A MetaArchitecture for QoS-Based Distributed Resource Management

Nalini Venkatasubramanian[*] Gul Agha[†] Carolyn Talcott[‡]

Abstract
Systems that provide distributed multimedia services are subject to constant evolution - customizable middleware is required to effectively manage this change. In this paper we develop a meta-architectural framework for customizable QoS-based middleware using Actors, a model of concurrent active objects. Middleware services for resource management execute concurrently with each other and with application activities— scheduling, protocols providing security and reliability, load balancing and stream synchronization can therefore potentially interfere with each other. To ensure cost-effective QoS in distributed multimedia systems, safe composability of resource management services is essential. For instance, system protocols and activities must not cause arbitrary delays in the presence of timing based QoS constraints. 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. *Keywords: Distributed Multimedia, resource management, Qos control/scheduling.*

In the coming years, distributed multimedia servers will be deployed to deliver a variety of interactive, digital multimedia(MM) services over emerging broadband (wide-area) networks [**?**] to form a wide-area infrastructure. Future multimedia services will include such diverse applications as telemedicine, electronic commerce, instructional video-on-demand and digital studios. These kinds of applications exhibit varying requirements such as timeliness, security, reliability and availability. For example, future clinical environments and medical information systems require access to a variety of information with stringent requirements on security, reliability, timeliness and accuracy of data.

Although service requirements for many applications are stringent, most MM applications can tolerate minor, infrequent violations of their performance requirements. This degree of freedom is specified as a quality-of-service (QoS) parameter, for e.g., tolerable jitter in a video frame, and must be enforced by the underlying system. Systems that provide distributed multimedia services are continuously changing and evolving. For instance,

- The set of servers and clients in a wide area infrastructure change continuously.
- Multiple protocols that cater to a wide range of network configurations and software applications must be supported; new protocols will need to be incorporated seamlessly.
- The number of users of a distributed application and their requirements varies with time.
- QoS requirements change dynamically based on service availability, pricing and negotiation factors.
- Network conditions vary depending on congestion, adaptive routing and failures.
- The degree of load at the endpoints of the system can change based on users and applications.

In order to manage the distributed components and adapt to the above dynamic changes in multimedia applications, middleware services are required. Providing customizability in the middleware allows efficient management of dynamicity and ensures cost effectiveness of the system. As networked applications and global distributed services expand and become more widely deployed, customizable infrastructures will become a necessity. In this paper, we describe foundational techniques for developing components of customizable component based middleware infrastructures. We also develop a framework that enables flexible and cost-effective management of resources in a distributed multimedia infrastructure.

Today, the task of distributed systems management is performed in middleware layers using frameworks such as CORBA and DCOM. Such frameworks are designed for heterogeneous interoperability but are limited in the degree of flexibility and customizability of services. They provide only limited capabilities for the specification and adaptation of end-to-end timing based QoS properties. Future applications will require dynamic invocation and revocation of services distributed in the network without violating QoS constraints of ongoing applications. Customizable middleware

[*]Contact Author: Nalini Venkatasubramanian Dept. of Information and Computer Science, Univ. of California Irvine, Irvine, CA 92697-3425, Email: nalini@ics.uci.edu

[†]Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, Email: agha@cs.uiuc.edu

[‡]Department of Computer Science, Stanford University, Stanford, CA 94305, Email: clt@cs.stanford.edu

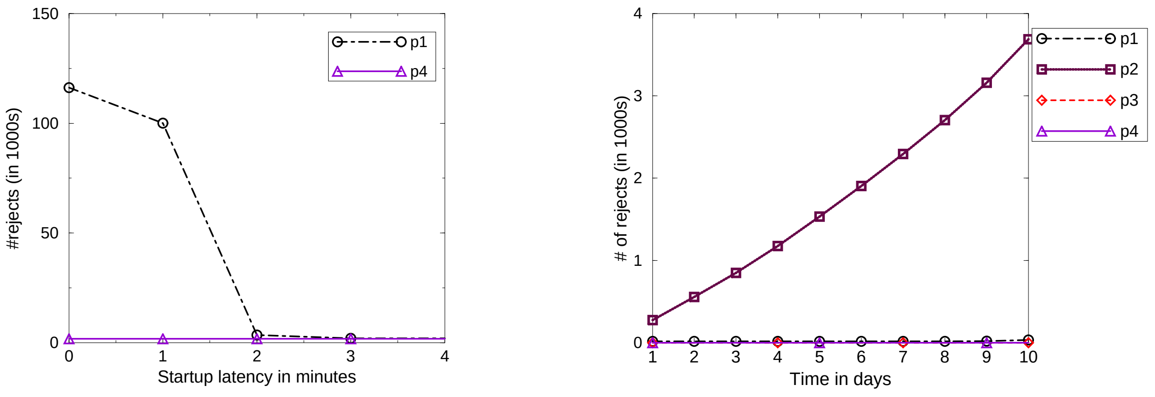

Figure 1: Comparison of the performance of load management policies for request scheduling and video placement in a distributed video server. Policies P1 and P2 represent adaptive and predictive policies exclusively and P3 and P4 represent composite adaptive and predictive mechanisms

allows us to deal with changes in systems and applications in a non-intrusive way. Customizability also facilitates dynamic performance tuning, for example, by replacing one management mechanism/protocol by more suitable ones as system conditions vary.

In the following sections, we develop a model for flexible, cost-effective middleware to enforce QoS requirements in multimedia applications. Specifically, we describe a two-level multimedia management architecture based on Actors, a model of concurrent active objects. In particular, we illustrate

- a placement policy that determines the degree of replication necessary for popular MM objects using a cost-based optimization procedure based on *a priori* predictions of expected subscriber requests.
- For scheduling requests, we represent an adaptive scheduling policy that compares the relative utilization of resources in a multimedia server to determine an assignment of requests to replicas.
- To optimize storage utilization, we introduce methods for dereplication of MM objects based on changes in their popularity and in server usage patterns.

We develop an object-based multimedia meta-architecture that consists of application MM objects servicing MM streams from multiple servers and a metalevel load-management system that manipulates the base level execution environment to improve performance. Performance studies show application objects can be managed effectively by composing multiple resource management activities managed at the metalevel [**?**]. Figure **??** illustrates the performance, measured by request rejection rate, of various policies for load management - (1) purely adaptive(P1), (2) purely predictive(P2), - (3) composite adaptive and predictive(P3 and P4). The left hand side of Figure **??** 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 (on the fly). The graph demonstrates than when the startup latency is below a threshold value of 2 minutes, the purely adaptive mechanisms, represented by P1 force a very large fraction of the requests received to be rejected. The right hand side of Figure **??**, depicts the change in number of rejected requests as a function of time, assuming that startup latency is sufficiently large. The inadequacy of P2, that relies on only predictive policies for scheduling and placement, can be observed in comparison to the other 3 policies, which 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.

The middleware framework developed in this paper ensures composability of concurrently executing system components. This will allow safe integration of mechanisms for services such as load balancing, fault tolerance and end-to-end QoS management. Our middleware framework is based on a semantic model for open distributed systems that supports specification and reasoning about multiple aspects of a system ranging from system-wide requirements to behavior and interactions of individual components making up the system. Using this framework, we represent policies for load management in distributed multimedia servers. Detailed specification of system properties in the actor-based multimedia metaarchitecture and reasoning about the correctness of the load management implementation is beyond the scope of this paper and is described elsewhere [**?**].

The rest of this paper is organized as follows: Section 1 reviews on the two level semantic framework for distributed resource management based on the model of concurrent objects. Section 2 describes a multimedia metaarchitecure, its components and interactions among resource management services. Section 3 presents the different policies and optimizations for load management represented in the meta-architecture. Section 4 briefly describes the formalization of the multimedia meta-architecture and the representation of the policies discussed. Section 5 discusses related work and outlines areas for future research.

1 The Two Level MetaArchitectural Model

Distributed multimedia systems handle applications that are open and interactive; such systems evolve dynamically and their components interact with an environment that is not under their control. Actors, a model of concurrent active objects, has a built-in notion of encapsulation and interaction and and is well suited to represent evolution and co-ordination among interacting components in distributed multimedia applications.

In the actor paradigm, the universe contains computational agents called *actors*, distributed over a network. Traditional passive objects encapsulate state and a set of procedures that manipulate the state; actors extend this by encapsulating a thread of control as well. Each actor potentially executes in parallel with other actors and may send messages to actors it knows the addresses of. The Actor model of computation has the following characteristics:

- Each actor has a conceptual location (its *mail address*) and a *behavior*. Actor addresses may be communicated in messages, allowing dynamic interconnection governed by *locality laws*.
- The communication topology of an actor system is called the *acquaintance relation*. An actor can only send messages to its acquaintances. The acquaintances of an actor are among those it is given at creation time, those it has created and those sent to it in a message. The acquaintances that an actor can be given at creation time are among the acquaintances of its creator. An actor can also forget acquaintances. Thus the topology can change dynamically.
- Finally, new actors may be created; such actors have their own unique addresses. On receiving a communication, an actor processes the message and as a result may cause one or more of the following events: (a) Creation of a new actor, (b) Change of behavior, and (c) Sending of a message to an existing actor.

(See [**?**, **?**] for more discussion of the actor model, and for many examples of programming with actors.)

In [**?**], we presented the TLAM(Two Level Actor Machine) model as a first step towards providing a formal semantics for specifying and reasoning about properties of and interactions between components of open distributed systems. In the TLAM, a system is composed of two kinds of actors, base actors and meta actors, distributed over a network of processing nodes. Base level actors carry out application level computation, while meta-actors are part of the runtime system which manages system resources and controls the runtime behavior of the base level. Meta-actors communicate with each other via message passing as do base level actors, but they may also examine and modify the state of the base actors located on the same node. The application level of the model refines the model of [**?**], explicitly representing more of the runtime structures and resources. It also abstracts from the choice of a specific programming language, providing a framework for reasoning about heterogeneous systems.

A TLAM provides an abstract characterization of actor identity, state, messages, and computation, and of the connection between base and meta level computation. 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. This provides a flexible mechanism for interaction of meta-actors with the built-in runtime system. A TLAM configuration, C, has a set of base and meta level actors and a set of undelivered messages. The actors are distributed over the TLAM nodes. Each actor has a unique name (address) and the configuration associates a current state to each actor name. The undelivered messages are distributed over the network – some are traveling along communication links and others are held in node buffers. The semantics of a TLAM is given by a labeled transition relation on configurations. There are two kinds of transitions: communication and execution. Communication transitions move undelivered messages around the network and are the same in every TLAM. An execution transition is a computation step taken by a base or meta level actor. These transitions are described by *reaction rules* specific to a particular TLAM that determine how individual actors react to messages received, and in the case of meta actors how they react to other events. A *computation path* for a configuration is a possibly infinite sequence of labeled transitions. The semantics of a configuration is the set of fair computation paths starting with that configuration. A *system* is a set of configurations closed under the transition relation.

Core Services for Distributed Resource Management

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

1. Serialize or delay activities: This can result in over-serialization of resource management activities causing performance degradation. A more serious consequence, however, is that global delays and halts may cause QoS violations, making this approach unattractive when timing based QoS constraints need to be satisfied.

2. Design a closed system: Here, hand-crafted mechanisms are tied to applications. This results in loss of openness, flexibility and modularity. It also reduces portability of code and time is wasted in re-engineering systems for new applications.

3. Ad-hoc validation of non-interference among design policies: The system runs the risk of failure and unknown consequences.

We argue that composability of resource management activities is not just desirable, but *essential* to ensure cost-effective QoS in distributed multimedia systems. For instance, system protocols and activities must not enforce arbitrary delays in the presence of timing based QoS constraints. This adds complexity to the design and reasoning of such systems. We would like to be able to consider separately issues such as: functional behavior of an application; and resource management issues such as memory management, load balancing, QoS specification and enforcement. To ensure non-interference and manage the complexity of reasoning about components of open distributed systems in general, our strategy is to identify key system services where non-trivial interactions between the application and system occur, i.e. base-meta interactions. We refer to these key services as *core services*. We use the commonly observed patterns in distributed algorithms to identify three core basic activities:

- Recreation of services/data at a remote site
- Capturing information at multiple nodes/sites
- Interactions with a global repository.

Correspondingly, we have specified three core services - remote creation, distributed snapshot and directory services in the TLAM framework that can be used as a basis for more complex activities. Core services may be used in specifying and implementing more complex activities within the actor framework as purely meta-level interactions. For example, remote creation can be used as the basis for designing algorithms for activities such as migration, replication and load balancing. Distributed snapshots (cf. [**?**]). can also be used as the basis for global activities like quiescence detection, checkpointing and recovery and distributed garbage collection (cf. [**?**]). Similarly, a naming service can be used to design access control mechanisms, routing policies and group based communication. The development of suitable non-interference requirements allows us to reason about composition of multiple system services.

2 A Meta-Architecture for Multimedia Servers

Building on the two-level described in the previous section, we develop a meta-architectural model of a multimedia server that provides QoS based services to applications. This meta-architecture is the basis of experimental implementation of resource management mechanisms for distributed MM servers. The details of the mechanisms implemented and the performance evaluation of the composite environment is described in [**?**]. The architecture of the MM server consists of:

- A set of data sources 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

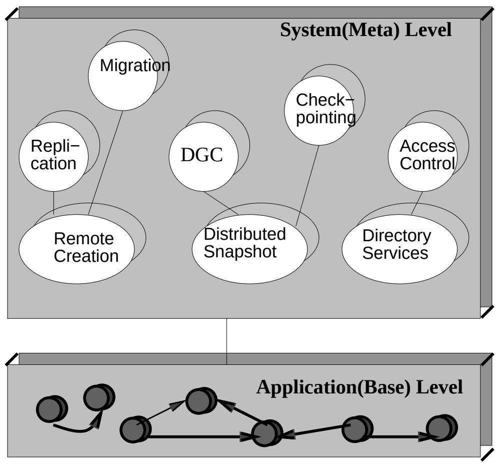

Figure 2: Classification of Core Services

- a specific node designated as the distribution controller that coordinates the execution of requests on the data source 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 computers and set-top devices at the client end. A lower speed back-channel conveys subscriber commands back to the data sources via the DC. The meta-architecture model of a multimedia server consists of two subsystems - the base level and meta-level subsystems corresponding to the application and system management components respectively. We develop the meta-level system to ensure QoS based resource management and show how this relates to the core service based model of distributed resource management.

2.1 The Base Level System

The base level component of the meta-architecture implements the functionality of the multimedia application. Two important components of a multimedia session that an application deals with are (1) data – which include MM objects, e.g. video and audio files, and (2) requests to access this data via sessions. In the MM meta-architecture, we distinguish between two kinds of base actors that represent these two aspects.

• **Media request actors:** that represent individual MM requests. They hold information about the media request including the audio/video (and possibly other) data required and the QoS requirements for that request. A request actor accesses the replica objects needed to satisfy the request, and may perform other tasks like synchronization of multiple data elements in order to generate a stream.

• **Media replica actors:** hold the representation of MM data, e.g. video object, audio object etc. A replica object stores state information corresponding to the media object. For example, it holds metadata corresponding to an object such as the length of video, amount of storage occupied, start location(disk block) of object and location of fast-forward/fast-rewind streams.

2.2 The Meta-Level System

The meta-level component handles system related issues transparent to the end-user of an application. It deals with the coordination of multiple requests and sharing of existing resources among multiple requests. To provide coordination at the highest level and perform admission control for new incoming sessions, we introduce a meta-level entity called the *QoS Broker*. Corresponding to the two kinds of base objects, the two main function of the QoS Broker are data management and request management. The high level architecture of the system is depicted in Figure **??**.

- **Data management:** The data management component decides on the placement of replica actors in the distributed MM system, i.e. it decides when and where to create additional replicas of data. It also determines

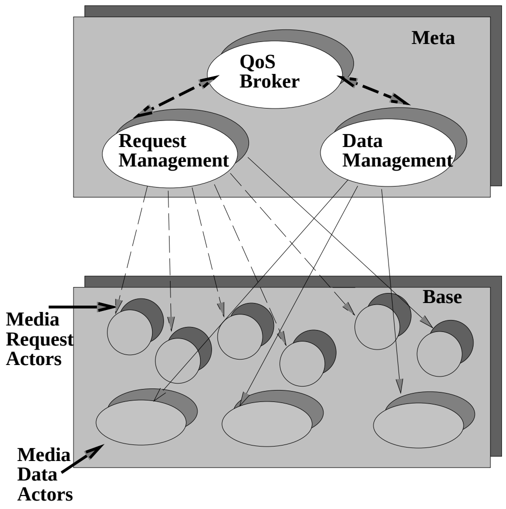

Figure 3: High Level View of a Multimedia Meta-Architecture

when additional replicas are no longer needed and can be garbage collected/dereplicated. *Data Placement* can be performed adaptively or offline using a predefined placement strategy. A predictive placement policy determines the degree of replication necessary for popular objects using a cost-based optimization procedure based on *a priori* predictions of expected subscriber requests. The adaptive data placement policy dynamically initiates data placement when data or resources that are required to satisfy a request are not available.

- **Request management** - Request arrival is unpredictable; typically, it is not possible to know when requests will arrive and for how long they will last [1]. This creates a need for adaptive admission control. Requests establish the overall QoS constraints for a session. The feasibility of these constraints is determined during the request scheduling process when a new request arrives. During this process of request scheduling, the request is assigned to a DS node based on existing load conditions and data availability[2]. Once a request has been admitted, a message scheduling component assures the satisfaction of QoS constraints for service within the session. The two main functions of the request management component are:
  - Request scheduling - For scheduling requests, we propose an adaptive scheduling policy that compares the relative utilization of resources in a MM server in order to determine an assignment of requests to replicas.
  - Message scheduling - which enforces QoS constraint satisfaction once requests have been initiated.

Media data management must decide where to place the replica actors in such a system, while media request management must figure out how to schedule actors to guarantee the properties (availability, timeliness, etc.) required by a service request. These functions in turn require a number of services. For example:

Synchronization service: schedules message service in order to maintain a uniform notion of time among multiple nodes and thereby among actors in the session that are distributed across multiple nodes. The notion of synchronization among audio and video actors can be built by using this uniform time value. This can be implemented as a snapshot broadcast phase that informs nodes about the current value of time within some delta. In [**?**], we specify the notion of a session in the context of MM actors and describe a construct, *QoSSynchronizer*, for expressing session specific QoS constraints.

Replication service: creates replicas to support availability, load-balancing and fault tolerance. When the existing copies of a MM object are not capable of supporting additional subscriber requests, replication of the MM object becomes necessary. At the time of replication it is necessary to choose a data source on which a new replica should be created. Once a candidate source has been chosen, replication may proceed. The rate at which replication proceeds has a direct impact on the degree of application interactivity. Replication at a rate that is faster than real-time minimizes the time for replication (e.g., replication of a 2 hour long movie at 10 times the playback speed will take 12 minutes), but consumes greater server resources. Whether the playback of a MM object can commence while replication is

[1]There are special applications like NVOD where it is possible to predict when the next stream of transmission will begin.

[2]Note that with multimedia information, scheduling of requests is based on placement of data. This is in contrast to applications in scientific computing where data placement is based on request scheduling.

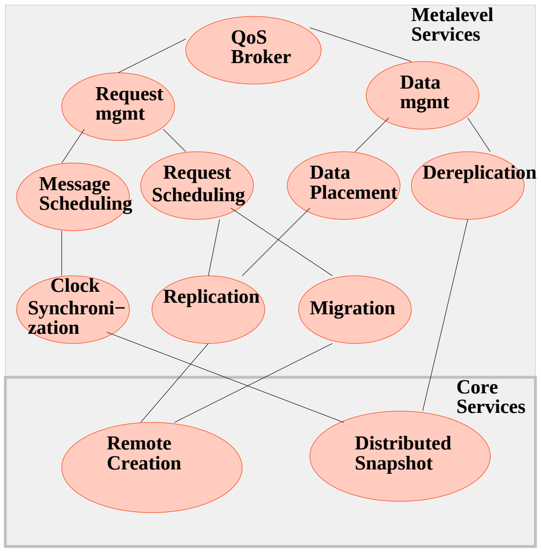

Figure 4: Detailed Architecture of the Meta-level System

in progress or whether playback should commence only after replication is completed depends on the type of the subscriber request. For example, in the case of non-interactive playback applications such as digital broadcasts, or in pseudo-interactive applications such as Near Video-on-Demand (NVOD), playback can commence at any stage after replication is initiated. On the other hand, to support True Video-on-Demand (TVOD) playback can commence only after replication is completed.

Dereplication service: can dereplicate data (additional replicas). In many time periods, a MM object may drop in popularity. By tracking the demand for different MM objects and determining when copies of a MM object can be dereplicated, the load management procedure can optimize utilization of storage space on a MM server. In fact, since storage space is a premium resource in a data source, dereplication or garbage collection is fundamental for effective load management. As in the case of replication, the copy to be dereplicated must be carefully chosen based on current load on the different data sources, as well as the expected future demands for other MM objects currently being stored or that are expected to be stored on the MM server. Like replication, dereplication may also not occur instantly: a copy that has been chosen for dereplication may be removed only after all requests that are currently being serviced by that copy have completed.

Migration service: migrates data or requests to support load balancing, availability and locality. Often, instead of replicating a MM object to service a new request, a cost-effective load management technique is to migrate an existing request to another data source. Such *copy-free* migration is an attractive alternative for Internet Web servers and other services that handle non-real-time data. Often, this requires explicit tear-down and reestablishment of network connections as well as exchange of state information between data sources, both of which can cause significant playback jitter in the case of a video stream.

In practice, replication and dereplication can be performed adaptively. However, considering the delay between the initiation and completion of either operation, it is most likely that both of these load management operations have to be initiated predictively, based on expected future subscriber request patterns. Each of these services in turn can be based on one or more of the three core services discussed earlier - remote creation and distributed snapshot. The organization of meta-level services provided by a QoS broker is shown in Figure **??**.

2.3 Mapping the Meta-system to the Physical Architecture

We distinguish between local and global components and define interactions between local resource managers on nodes and the global resource management component, the QoS broker, to satisfy user demands. The node local components [3] of the meta-architectural implementation of the QoS broker service include, for each DS node:

[3]components and facilities local to a node

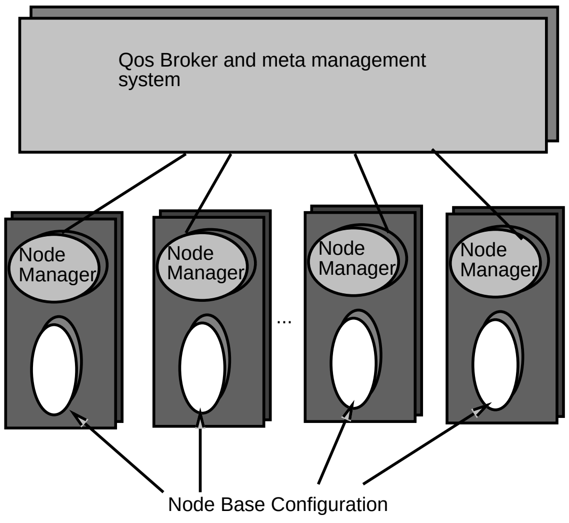

Figure 5: Architecture of a Networked Multimedia System

- 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.

The global component of the meta-architectural implementation includes a system wide QoS brokerage service that resides on the distribution controller node. The distribution controller is the architectural unit responsible for managing the assignments of requests and replicas. In addition, it may perform other administrative tasks such as logging and billing. The QoS broker is a meta-actor on the distribution controller that contains system wide state information as well as connections to all the DS meta-actors which hold the state of each node in the system. The QoS broker receives all subscriber requests directed to the multimedia server, considers the current commitment of the data sources, and evaluates whether a new request can be admitted for service without affecting service for the requests being currently serviced. In the process, the broker also determines the data source to which the new request should be assigned.

Apart from the QoS broker, the MM system contains a number of meta-actors whose behaviors implement specific resource management policies. The behaviors of these meta-actors are such that they collectively provide the desired MM service. still at a very high level, but they provide a clear path to actual system implementation, and in fact reflect the existing experimental implementation.

More specifically, an MM system providing QoS Brokerage service contains a collection of meta-actors such as – QB (a QoS Broker), RS (a Request Scheduler), ROD (Replication-On-Demand), DR (Dereplication), PP (Predictive Placement), ER (Eager Replication). These meta-actors are located on the DC node. The QoS broker meta actor contains a model of the system wide state and behaviors of these meta-actors are such that they collectively provide the desired MM service as described in the next section.

3 Load Management in Multimedia Servers

In this section, we describe mechanisms that 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 entails 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: $LF(R, DS) = max(\frac{DB^R}{DB^{DS}}, \frac{M^R}{M^{DS}}, \frac{CPU^R}{CPU^{DS}}, \frac{NetBW^R}{NetBW^{DS}})$ where $DB^{R(DS)}$, $M^{R(DS)}$, $CPU^{R(DS)}$, and $NetBW^{R(DS)}$ denote the disk bandwidth, memory buffer space, CPU cycles, and network transfer bandwidth, respectively, that are necessary for supporting request R (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 broker and various helped meta-actors.

3.1 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 RS meta-actor can either reject the incoming request or initiate *replication on demand*. In the former case, the RS meta-actor rejects the subscriber request in the short term, analyzes the rate of rejections over longer time frame and triggers appropriate placement policies to reduce the rate of rejection. With *replication on demand*, the RS meta-actor decides to create a new replica of the MM object on one of the sources on the fly. The source DS_j on which the new replica is to be made can be determined to be the one that has minimum load-factor with respect to R, i.e., with the minimum value of $LF(R, DS_j)$. By doing so, the QoS broker attempts to maximize the possibility of 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.

3.2 Placement of MM Objects

The placement meta-actor(PP) implements a placement policy that determines when, where, for how long, and how many replicas of each MM object should be placed in a MM server so as to maximize the number of requests serviced by it. Since reallocation of MM objects to data sources is a time-consuming and resource-intensive process, such reallocation should ideally be performed in advance, to satisfy expected future subscriber requests.

Predictive placement can be initiated periodically. The periodicity can be varied and the frequency with which the predictive placement process is executed governs its effectiveness. Smaller the time period, greater the computational overheads and greater the difficulty in predicting the request rates more accurately. Infrequent execution of predictive placement may result in under-utilization of the MM server's In general, using predictive placement, the MM server has a more accurate estimate of the current load of the system and a better prediction of expected subscriber demands, and hence, the allocation of MM objects can be changed so as to adapt to changes in request patterns more quickly, resulting in more flexible load management. Note that in order for predictive placement to execute concurrently with the adaptive scheduling model, a current snapshot of the system is taken prior to initiating predictive placement. The two main tasks of the predictive placement module are replication of MM objects from one data source to another, or from tertiary storage to secondary storage, and dereplication of MM objects from the data sources.

The placement module must determine how many replicas of each MM object are necessary, and which of the data sources these replicas should be allocated to. The number of replicas necessary for each MM object is dependent on the data source(s) to which the replicas are allocated, and upon the degree of loading of the data sources. In a predictive placement policy, the expected number of requests for that object is also taken into consideration. Therefore, in the process of determining the number and location of replicas that are necessary, the predictive placement module has to derive an allocation of MM objects to data sources, as well as a tentative assignment of expected requests to data sources; we term this the *pseudo-allocation* procedure.

Placement mechanisms must be designed to work effectively with request scheduling. The goal of the pseudo-allocation procedure is to facilitate the task of the adaptive scheduler module, 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. In this procedure, the request assignment is tentative in the sense that the adaptive scheduling module may abide by this assignment only if the actual subscriber requests match the expected pattern. Also, note that in deriving a tentative assignment for subscriber requests, the predictive placement module does not consider the exact times at which requests may arrive. The adaptive scheduling module, on the other hand, makes assignment decisions based on the exact arrival times of requests.

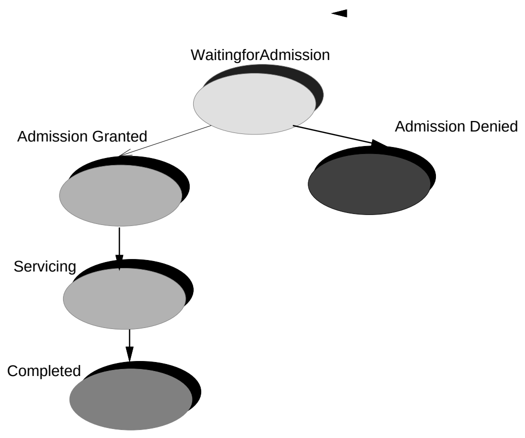

Figure 6: Change of Request State: State Transition Diagram depicting the change of state of requests

The pseudo-allocation procedure only determines when and to where replication must be initiated. The predictive placement module then has to figure out at what rate the replication should proceed. Replication may be initiated at the maximum feasible replication rate and this rate may even be changed dynamically, depending on the instantaneous load on the data sources and the number of replications scheduled to be performed from the same data source (replications from the same data source may proceed either simultaneously or in sequence). In [**?**], we propose a greedy matrix algorithm to implement the pseudo-allocation process for predictive placement [**?**].

3.3 Optimizing the Load Management Policies

We discuss possible optimizations that can be used to further enhance resource utilization in a distributed MM server. It is important to ensure that optimizations implemented do not impact the basic correctness and safety of the request management (e.g. scheduling) and data management (e.g. placement) processes.

Eager Replication: Eager replication is a technique to make use of idle resources effectively to reduce overhead and provide good performance during periods of high demand. During prior information on demand for MM objects, the eager replication mechanism creates additional replicas of popular information so that they are readily available when needed. Eager replication is assigned lowest priority compared to the other load management tasks in the MM server and is initiated only during periods when it is not likely to impact MM server performance and throughput. Since replication of MM objects is a time-consuming and resource-intensive process, eager replication can significantly reduce the overheads of replication during periods of great demand for MM server resources. By doing so, eager replication can potentially not only enhance MM server utilization and throughput, but it can also lower the latency between reception of a subscriber request and servicing of that request.

Lazy Dereplication: Just as eager replication is used as an efficient method for improving server performance, we propose to employ a dereplication strategy called *lazy dereplication*. As per this strategy, when a MM object MM_i is dereplicated, the storage resources that were being taken up by MM_i are released and marked as available for other objects. However, the disk blocks that were being used for MM_i are rewritten only if there is an immediate need to reuse these blocks for storage of some other object. In the interim period, between the time dereplication is initiated and the time when the disk blocks of MM_i are overwritten, MM object MM_i exists on the data source and can be reclaimed if so desired. We term this strategy lazy dereplication because a MM object is not deleted from a data source until the time when the disk space that the object uses is required for other objects.

3.4 Characterization of Request and Replica Actor

To support the management functions, request and replica actors are augmented with annotations that represent state information that can be observed and modified by appropriate meta-level actors. This state information is subject to constraints that ensure correct overall behavior.

The MM system contains request actors in different states of admission control and execution. Constraints on the request state of a request actor are given by the state machine shown in Figure **??**. A request actor is in state

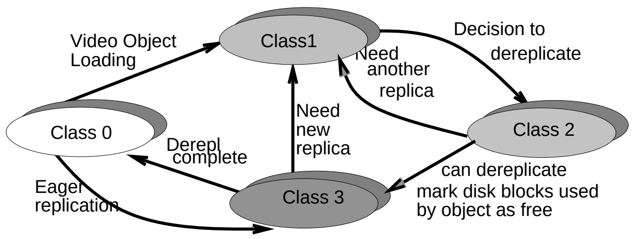

Figure 7: State Transition Diagram. The transitions depict phases in the life of a MM object

$WaitForAdm$ when request actor is initially created . As the system progresses, the value of this function will eventually change from $WaitForAdm$ to $admGranted$ or $admDenied$. From $admGranted$ it moves to $Servicing$ and then to $Completed$. After the value reaches $Completed$ or $admDenied$ it remains constant. The replica chosen to service the request is not defined until admission has been granted and once defined it remains constant.

To support replication and dereplication activities each replica of a MM object on a DS node is assigned one of four classes. $Class_0$ indicates that the replica is not present on the node. A $Class_1$ replica is guaranteed to be available. A $Class_2$ replica is marked as dereplicable but remains available until all requests assigned to it have completed. A $Class_3$ replica 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. Figure **??** depicts the transitions between the different MM object classes.

4 Specifying and 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. To analyze designs, clarify assumptions that must be met for correct operation, and establish criteria for non-interference. it is important to have a rigorous semantic model of the system: the resources, the managment processes, the application activities, and the sharing and interactions among these.

In this section we briefly describe our formalization in the TLAM framework of the multimedia meta-architecture and resource management policies presented above. We state the key theorems relating QoS requirements to QoS broker architecture and discuss reasoning about correctness and non-interference. More detail can be found in [**?**, **?**].

Properties of a system modelled 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. Such 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.

To set the stage we informally describe the notion of a system providing QoS-based MM Service ($MMService_{QoS}$). We then map QoS requirements to resource requirements and focus on modeling and reasoning about the resource management underlying a QoS-based service. We define the notions of a *Resource-based MM Service* ($MMService_{Resource}$) and of a system having *Resource Based MM Behavior* ($MMBehavior$). $MMService_{Resource}$ reflects the chosen system resource architecture and allows us to reason about the availability and use of resources. $MMBehavior$ reflects the QoS broker software architecture and models the resource management algorithms as specific meta actor behaviors. The main result is that a MM system meeting the Resource Based MM Behavior specification provides Resource-based MM Service, which in turn implies (under the assumptions of the mapping from QoS requirements to Resource requirements) that it provides 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 $MMObjects$. A MM request message specifies a triple $(\alpha_{cl}, \langle MM_0, \ldots, MM_n\rangle, QoS)$ which is interpreted as a request to initiate a MM streaming service from the server receiving the request, to the client actor α_{cl}, using the MM objects $\langle MM_0, \ldots, MM_n\rangle$, and obeying the QoS requirement QoS. To simplify the presentation, we shall assume that there is only one MM object involved in each request.

Definition 1 (QoS-based MM Service ($MMService_{QoS}$)) A system S provides a QoS-based MM Service over the set of MM objects, $MMObjects$, iff for a configuration $C \in S$, if there is an undelivered message $MMreq$, then along any path π from C there are configurations such that one and only one of the following properties hold: (1) the request is accepted for service and service is provided with the required QoS until complete, or (2) the request is rejected, for this to happen it must be the case that the requested QoS cannot be provided at the time the request arrives.

4.1 Specifying a Resource Based MM Service

We assume given a function $QoSTranslate$ which maps MM requests to resource requirements which, if met, will ensure the requested QoS (See [**?**] for examples of such QoS translation functions). The function $QoSTranslate$ maps MM requests to 4-tuples representing resource allocation requirements for the four managed resources: network bandwidth ($NetBW$), CPU cycles (CPU), disk bandwidth ($DiskBW$), and memory buffer ($BufMem$).

MM system Resource Description: MM nodes (data source, distribution controller, and client) are modelled as TLAM nodes. We assume given a function $Capacity$ such that $Capacity(DS, Res) \in Unit_{Res}$ for any data source node DS and resource Res. To provide an abstraction of the admission and placement processes, we introduce functions characterizing the state of replicas on each data source node and functions characterizing the state of each request that has arrived. The values of the given functions are constrained according to the discussion in Section 3. so that if the MM system obeys the constraints, then the service specification requirements will be met. Thus any admission and placement policies that obey the constraints can be used to implement the MM service. For example, $ReplClass(C, DS, MM)$ is the replication class of the mulitmedia object MM on node DS. The constraints on this function are given by the class transition diagram in Figure **??**. Each request is uniquely associated to a base-level request actor. This relies on the uniqueness of messages and of newly created actors in the TLAM model. $ReqState(C, \alpha^{req}) \in \{WaitForAdm, admGranted, admDenied, Servicing, Completed\}$ is the request status of request actor α^{req} which is constrained according to the diagram in Figure **??**. The functions $ReqObjId(C, \alpha^{req})$ and $ReqReplica(C, \alpha^{req})$ is the replica to which the request has been assigned (the MM object and the DS node containing the assigned replica). $ReqQoS(C, \alpha^{req})$ is the tuple of resources assigned to the request. The *total resource property* $\phi^{total}_{res}(S)$ states that for every configuration in the system and every data source node the sum of the resources allocated to the requests on the node will not exceed the total capacity of resources on the node.

Using the characterizing functions and the corresponding constraints on their values, we define a Resource-based MM service as follows.

Definition 2 (Resource-Based MM Service Specification ($MMService_{Resource}$)) A system S provides Resource-based MM service with respect to functions $QoSTranslate$, $Capacity$, and the functions characterizing replica and request state as specified above iff S obeys the constraints given for the replica and resource functions, $\phi^{total}_{res}(S)$ holds, and for $C \in S$, if there is an undelivered message, $MMreq = (\alpha_{cl}, MM, QoS)$, then along any path π from C there is one of the following segments:

(D) $C_{\mathrm{start}} \xrightarrow{+} C_{\mathrm{deny}}$
(G) $C_{\mathrm{start}} \xrightarrow{+} C_{\mathrm{grant}} \xrightarrow{+} C_{\mathrm{serve}} \xrightarrow{+} C_{\mathrm{complete}}$

such that

1. C_{start} is the result of delivery of $MMreq$ and there is a newly created request actor $\alpha^{req} \in ReqActors(C_{\mathrm{start}})$ such that:
   - $ReqClientId(C_{\mathrm{start}}, \alpha^{req}) = \alpha_{cl}$, $ReqObjId(C_{\mathrm{start}}, \alpha^{req}) = MM$,
   - $ReqQoS(C_{\mathrm{start}}, \alpha^{req})_{Res} \geq QoSTranslate(QoS)_{Res}$ for $Res \in Resources$, and
   - $ReqState(C_{\mathrm{start}}, \alpha^{req}) = WaitForAdm$.

   Request actors are only created at request delivery and correspond uniquely to MM request messages.

2. $ReqState(C_{\mathrm{deny}}, \alpha^{req}) = admDenied$, and there is a message to α_{cl} notifying of rejection of $MMreq$. In this case the system had insufficient resources to schedule $MMreq$ in C_{start}.

3. $ReqState(C_{\mathrm{grant}}, \alpha^{req}) = admGranted$, and $ReqReplica(C_{\mathrm{grant}}, \alpha^{req}) = DS$ for some DS node such that $ReplClass(C_{\mathrm{grant}}, DS, MM) = Class_1$.

4. $ReqState(C_{\mathrm{serve}}, \alpha^{req}) = Servicing$ and $ReqState(C_{\mathrm{complete}}, \alpha^{req}) = Completed$.

Theorem 1 (QoS2Resource) If a system S provides $MMService_{Resource}$, and $QoSTranslate$ satisfies the stated requirements, then the system provides $MMService_{QoS}$.

4.2 QoS Broker MM Behavior

A system with MM behavior has meta actors: QB (a QoS Broker), RS (a Request Scheduler), ROD (Replication-On-Demand), DR (Dereplication), PP (Predictive Placement), located on the DC node, and a data source manager $DSma(DS)$ on each data source node DS.

Information about resource capacity, replication, and request allocation is distributed on the data source nodes. The replica of an MM object MM on data source node DS, if present, is represented by a replica base actor $NodeReplica(C, DS, MM)$ and the request and replica state information is kept in annotations of the base actors state. For example $ReplClass(C, DS, MM) = getA(C, NodeReplica(C, DS, MM), Class)$ and $ReqQoS(C, \alpha_{ds}^{req}) = getA(C, \alpha_{ds}^{req}, QoS)$.

The QoS broker maintains, as part of its current state beh^{qb}, a model $MMstate(beh^{\mathrm{qb}})$ of the distributed MM state which is used by the resource managers to make decisions. There are analogs of the configuration based functions characterizing request and replicas in which QoS broker state plays the role of configuration. To support more sophisticated resource management processes, the QoS broker maintains a current prediction/statistics model, $Pmodel(beh^{\mathrm{qb}})$, and knowledge of what management processes are ongoing at any time, given by the $Flag$ function.

The dynamic behavior of the QoS meta actors is given by TLAM rules specifying the effects of reaction to events and messages: updating the actors state, creating new actors, sending messages, modifying base-level annotations. There are QoS broker rules for receiving requests, invoking resource management processes (request scheduling, predictive placement, dereplication, . . .), and updating the broker model in response to reports from these processes. In addition there are rules specifying the behavior of DS node meta actors that manage the replicas and local request actors.

All such broker processes are required to terminate and to accurately report effects to QB. DR can only change the class annotation of replicas and can only change class 1 to class 2. ROD and PP can change change class annotations and initiate replication (changing replication status and bandwidth). Class annotations can be moved from 0, 2, or 3 to 1. Processes that can change resource allocation are required to maintain the total resource invariant of the model they are provided with. A replication in progress must complete and we assume request servicing is a finite process.

We assume that only the MM meta actors set or modify the MM annotations of replica and request base actors. The QoS broker must coordinate the management activities sufficiently to assure that the model used is accurate. Thus we require that in any configuration, the QOS brokers model agree with the distributed state, modulo the effects of delivery of pending update notifications from the managers.

Theorem 2 If a system S has QoSBroker MM behavior as specified above, then S provides $MMService_{Resource}$.

The shared information used and modified by the resource managers— resources allocated to requests, replication status and replica class— is distributed on the DS nodes as base actor annotations. Interleaving of incremental modifications keeps the QoS brokers model sufficiently accurate to prevent interference among these processes. In fact $\phi_{res}^{total}(S)$ and the class transition constraints provide simple criteria for admitting other management processes in the interleaved setting.

A rigorous framework such as we have outlined is crucial in analyzing and preventing interference between concurrent activities. As an example of problems that arise, we might imagine that DR and RS could be allowed to proceed concurrently. An attempt to establish that the system invariants are maintained reveals the following problem. Suppose MM on DS is $Class_1$, RS is invoked with request α^{req} requiring a copy of MM and concurrently DR is invoked. Then we have the following possible scenario: DR decides to mark MM on DS as $Class_2$ and RS decides to allocate MM on DS to α^{req}; the notification from DR reaches $DSma(DS)$ first and all requests previously allocated to MM complete so MM becomes $Class_3$; then the notification from RS arrives and now we have a request allocated to a $Class_3$ replica which can be overwritten. Protocols and annotations that support increased concurrency of MM resource management are the topic of current investigations.

5 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 support the development and maintenance of large applications. Specifically, they do not deal with interactions of multiple object services executing at the same time, or the implication of composing object services. Many ORB implementations require extensions to provide and maintain architectural properties such as robustness, scalability, and the ability to support bandwidth sensitive and real-time applications. For instance, the Electra framework [?] extends CORBA to provide support for fault tolerance using group-communication facilities and protocols like reliable multicast. Various researchers have investigated architectures that provide real-time extensions to CORBA [?, ?, ?] necessary to support timing-based QoS requirements [?]. TAO is a framework that supports real-time CORBA extensions to provide end-to-end QoS and it has been used to study performance optimizations [?], event management [?], and patterns for extensible middleware [?]. Similarly, real-time method invocations have been explored by transmitting timing information in CORBA data structures [?].

The Java Development Environment is another distributed object implementation framework that transforms a heterogeneous network of machines into a homogeneous network of Java virtual machines. Java's main advantage is mobility; however the semantics of interaction with other customizations is dependent on the implementation. The ability to deal with the management of thread priorities for real-time thread management is dependent on the underlying threads implementation, making QoS support complicated to achieve. The Infospheres Infrastructure [?] uses active objects to define compositional structures for collaboration such as personal networks and sessions and develops techniques to permit customizations in this framework.

The Globe system [?] explores the construction of large scale distributed systems using the paradigm of distributed shared objects; here shared objects can be managed by appropriate subobjects. Various systems use resources on the Internet for wide-area parallel computing, e.g. Legion [?], Atlas [?]. Globus, a metacomputing framework for networked virtual machines defines a QoS component called Qualis [?] where low level QoS mechanisms can be integrated and tested. *Reflection* allows application objects to customize the system behavior [?]. The Aspect Oriented Programming paradigm [?] makes it possible to express programs where design decisions (aspects) can be appropriately isolated permitting composition and re-use of the aspect code. Research prototypes of reflective operating systems include Apertos [?] and 2K [?]. Some of the more recent research on actors has focused on coordination structures and meta-architectures [?] and runtime systems such as Broadway [?] and the Actor Foundry [?].

Multimedia QoS enforcement has been a topic of extensive research. The Omega architecture [?] developed end-to-end real-time communication protocols and QoS brokers at the endpoints to supply end-to-end QoS. *QualMan* [?] is a QoS aware resource management platform which contains a set of resource brokers that provides negotiation, admission and reservation capabilities for sharing end-system resources such as CPU, memory and network bandwidth. Work on resource management mechanisms for multimedia servers has focussed on placement of media on disk to ensure real-time retrieval [?], admission control procedures to maximize server throughput [?], buffer management policies to minimize memory requirements [?], replication and striping strategies for optimizing storage across disk arrays [?, ?], batching mechanisms that group closely spaced requests for the same objects [?], load balancing mechanisms for effective utilization [?, ?].

We are actively working on extending the existing meta-architecture to support more services. Modeling client interaction requires a notion of session and resources within a session. Further work is required to provide a generalized model that captures the architectural resources required in the server and network to support the session connection. *Dynamic negotiation protocols* involves negotiation of resources on the fly to degrade QoS of ongoing requests in order to admit more requests. In addition to providing a clear model of negotiation, this requires developing a mechanism to allow requests to decide when and how to renegotiate. 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. Also, further work is required in order to model the request migration service in the meta-architecture and develop strategies for its effective use.

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. We believe that a cleanly defined meta-architecture which supports customization and composition of such protocols is needed to support the flexible use of component based software.

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. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7(1):1–72, 1997.

[4] J. Eric Baldeschwieler, Robert D. Blumofe, and Eric A. Brewer. ATLAS: An Infrastructure for Global Computing. In *Proceedings of the The Seventh ACM SIGOPS European Workshop on Systems Support for Worldwide Applications*, 1996.

[5] M. Buddhikot and G. Parulkar. Efficient data layout, scheduling and playout control in mars. In *Proceedings of NOSSDAV'95*, pages 339–351, April 1995.

[6] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of a distributed system. *ACM Transactions on Computing Systems*, 3(1):63–75, 1985.

[7] K.Mani Chandy, Adam Rifkin, Paolo A.G. Sivilotti, Jacob Mandelson, Matthew Richardson, Wesley Tanaka, and Luke 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*, August 1996.

[8] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *ACM Multimedia Systems*, 4:112–121, 1996.

[9] Asit Dan and Dinkar Sitaram. An online video placement policy based on bandwidth to space ratio (bsr). In *SIGMOD '95*, pages 376–385, 1995.

[10] Aniruddha Gokhale and Douglas C. Schmidt. Evaluating the Performance of Demultiplexing Strategies for Real-time CORBA. In *Proceedings of GLOBECOM '97*, Phoenix, AZ, 1997.

[11] Andrew S. Grimshaw, William A. Wulf, and et al. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1), January 1997.

[12] Tim Harrison, David Levine, and Douglas C. Schmidt. The Design and Performance of a Real-time CORBA Event Service. In *Proceedings of OOPSLA'97*, October 1997.

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

[14] K. Keeton and R. Katz. The evaluation of video layout strategies on a high-bandwidth file server. In *Proceedings of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, UK*, pages 237–250, November 1993.

[15] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proceedings of ECOOP'97 European Conference on Object-Oriented Programming*, June 1997.

[16] Fabio Kon, Ashish Singhai, Roy H. Campbell, Dulcineia Carvalho, Robert Moore, and Franscisco 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.

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

[18] P. Lougher and D. Shepherd. The design of a storage server for continuous media. *The Computer Journal - Special Issue on Distributed Multimedia Systems*, 36(1):32–42, February 1993.

[19] Silvano Maffeis and Douglas C. Schmidt. Constructing reliable distributed communication systems with corba. *IEEE Communications*, 14(2), February 1997.

[20] K. Nahrstedt. *Network Service Customization: End-Point Perspective*. PhD thesis, University of Pennsylvannia, 1995.

[21] Klara Nahrstedt, Hao-Hua Chu, and Srinivas Narayan. Qos-aware resource management for distributed multimedia applications.

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

[23] Open Systems Lab. The actor foundry: A java-based actor programming environment. Available for download at http://osl.cs.uiuc.edu/foundry.

[24] Shangping Ren, Nalini Venkatasubramanian, and Gul Agha. Formalizing qos constraints using actors. In *Proceedings of Second IFIP International Conference on Formal Methods for Open Object Based Distributed Systems,FMOODS'97*, July 1997. Also post-conference book, Editors Bowman and Derrick, Published by Chapman and Hall, pp. 139-157.

[25] Douglas C. Schmidt and Chris Cleeland. Applying patterns to develop extensible and maintainable orb middleware. *Communications of the ACM*, 1998. (to appear).

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

[27] Brian C. Smith. *Reflection and Semantics in a Procedural Langauge*. PhD thesis, Massahusetts Institute of Technology, January 1982.

[28] 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.

[29] B. Thuraisingham, P. Krupp, A. Schafer, V. Wolfe, and J. Black. On Real-time Extensions to the Common Object Request Broker Architecture. In *Proceedings of the OOPSLA'94 Workshop on Experiences with CORBA*, December 1994.

[30] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming raid: A disk storage system for video and audio files. In *Proceedings of ACM Multimedia'93, Anaheim, CA*, pages 393–400, August 1993.

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

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

[33] 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.

[34] N. Venkatasubramanian, C. L. Talcott, and Gul Agha. Specifying Composable Services for QoS-Based Distributed Resource Management . In *Submitted to the ACM Symposium on Principles of Distributed Computing*, 1999.

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

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

[37] 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, January 1993.

[38] Joel L. Wolf, Philip S. Yu, and Hadas 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.

[39] Victor Fay Wolfe, John K. Black, Bhavani Thuraisingham, and Peter Krupp. Real-time method invocations in distributed environments. In *Proceedings of the HiPC'95 Intl. Conference on High Performance COmputing*, 1995.

[40] P. Yu, M.S. Chen, and D.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.

[41] J.A. Zinky, D.E. Bakken, and R.E. Schantz. Architectural support of quality of service. *Theory and Practice of Object Systems*, 3(1), 1997.