on that node.

4 Resource Management Policies for MM Servers

Apart from the QoS broker, QB, the MM system contains a number of meta-actors whose behaviors are coordinated by QB to provide the resource management services discussed above. In this section, we describe some of the load management policies that have been treated in the formal model. The policies are implemented as meta-level actors and provide a modular and integrated approach to managing the individual resources of a MM server so as to effectively utilize all of the resources such as disks, CPU, memory and network resources. A MM request specifies a client, one or more multi-media objects, and a required QoS. The QoS requirement in turn is translated into resource allocation requirements. The ability of a data source to support additional requests is dependent not only on the resources that it has available, but also on the MM object requested and the characteristics of the request (e.g., playback rate, resolution). We characterize the degree of loading of a data source DS with respect to request R in terms of its load factor, $LF(R, DS)$, as:

$$LF(R, DS) = max(\frac{DB^R}{DB^{DS}}, \frac{Mem^R}{Mem^{DS}}, \frac{CPU^R}{CPU^{DS}}, \frac{NetBW^R}{NetBW^{DS}})$$

where DB^R, Mem^R, CPU^R, and $NetBW^R$ denote the disk bandwidth, memory buffer space, CPU cycles, and network transfer bandwidth, respectively, that are necessary for supporting request R and similarly Res^{DS} denotes the amount of resource Res available on data source DS. The load factor helps identify the critical resource in a data source, i.e., the resource that limits the capability of the data source to service additional requests. By comparing the load factor values for different servers, load management decisions can be taken by the QoS brokerage service.

Below we briefly describe the QoS meta-actors coordintaed by the QoS broker for scheduling of MM requests and placement of MM data.

Scheduling of Multimedia Requests: The Request Scheduling meta-actor (RS) implements an adaptive scheduling policy that compares the relative utilization of resources at different data sources to generate an assignment of requests to replicas, so as to maximize the number of requests serviced. The data source that contains a copy of the MM object requested and which entails the least overhead (as determined by the computed load factor) is chosen as the candidate source for an incoming request. If no candidate data source can be found for servicing request R, then the meta-actor RS can either reject the incoming request or initiate *replication on demand* - implemented via a replication on demand meta-actor (ROD). The QoS broker also analyzes the rejections over time and triggers appropriate placement policies, implemented via predictive placement and dereplication meta-actors (PP and DR) to reduce the rate of rejection. The replication on demand meta-actor ROD attempts to create a new replica of a requested MM object on the fly. The source DS on which the new replica is to be made is one that has minimum load-factor with respect to the request, R, i.e., with the minimum value of $LF(R, DS)$. By doing so, ROD attempts to maximize the possibility of the QoS broker servicing additional requests from the same replica. In order for this approach to be feasible and attractive, the replication must proceed at a very high rate, thereby consuming vital server resources for replication.

Placement of MM Objects: The predictive placement and dereplication meta-actors (PP and DR) implement a placement policy that determines in advance when, where and how many replicas of each MM object should be placed in a MM server, and when to dereplicate an existing replica. In particular, the goal of the predictive placement procedure is to facilitate the task of the adaptive scheduler meta-actor, by allocating MM objects in such a way as to maximize system-wide revenue, by permitting a maximum number of requests to be admitted and scheduled for service.

Placement mechanisms must be designed to work effectively with request scheduling. For example replication and derpelicatation activities should not simply cancel each others effects. One way of coordiating these activities is to run dereplication before replication. Another potential for interference arises with the concurrent execution of the ROD and PP processes. The current PP process that initiates replica creation is based on a current snapshot of available systems resources, e.g. disk space. Without proper constraints, this snapshot will not account for replicas being created dynamically by the ROD process. A simple solution is to disable ROD when the PP process is initiated. Achieving more concurrency requires more complex coordination and synchronization.

Some Performance Results: Performance studies show that application objects can be managed effectively by composing multiple resource management activities managed at the metalevel [27]. Figure 2 illustrates the performance, measured by request rejection rate, of various policies for load management - (a) purely adaptive (on-the-fly) scheduling and placement(P1), (b) purely predictive (decided a priori) scheduling and placement(P2), (c) composite policies that provide adaptive scheduling and predictive placement(P3 and P4, an optimized version of policy P3). The left hand side graph illustrates the request rejection rate under purely adaptive policies for placement and scheduling. *Startup latency* is a QoS factor that indicates how long the user is willing to wait for a replica to be created adaptively. The graph demonstrates that when the startup latency is below a threshold value (2 min), the purely adaptive mechanisms, represented by P1 force a very large fraction of the requests received to be rejected. Assuming that startup latency is sufficiently large, the right hand side depicts the inadequacy of P2, that relies on only predictive policies for scheduling and placement. In comparison, the other 3 policies (P1, P3 and P4), show hardly any rejects (indicated by the overlapping lines in the graph). As can be observed from the performance results, the ability to run multiple policies simultaneously (as in cases P3 and P4) reduced the total number of rejected requests in the overall system. In this paper, we study complex interactions that can arise due to the simultaneous execution of multiple system policies.

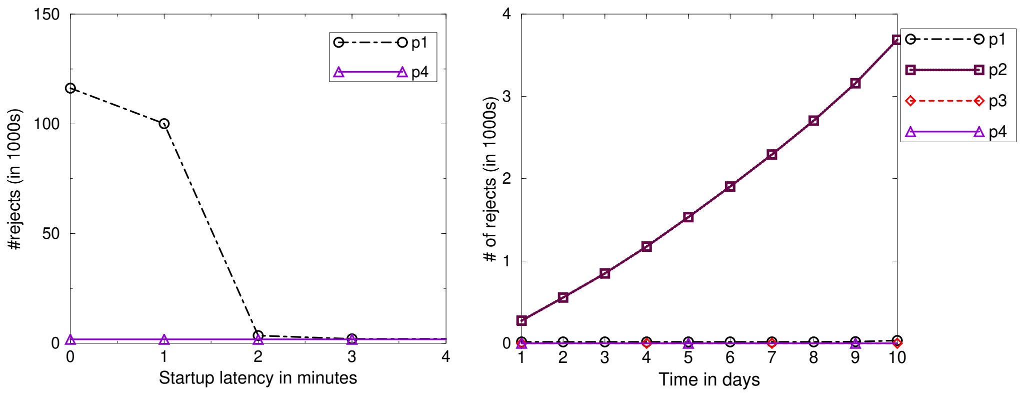

Fig. 2. Comparison of the performance of load management policies for request scheduling and video placement in a distributed video server.

5 Reasoning about QoS-Based MM Services

Assuring safe composability of resource management services is essential for efficient management of distributed systems with widely varying and dynamically changing requirements. In this section we show how the TLAM framework is used to model and reason about the multimedia meta-architecture and resource management policies presented above. Following our basic approach to modeling systems in the TLAM framework we specify the QoS broker from different points of view (end-to-end services and individual behaviors) and at different levels of abstraction and establish theorems relating the different specifications.

In § 5.1 we informally describe the notion of a system providing *QoS-based MM Service*. This is the high-level system wide request based service that the customer sees. In § 5.2 we define the notion of a system providing *Resource-based MM Service*. This reflects the relevant system resources and expresses high-level resource management requirements that must be met in order to provide the QoS-based MM Service. We postulate a function that translates QoS requirements to resource requirements and argue that:

- if a system provides Resource-based MM Service, then under the given assumptions on the mapping from QoS requirements to Resource requirements, the system provides QoS-based MM Service.

In § 5.3 we define the notion of a system having *Resource Based MM Behavior*. This specification reflects the QoS broker software architecture and places constraints on the actions of the QoS meta actors. We define initial and non-interference conditions, and show that

- if a system has Resource-based MM Behavior, then if the initial and non-interference conditions hold, the system provides Resource-based MM Service.

In § 5.4 we refine the behavior by requiring the system to act according to given *Resource Based MM Broker Policies*. Here we focus on one specific policy, the *Adaptive Request Scheduling* Policy. We show that

- if a system acts according to the Resource-based MM Broker Policy, (e.g. Adaptive request Scheduling Policy), then it has Resource-based MM Behavior.

The Resource-based policy specifications include algorithms for request scheduling and replication / dereplication decisions. Thus they constitute a step towards implementation.

5.1 QoS based MM service

We assume that there is a fixed set, $MMObjects$, of MM objects available in the system and let MM range over $MMObjects$. We also assume given a set $MMreqset$ of

MM requests—messages used to request MM service—and let $MMreq$ range over $MMreqset$. A MM request message $MMreq$ determines a triple (α_{cl}, MM, qs). This is interpreted as a request to initiate a MM streaming service from the server receiving the request to the client α_{cl}, using the MM object MM, and obeying the QoS requirement qs.

Definition 1 (QoS-based MM Service). A system S provides a QoS-based MM Service over the set of MM objects, $MMObjects$, and request messages $MMreqset$ iff for every configuration C of S, if there is an undelivered request message $MMreq$ in C, then along any path π from C exactly one of the following properties hold:

(1) there is a unique transition in π where $MMreq$ is accepted for service, and service is provided with the required QoS until complete, or
(2) there is a unique transition in π where $MMreq$ is rejected, and for this to happen it must be the case that the requested QoS cannot be provided at the time that $MMreq$ arrives.

A more advanced QoS-based service that negotiates with the client for lower QoS or delayed service in the case that the request can not be served as presented could be built on top of the simple QoS-based MM Service specified above. That is outside the scope of the current work.

5.2 Specifying a Resource Based MM Service

We assume given a function $QoSTranslate$ that maps MM requests to resource requirements which, if met, will ensure the requested QoS. Thus real-time requirements typical of MM applications, for example required bit-rate of video, are translated into corresponding resource requirements, for example a bandwidth requirement. (See [18] for examples of such QoS translation functions). For the purposes of this specification, we assume that if resource are allocated for a request, then they are used (as needed) to provide the requested QoS.

Definition 2 (Managed Resources). We consider four managed resources: network bandwidth ($NetBW$), CPU cycles (CPU), disk bandwidth (DB), and memory buffer (Mem). We let $Resources$ denote this set of resources and let Res range over $Resources$. We use the notation $Unit_{Res}$ for the units in which we measure the resource Res, and let $QoSTuple = Unit_{DB} \times Unit_{CPU} \times Unit_{NetBW} \times Unit_{Mem}$. For an element qt of $QoSTuple$ we write qt_{Res} to select the component associated to Res.

Definition 3 (QoSTranslate requirements). The function $QoSTranslate$ maps MM requests to 4-tuples representing resource allocation requirements for the four managed resources: $QoSTranslate : MMreqset \rightarrow QoSTuple$. For request $MMreq$, we require that $QoSTranslate(MMreq)$ be such that the QoS requirement of $MMreq$ is met if

(a) the resources allocated to $MMreq$ are at least those specified by $QoSTranslate(MMreq)$ together with access to a copy of the MM object of $MMreq$, and
(b) the allocated resources are continuously available during the service phase for $MMreq$.

Availability means that the MM object replica is not deleted (or overwritten) and that the total allocation never exceeds capacity, since over-allocation implies that resources must be taken from some already admitted request thereby possibly violating the QoS constraints for that request.

MM System Architecture. The physical layer of Section 3 is represented as a TLAM network by mapping nodes in the MM server to TLAM nodes. Recall that there are several kinds of nodes: a set of data source nodes that hold replicas and provide the actual MM streaming, a distribution controller node responsible for coordinating the data source nodes, and a set of client nodes from which MM requests arise. (There are also a tertiary storage nodes that contain the MM objects which we omit here to simplify the presentation.) We let $DSnodes$ be the set of data source nodes and let DS range over $DSnodes$. We assume, given, a function $capacity$ such that $capacity(DS, Res) \in Unit_{Res}$ for any data source node DS and resource Res.

The MM state of the system is modeled by functions characterizing the state of replicas on each data source node and functions characterizing the state of each request that has arrived. The resource based MM service is then specified in terms of constraints on the values of these given functions and the way the values may change, such that if the MM request scheduling and replication processes obey these constraints, then the QoS-based service requirements will be met and the underlying streaming mechanisms will be able to provide the desired QoS-based service. In particular, any policies that entail these constraints can be used to implement the MM service.

Definition 4 (Functions characterizing replicas). There is at most one replica of a given MM object on any DS node, and thus it can be uniquely identified by the node and object. There are three functions characterizing the state of the replica of a MM object MM on a DS node DS in system configuration C:

- $replState(C, DS, MM) \in ReplStates = \{InQueue, InProgress, replCompleted\}$ is the replication status of the multimedia object MM on node DS. $InQueue$ indicates that replication has been requested but not initiated, $InProgress$ indicates that replication is in progress, and $replCompleted$ indicates that replication is complete.
- $replnBW(C, DS, MM) \in Unit_{NetBW}$ is the minimum bandwidth available to complete a replication, meaningful only if replication is in progress or in queue.
- $replClass(C, DS, MM) \in \{0, 1, 2, 3\}$ is the replication class of the multimedia object MM on node DS. Class 0 indicates that the replica is not present on the node. A replica of class 1 is guaranteed to be available. A replica of class 2 is considered marked as dereplicable but remains available until all requests assigned

to it have completed. A replica of class 3 exists on the node in the sense that it has not been overwritten, but there is no guarantee it will remain that way and can not be considered available until its class is changed.

The constraints, $\phi_{\mathrm{repl}}(S)$, on the replica functions require that as the system evolves, the replication state of a replica moves from $InQueue$ to $InProgress$ to $replCompleted$ and that the $replClass$ function satisfies the constraints specified by the transition diagram given in Figure 3. For example the diagram specifies that if $replClass(C, DS, MM) = 0$, and $C \to C'$, then $replClass(C', DS, MM) \in \{0, 1, 3\}$. Also, if $replClass(C, DS, MM) = 2$, and $replClass(C', DS, MM) = 3$, then there are no active requests assigned to this replica.

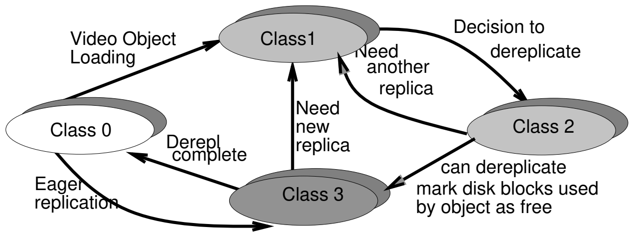

Fig. 3. State transition diagram specifying the allowed changes in the class of a MM object replica. States are labelled by values of the class values and the arrows indicate allowed changes in the value as the system evolves. Class 0 indicates that the replica is not present on the node. A replica of class 1 is guaranteed to be available. A replica of class 2 is considered marked as dereplicable but remains available until all requests assigned to it have completed. The Class2 to Class3 transition is allowed only when there are no active(ongoing) requests assigned to the replica. A replica of class 3 cannot be assigned until it is changed to a 1 replica.

Definition 5 (Functions characterizing requests). Each MM request that has been delivered in a system has a uniquely associated base actor that represents the request during admission control and servicing. We let $ReqActors$ be a subset of the base actor identifiers set aside for association with MM requests and let α^{req} range over $ReqActors$. We assume given, the following functions characterizing requests:

- $reqClientId(C, \alpha^{req})$ – identifies the client making the request.
- $reqObjId(C, \alpha^{req})$ – the MM object requested.
- $reqQoS(C, \alpha^{req})$ – the 4-tuple returned by $QoSTranslate$.
- $reqState(C, \alpha^{req}) \in ReqStates = \{Waiting, Granted, Denied, Servicing, reqCompleted\}$
- $reqReplica(C, \alpha^{req}) \in DSnodes + \{\mathtt{nil}\}$ – the DS node to which the request has been assigned if any.

The constraints, $\phi_{\mathrm{req}}(S)$, on the request functions require that as a system evolves, the values of $reqClientId$, $reqObjId$, and $reqQoS$ are constant throughout the life of a request actor, and once defined, the replica associated to a request actor remains constant. Furthermore, the state of request actor must move from $Waiting$ to $Granted$ or $Denied$, and from $Granted$ to $Servicing$ to $reqCompleted$, and if the state is $Servicing$, then the replication of the associated replica is competed.

The final definition needed before we state the full specification deals with the use of resources as determined for a given configuration by the replica and request functions.

Definition 6 (TotalResource Property ($\phi_{\mathrm{res}}(S)$)). $\phi_{\mathrm{res}}(S)$ states that for every configuration in the system, every data source node, and every managed resource, the sum of the resource allocation over requests on the node do not exceed the nodes total capacity for that resource. The resources currently allocated on a DS node include resources allocated to streaming accepted MM requests, as given the $reqQoS$ function for requests assigned to that node, as well as replications that are currently ongoing.

Using the characterizing functions and constraints discussed above, we now give the requirements for a Resource-based MM service.

Definition 7 (Resource-Based MM Service). A system S provides Resource-based MM service with respect to requests in $MMreqset$, functions $QoSTranslate$, $capacity$, and the functions characterizing replica and request state as specified above iff

- S satisfies the constraints $\phi_{\mathrm{req}}(S)$ (Definition 5), $\phi_{\mathrm{repl}}(S)$ (Definition 4), and $\phi_{\mathrm{res}}(S)$ (Definition 6), and
- for $C \in S$, if there is an undelivered message, $MMreq$, with parameters (α_{cl}, MM, qs), then along any computation path from C there is a (unique) transition which delivers $MMreq$ and creates a new request actor, α^{req}, such that in the resulting configuration C': $reqClientId(C', \alpha^{req}) = \alpha_{cl}$, $reqObjId(C', \alpha^{req}) = MM$, $reqQoS(C', \alpha^{req}) = QoSTranslate(qs)$, $reqState(C', \alpha^{req}) = Waiting$, and $reqReplica(C', \alpha^{req}) = \mathtt{nil}$.

Theorem 1 (QoS2Resource). If a system S provides Resource-based MM service as defined in 7 and the function $QoSTranslate$ satisfies the requirements of definition 3, then S provides QoS Based Service according to definition 1.

5.3 A Resource Based MM Behavior

At the behavior level we make explicit the QoS meta actors that cooperate to provide the Resource-based MM service. Constraints on their behavior are expressed in terms of abstract meta-actor states and reaction rules specifying allowed actions.

Representing Request and Replica functions (MM resource state)

The actual MM resource state of a configuration, modeled previously by the request and replica functions, is recorded in annotations of base-level actors implementing the MM streaming service. We partition the MM base actors of a configuration into three groups. $ReqActors$ correspond to delivered MM requests and are located on the control node. $DSReqActors$ are request actors representing granted requests, each located on the DS node to which the request is assigned. We define $NodeReq(C, \alpha^{req})$ to be $\alpha_{\mathrm{ds}}^{req}$ if α^{req} represents a granted request with DS representative $\alpha_{\mathrm{ds}}^{req}$, and `nil` if the request status is waiting or denied. $DSReplActors$ correspond to the replica actors on the DS nodes. We define $NodeRepl(C, DS, MM)$ to be the replica base actor corresponding to the replica of MM on DS, if a replica is present, and `nil` otherwise.

Definition 8 (Replica and request functions). For each replica function $replX$, X one of $State$, BW, $Class$, there is an annotation tag X to represent that function: $replX(C, DS, MM)$ is the value of the X annotation of $NodeRepl(C, DS, MM)$ in C. Similarly, for each request function $reqX$ (for X one of $ClientId$, $ObjId$, QoS, $State$, $Replica$) there is an annotation tag X and $replX(C, \alpha^{req})$ is defined to be the value of the X annotation of $NodeReq(C, \alpha^{req})$ in C for a granted request, and to be the X annotation of α^{req} in C otherwise.

Representing MM Meta-Actor Behavior

Service(ActorName)	States
QoS Broker (QB)	$QBB(MMState, Status)$
Request Scheduler(RS)	$IdleB_{\mathrm{rs}}$, $WaitB_{\mathrm{rs}}(MMState, ReqActors, MMState)$
Replication on Demand (ROD)	$IdleB_{\mathrm{rod}}$, $WaitB_{\mathrm{rod}}(MMState)$
DeReplication(DR)	$IdleB_{\mathrm{dr}}$, $WaitB_{\mathrm{dr}}(MMState, \mathbf{P}_{\omega}(()DSnodes))$
Predictive Placement(PP)	$IdleB_{\mathrm{pp}}$, $WaitB_{\mathrm{pp}}(MMState, \mathbf{P}_{\omega}(()DSnodes))$
DSnode QoS mgr($DSma(DS)$)	$DSB(DS)$

Fig. 4. QoS meta actors, services and states.

Following the QoS Broker software architecture discussed in section 3 there are five broker meta actors residing on the DC node. In addition there is a DSNode meta actor $DSma(DS)$ on each DS node DS. The QoS meta actors, their services and their possible states are summarized in figure 4. We represent the QoS meta actors knowledge of the MM state as a function from requests (represented by actors in $ReqActors$) and replicas (represented by pairs (DS, MM)) to a function from annotation tags to corresponding values or `nil` if undefined. We let $MMState$ denote this set of functions, and we let $mms, mms', mmsU$ range over $MMState$.

The QoS broker QB coordinates the QoS resource management services: scheduling, replication, predictive placement, and dereplication. Since these activities use and modify the actual MM state, care must be taken to avoid interference among these activities. The QoS broker uses a function, $status$, to keep track of which processes are ongoing. $status(RS) = (\alpha^{req}, rod)$ for rod a boolean, indicates that RS has been requested to schedule α^{req} and scheduling is in progress, with RS allowed to invoke ROD only if rod is true, $status(RS) = \mathtt{nil}$ indicates that there is no outstanding request from QB to RS and consequently no undelivered messages to or from RS. For $X \in \{PP, DR\}$, $status(X) = \mathtt{true}$ indicates the process X is ongoing and $status(X) = \mathtt{false}$ indicates the process X is not active and there are no outstanding requests from QB to X. We let $Status$ denote the set of status functions and let $status$ range over $Status$. The rules for QB behavior assure non-interference amongst the QoS broker services, by not allowing two replication services to run concurrently or a dereplication service to run concurrently with either scheduling or replication.

Meta-level Messages

The QoS meta-actor rules specify the reaction of a meta-actor upon receiving a QoS message. Figure 5 summarizes the internal QoS messages. In addition to the internal QoS messages there are the messages used to communicate with clients, and transition rules for these messages. A client α_{cl} may send MM requests to QB of the form $QB \triangleleft \mathtt{mmReq}(\alpha_{cl}, MM, qs)$ and the reply has one of the following forms:

$$\alpha_{cl} \triangleleft \mathtt{granted}(MM, qs) \mathbin{@} QB \quad \text{or} \quad \alpha_{cl} \triangleleft \mathtt{denied}(MM, qs) \mathbin{@} QB.$$

The Transition Rules

In the following we briefly summarize the behavior specified by QoS Broker transition rules. The complete set of rules appears in the full paper.

QB rules. The transition rules for the QoS broker, QB, provide the overall organization of the QoS service activities. If neither dereplication nor request scheduling are in progress then an MM request, $QB \triangleleft \mathtt{mmReq}(\alpha_{cl}, MM, qs)$, can be processed. A message $RS \triangleleft \mathtt{schedule}(mms', \alpha^{req}, rod)$ is sent, where mms' is QBs current model of the MM state augmented with request information associated to the new request actor α^{req}, and rod indicates whether replication-on-demand is enabled for the scheduler.

QB may invoke predictive placement if dereplication is not in progress and replication-on-demand is not enabled, by sending a message of the form $PP \triangleleft \mathtt{place}(mms) \mathbin{@} QB$.

request	reply
$RS \triangleleft \mathtt{schedule}(\alpha^{req}, mms, b) @ QB$	$QB \triangleleft \mathtt{scheduleReply}(mmsU) @ RS$
$DR \triangleleft \mathtt{derepl}(mms) @ QB$	$QB \triangleleft \mathtt{dereplReply}(replU) @ DR$
$PP \triangleleft \mathtt{place}(mms) @ QB$	$QB \triangleleft \mathtt{placeReply}(replU) @ PP$
$DSma(DS) \triangleleft \mathtt{assign}(reqU) @ RS$	$RS \triangleleft \mathtt{assignAck}() @ DSma(DS)$
$ROD \triangleleft \mathtt{repl}(mms, MM, qt) @ RS$	$RS \triangleleft \mathtt{replAck}(replU) @ ROD$
$DSma(DS) \triangleleft \mathtt{repl}(replU) @ X$	$X \triangleleft \mathtt{replAck}() @ DSma(DS)$
for $X \in \{ROD, DR, PP\} \wedge DS \in DSnodes$	

notifications

$QB \triangleleft \mathtt{notify}([(DS, MM) = [State = replCompleted]]) @ DSma(DS)$

$QB \triangleleft \mathtt{notify}([\alpha^{req} = [State = reqCompleted]]) @ DSma(DS)$

Fig. 5. Internal QoS Messages. These are classified either as requests with corresponding reply messages, or as notifications, which need no reply. The general form of a message is $X \triangleleft \mathtt{mid}(\ldots) @ Y$ where X is the intended receiver of the message, Y is the sender, `mid` is the message type, and $(\ldots)$ contains parameters.

The firing of this rule depends only on the state of QB and, unlike most of the other rules, does not consume a message. Initiation of dereplication is analogous.

When a reply to an outstanding scheduling, placement or dereplication request arrives, QB updates its state using the update MM state contained in the reply message. In the case of a scheduling request, QB also sends a reply to the requesting client indicating whether the request has been granted or denied. Similarly, when a DS notification arrives the QoS broker uses the contained MM state to update its MM state. The `notify` messages are just to inform QB that some resources have been released. Notification is needed because the resources can not be considered available for reuse until such a notification is received by QB.

DS manager rules. When a DS node manager $DSma(DS)$ receives an assignment request with MM state $mmsU$ it creates a new request actor, sets the annotations of this actor and of the replica actor for the requested MM object using $mmsU$ (which contains the MM request information, and name of the associated request actor), and sends an `assignAck` reply to RS. When a DS node manager $DSma(DS)$ receives a replication request it uses the MM state replica information to update the annotations of its replica actors and then sends a `replAck` reply to the requester (which could be ROD, PP, or DR).

When servicing of a request with request actor $\alpha_{\mathrm{ds}}^{req}$ completes on a DS node, an event `reqCompletes`($\alpha_{\mathrm{ds}}^{req}$) is signaled. The DS node manager then updates the annotations of $\alpha_{\mathrm{ds}}^{req}$ to record the completion, and sends a notification to QB with the state update for the request actor associated to this request.

Similarly, when replication of an MM object on a DS nodes completes the replica actor annotations are updated, and a notification is sent to QB. Also, if any requests are waiting for this completion they are moved from $Granted$ state to $Servicing$ state.

Rules for RS, PP, ROD, DR .

RS. Suppose RS receives a scheduling request, $RS \triangleleft \mathtt{schedule}(mms, \alpha^{req}, rod)$. If there is some DS node to which the request represented by α^{req} in mms can be assigned without violating resource constraints, then RS picks one and sends an assignment request to the DS manager of that node. If there is no such DS node and ROD is disabled (rod is false), then RS sends the broker a denied update. If there is no DS suitable node to which the request can be assigned but rod is true, then RS sends a replication-on-demand request to ROD containing mms, along with the requested MM object and QoS requirement. If RS receives an acknowledgment to an outstanding assignment request to a DS node, then it sends a granted reply to the broker with MM state that contains the update information for α^{req} representing the request along with any replication update that has been done. If RS receives a replication update from an outstanding request to ROD that allows the request it is attempting to schedule to be granted, then RS picks a suitable DS node and sends a corresponding assign request to the nodes DS manager. If the replication update from ROD does not allow the request to be granted, then RS sends a denied reply to QB.

ROD. When ROD receives a request, $ROD \triangleleft \mathtt{repl}(mms, MM, qt)$ @ RS for replication of MM object, MM, with QoS resource requirements qt, it looks, using mms, for a DS node that doesn't have the needed MM object and that has the required resources available. If one is found, a replication request is sent to that DS node and ROD waits for an acknowledgment. When the acknowledgment message is received, a reply is sent to RS with MM state containing the replica update information. If no such DS node is found, then a failure reply is sent to RS.

PP, DR. Based on the information in the MM state of a `place` request, PP may decide to reclassify some replicas from 0, 2, or 3 to 1, and in the case of moving from 0 to 1 initiate replication. It then notifies each DS node of any changes on that node, waits for acknowledgments from these DS nodes, and then sends a reply to QB containing the updated replica state. Similarly, upon receiving a dereplication request, DR may decide to reclassify some replicas from 1 to 2. It then notifies each DS node of any changes on that node, waits for acknowledgments, and then sends a reply QB containing the replica state update. (Note that in practice, PP and DR also use other information for predicting requests which we do not model at this level of abstraction.)

Definition 9 (Resource-Based MM Behavior). A system S has *Resource-Based MM Behavior* with respect to the underlying system architecture (DS Nodes and capacity function), the QoS meta actors QB, RS, ROD, DR, PP, and the DS node managers $DSma(DSnodes)$, if

- for each configuration C in S, the state of X in C is appropriate for X according to Figure 4, for X one of QB, RS, ROD, DR, PP, or $DSma(DS)$) for $DS \in DSnodes$;
- every computation π of S obeys the transition rules for QoS meta-actors discussed above and guarantees termination of servicing and replication-in-progress states;
- for any transition $\tau = C \longrightarrow C'$ of S `replCompletes`(α) is an event of τ only if $getA(C, \alpha, State) = InProgress$, for any replica actor α, and `reqCompletes`$(\alpha_{\mathrm{ds}}^{req})$ is an event of τ only if $getA(C, \alpha_{\mathrm{ds}}^{req}, State) = Servicing$ for any DS request actor $\alpha_{\mathrm{ds}}^{req}$.

To state the "Resource-based MM Behavior provides QoS-based MM Service" theorem it is necessary to specify the conditions under which this in fact holds. For this purpose we define the requirements for QoS Initial configurations and QoS NonInterference. QoS Initial configurations are those in which no QoS meta activity is going on. QoS NonInterference expresses constraints on the environments in which the QoS system can operate correctly.

Definition 10 (QoS Initial). $QoSInitial(C)$ holds for a configuration C just if: QB's status function says there has no active processes; RS, ROD, DR, PP are Idle; and there are no undelivered internal QoS messages in C.

Definition 11 (NonInterference Requirement). S satisfies the *QoSBroker NonInterference Requirement* iff transitions that do not involve a QoS meta actor as the principle actor obey the following constraints

- Neither QoS annotations nor the state of actors in $ReqActors$, $DSReqActors$, or $DSReplActors$ are modified.
- No resource dedicated to QoS is used.
- No internal QoS messages (Definition 5) are sent.

Theorem 2 (Resource-based MM Behavior implies Resource-based MM Service). If a system S satisfies

- S has Resource-based MM behavior (Definition 9)
- S satisfies the QoS NonInterference Requirement (Definition 11)
- Every configuration C in S is reachable from a configuration satisfying the QoSInitial conditions (Definition 10)
- $QoSTranslate$ satisfies the QoSTranslate requirements (Definition 3)

then S provides Resource-based MM Service (Definition 7) with respect to the given functions $QoSTranslate$ and $capacity$, with $MMreqset$ being messages of the form $QB \triangleleft \texttt{mmReq}(\alpha_{cl}, MM, qs)$, and replica and request functions defined in terms of annotations according to Definition 8.

The proof of this theorem (which appears in the full paper) is organized as follows. First the possible system configurations are characterized in terms of combinations of meta actor states and undelivered messages is established. Then a notion of pending update for the broker model of the MM state is defined and it is shown that the broker MM state model modified by the pending updates is an accurate model of the actual MM state, and that the pending updates preserve the QoS replica, request, and resource constraints (with MM states in place of configurations). Finally, fairness of the actor model and the definition of QoS reaction rules are used to establish that the QoS broker meta actor, QB is always eventually able to receive a MM request.

Theorem 3 (Resource-based MM Behavior implies QoS-based MM Service). If a system S satisfies the premisses of Theorem 2, then S provides QoS-based MM Service (Definition 1) with respect to the given functions $QoSTranslate$ and $capacity$, with $MMreqset$ being messages of the form $QB \triangleleft \texttt{mmReq}(\alpha_{cl}, MM, qs)$.

Proof: By Theorem 2 and Theorem 1. □

5.4 Specifying an Adaptive Resource Scheduling Policy for a QoS Broker

In this section, we illustrate how we refine the Resource-Based MM Behavior specification, by introducing the load-factor based adaptive scheduling algorithm to constrain the behavior of the request scheduler meta-actor. To show this is correct we need only show that the resource-based behavior requirements are met. This follows from the fact that the algorithm meets the constraints implicit in the request scheduler transition rules.

The adaptive scheduling algorithm is used to select a data source to serve a request. The algorithm takes as input a MM request and returns the best data source on which to schedule that request based on the current load and the minimal load-factor criteria.

Recall that the request scheduler receives scheduling requests of the form $RS \triangleleft \texttt{schedule}(mms, \alpha^{req}, rod)$ from the QoS broker, where mms is the QoS brokers perception of the current MM state. In particular mms contains information about existing replicas and request assignments that reflects the availability of resources at each data source. We adapt the load-factor function (Section 4) to take an MM state, a request actor and a DS node as arguments, using correspondingly adapted functions for calculating the resource allocation in a given configuration. This is then used to define the $Candidates$ function that determines the best candidate DS node(s) for assignment of the request according to the load factor criteria.

Definition 12 (Candidates function). We calculate the available resources on each node for the load factor calculation by subtracting the resources currently allocated from the total capacity of a data source.

$$Available(mms, DS, Res) = capacity(DS, Res) - ResAlloc(mms, DS, Res)$$

where $ResAlloc(mms, DS, Res)$ is amount of resource Res allocated to requests on DS, according to the information in mms. The adaptive load factor calculation is then defined as follows.

$$LF(mms, \alpha^{req}, DS) = max(\frac{qt_{DB}}{DB^{DS}}, \frac{qt_{Mem}}{Mem^{DS}}, \frac{qt_{CPU}}{CPU^{DS}}, \frac{qt_{NetBW}}{NetBW^{DS}})$$

where

$$qt = mms(\alpha^{req}, QoS)$$

$$R^{DS} = Available(mms, DS, R) \quad \text{for} \quad R \in Resources$$

The candidate data sources, $Candidates(mms, \alpha^{req})$ are those such that the load factor is minimal and not infinite, and a replica of the requested MM object exists on the data source.

We define Adaptive Request Scheduling MM Behavior by modifying the RS rules for scheduling requests (§ 5.3) to pick a data source for assignment from the candidates set rather than from the set of all data sources that have the required resources available. The rules for ROD are similarly modified to use the load factor calculation to find a candidate node for replication.

Definition 13 (Adaptive RS MM Behavior). A system S has *Adaptive Request Scheduling MM Behavior* with respect to the underlying system architecture, the QoS meta actors QB, RS, ROD, DR, PP, and the DS node managers $DSma(DSnodes)$, if it satisfies the conditions for Resource-based MM behavior, modified by replacing the request scheduler and replication-on-demand rules as discussed above.

The correctness theorem for Adaptive Resource Scheduling MM Behavior is the following.

Theorem 4 (Adaptive RS MM Behavior implies QoS-based MM Service). If

- S has Adaptive Resource Scheduling MM behavior (Definition 13)
- S satisfies the QoS NonInterference Requirement (Definition 11)
- every $C \in S$ is reachable from a configuration satisfying the QoSInitial conditions (Definition 10)
- $QoSTranslate$ satisfies the QoSTranslate requirements (Definition 3)

then S provides QoS-based MM Service (Definition 1).

Proof: By Theorem 3 we only need to show that under the QoS-Initial and QoS-NonInterference assumptions a system that has Adaptive RS MM Behavior also has Resource-based MM Behavior. For this, it is sufficient to check that the each transition arising from an Adaptive RS or ROD rule is a transition allowed by the corresponding generic Resource-based rule. This holds because (1) every DS node in $Candidates(mms, \alpha^{req})$ can be assigned the request associated to α^{req} in mms by the generic rules, and (2) if $Candidates(mms, \alpha^{req}) = \emptyset$ then there is no DS node that can be assigned the request associated to α^{req} in mms by the generic rules. □

6 Related Work and Future Research Directions

Commercially available object-based middleware infrastructures such as 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, or the implication of composing object services. Architectures that provide real-time extensions to CORBA [22, 32] necessary to support timing-based QoS requirements [35] have been proposed and used to study performance optimizations [12]. In the Java Development Environment, the ability to deal with real-time thread management is dependent on the underlying threads implementation, making QoS support complicated to achieve. Various systems such as the Infospheres Infrastructure [7] and the Globe System [24] explore the construction of large scale distributed systems using the paradigm of distributed objects. Globus, a metacomputing framework, defines a QoS component called Related work in the area of multimedia QoS management includes projects such as *QualMan* [19] and systems that implement a variety of algorithms for MM server management [33, 30, 10, 31, 9].

Reflection allows application objects to customize the system behavior as in Apertos [13] and 2K [15]. The Aspect Oriented Programming paradigm [14] makes it possible to express programs where design decisions can be appropriately isolated permitting composition and re-use. Some of the more recent research on actors has focused on coordination structures, meta-architectures and runtime systems [2, 23]. In other reflective models for distributed object computation [20, 8, 4], an object is represented by multiple models allowing behavior to be described at different levels of abstraction and from different points of view. Many of the middleware systems described focus heavily on implementation issues while the focus of the work presented in this paper is on developing formal semantics and reasoning for a QoS-based middleware environment.

Much of the work on formal models for QoS has been in the context of QoS specification mechanisms and constructs. In some implementation driven methods of QoS specification, the specification of QoS requirements is intermixed with the service specification [16, 17]. Other approaches address the representation of QoS via multi-paradigm specification techniques that specify functional behavior and performance ' constraints distinctly using multiple languages [3, 34, 5, 6]. Synchronizers and RtSynchronizers [11, 21] allow us to express QoS constraints via coordination constraints in the actor model.

We are actively working on extending the existing meta-architecture to support more services. Specifying and reasoning about enforcement of timing-based QoS requirements of multiple sessions involves a more thorough treatment of time and synchronization. For end-to-end QoS, it is necessary to determine how *real-time scheduling* strategies for time constrained task management interact with strategies for other tasks such as CPU intensive calculations, or network communication with clients. We are currently working on formalizing other components of the MM meta-architecture such as message scheduling and synchronization. Supporting requirements such as fault-tolerance, availability and hard real-time QoS will require further extensions of the existing MM metaarchitecture. We are currently studying the composability of multiple protocols and mechanisms to address these requirements in the MM metaarchitecture.

In general, the dynamic nature of applications such as those of multimedia under varying network conditions, request traffic, etc. imply that resource management policies must be dynamic and customizable. Current mechanisms, which allow arbitrary objects to be plugged together, are not sufficient to capture the richness of interactions between resource managers and application components. For example, they do not allow customization of execution protocols for scheduling, replication, etc. This implies that the components must be redefined to incorporate the different protocols representing such interaction. In this paper we have shown, using the QoS broker MM architecture, how a meta-architectural framework, such as the TLAM, can be used to specify and reason about distributed middleware services and their composition, and have also indicated how specifications in the framework can lead to implementations. We believe that a cleanly defined meta-architecture which supports customization and composition of protocols and services is needed to support the flexible use of component based software.

[1]

References

1. G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, Cambridge, Mass., 1986.
2. G. Agha, S. Frølund, W. Kim, R. Panwar, A. Patterson, and D. Sturman. Abstraction and Modularity Mechanisms for Concurrent Computing. *IEEE Parallel and Distributed Technology: Systems and Applications*, 1(2):3–14, May 1993.
3. G. Blair, L. Blair, H. Bowman, and A. Chetwynd. *Formal Specifications of Distributed Multimedia Systems*. UCL Press, 1988.

[1] Acknowledgements: The authors would like to thank the anonymous referees for valuable suggestions for improving previous versions of this paper. This research was partially supported by DARPA/NASA NAS2-98073, ONR N00012-99-C-0198, ARPA/SRI subcontract 17-000042, NSF CCR-9900326.

4. G. Blair, G. Coulson, P.Robin, and M. Papathomas. An architecture for next generation middleware. In *Middleware '98*, 1998.
5. L. Blair and G. Blair. Composition in multiparadigm specification techniques. In *ECOOP Workshop on Aspect Oriented Programming*, July 1999.
6. L. Blair and G. Blair. Composition in multiparadigm specification techniques. In *IFIP Workshop on Formal Methods for Open Object-based Distributed Systems, FMOODS'99*, Feb. 1999.
7. 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.
8. 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.
9. A. Dan and D. Sitaram. An online video placement policy based on bandwidth to space ratio (bsr). In *SIGMOD '95*, pages 376–385, 1995.
10. A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *ACM Multimedia Systems*, 4:112–121, 1996.
11. S. Frølund. *Coordinating Distributed Objects: An Actor-Based Approach to Synchronization*. MIT Press, 1996.
12. A. Gokhale and D. C. Schmidt. Evaluating the Performance of Demultiplexing Strategies for Real-time CORBA. In *Proceedings of GLOBECOM '97*, Phoenix, AZ, 1997.
13. 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.
14. 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.
15. 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.
16. P. Leydekkers and V. Gay. Odp view on qos for open distributed mm environments. In J. d. Meer and A. Vogel, editors, *4th International IFIP Workshop on Quality of Service, IwQos96 Paris, France*, pages 45–55, Mar. 1996.
17. F. H. d. S. Lima and E. R. M. Madeira. Odp based qos specification for the multiware platform. In J. d. Meer and A. Vogel, editors, *4th International IFIP Workshop on Quality of Service, IwQos96 Paris, France*, pages 45–55, Mar. 1996.
18. K. Nahrstedt. *Network Service Customization: End-Point Perspective*. PhD thesis, University of Pennsylvannia, 1995.
19. K. Nahrstedt, H.-H. Chu, and S. Narayan. Qos-aware resource management for distributed multimedia applications.
20. 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 Archititetures*, pages 36–47. ACM SIGPLAN, 1992.

21. 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.
22. 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.
23. 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.
24. 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.
25. N. Venkatasubramanian. *An Adaptive Resource Management Architecture for Global Distributed Computing*. PhD thesis, University of Illinois, Urbana-Champaign, 1998.
26. 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.
27. 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.
28. N. Venkatasubramanian and C. L. Talcott. A metaarchitecture for distributed resource management. In *Hawaii International Conference on System Sciences, HICSS-26*, Jan. 1993.
29. 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.
30. H. M. Vin and P. V. Rangan. Designing a multi-user hdtv storage server. *IEEE Journal on Selected Areas in Communications*, 11(1):153–164, Jan. 1993.
31. J. L. Wolf, P. S. Yu, and H. Shachnai. Dasd dancing: A disk load balancing optimization scheme for video-on-demand computer systems. In *Proceedings of ACM SIGMETRICS '95, Performance Evaluation Review*, pages 157–166, May 1995.
32. 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.
33. P. Yu, M. Chen, and D. Kandlur. Design and analysis of a grouped sweeping scheme for multimedia storage management. *Proceedings of Third International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego*, pages 38–49, November 1992.
34. P. Zave and M. Jackson. Requirements for telecommunications services: An attack on complexity. In *IEEE International Symposium on Requirements Engineering*, 1997.
35. J. Zinky, D. Bakken, and R. Schantz. Architectural support of quality of service. *Theory and Practice of Object Systems*, 3(1), 1997.