A CORBA Framework for Managing
Real-Time Distributed Multimedia Applications

V. Kalogeraki, L. E. Moser and P. M. Melliar-Smith*
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
vana@alpha.ece.ucsb.edu, moser@ece.ucsb.edu, pmms@ece.ucsb.edu

Abstract

This paper presents a real-time distributed multimedia framework, based on the Common Object Request Broker Architecture (CORBA), that provides resource management and Quality of Service (QoS) for CORBA applications. The framework consists of multimedia components and resource management components. The multimedia components produce multimedia streams and combine multimedia streams generated by individual sources into a single stream for the users. The resource management components provide QoS guarantees during multimedia transmissions based on information obtained from monitoring the usage of the system’s resources. Experimental results show that the framework provides timeliness and jitter guarantees, and synchronize the streams of multimedia data efficiently.

Keywords: CORBA, Distributed Object Computing, Middleware, Multimedia, Resource Management, Quality of Service.

1. Introduction

Real-time distributed multimedia environments have set forth new challenges in the management of processor and network resources. Multimedia applications contain text, audio, video and control information and impose strong requirements on data transmission, including fast transfer and substantial throughput. Distributed multimedia applications are characterized by potentially variable data rates and sensitivity to losses due to the transmission of data between different locations in local- or wide-area networks and the concurrent scheduling of multiple activities with different timing constraints and Quality of Service (QoS) requirements.

As multimedia technologies advance, a unified standard is required for the development of multimedia applications. The Common Object Request Broker Architecture (CORBA) [13] has become a widely accepted commercial standard for the development of distributed object applications. The key feature of CORBA is the interoperability it provides between objects programmed in different languages, running on different types of hardware architectures and operating systems, and communicating over different types of networks.

This paper presents a framework that coordinates and manages the delivery of multimedia data, based on CORBA. The framework manages the transmission of real-time multimedia data and uses current resource measurements to make efficient management decisions.

2. Overview of the Framework

2.1. Design Objectives

The CORBA framework for managing real-time distributed multimedia applications is responsible for dynamic QoS monitoring and adaptation over changing processor and network conditions. End-users receive multimedia streams from different sources without the need to know the exact location of the sources or to have specialized processors to capture the multimedia data. The framework satisfies QoS requirements expressed by the users through a combination of system design choices (e.g., assigning priority/importance metrics to the multimedia objects) and physical configuration choices (e.g., allocating memory and bandwidth).

The framework has the following design objectives:

- To reduce the cost and difficulty of developing multimedia applications by eliminating the need to address low-level implementation details
- To satisfy the QoS requirements and to meet the timing constraints specified by the users

* This research has been supported by DARPA and AFOSR, Contracts N00174-95-K-0083 and F3602-97-1-0248.
To translate the users’ QoS requirements into application-level parameters and map them into requirements for the system-level resources

To coordinate the transmission of multimedia data through the appropriate configuration of the multimedia and management components of the framework

To balance the load on the resources and to minimize system overheads by allocating the multimedia objects to the processors appropriately

To adapt to changes in the QoS dynamically by modifying the resource allocations or appropriately reallocating the multimedia objects, if necessary.

2.2. Structure of the Framework

The framework manages multimedia applications and the underlying system resources in an integrated manner. The framework consists of multimedia components for managing the transmission and delivery of multimedia data and resource management components for managing the multimedia components and monitoring the underlying system resources, as shown in Figure 1.

The multimedia components consist of Suppliers that produce streams of multimedia data, a Coordinator that receives multimedia streams from different sources and combines them into a single stream, and Consumers that receive a single stream and separate the different flows in the stream for individual playback or display.

The resource management components consist of Profilers that measure the usage of the resources, Schedulers that schedule the tasks of the multimedia objects and a Resource Manager that allocates the multimedia objects to the processors and take appropriate actions when resource requirements are violated.

The Resource Manager is implemented as a set of CORBA objects that are allocated to various processors across the distributed system and replicated to increase reliability; logically, however, there is only a single copy of the Resource Manager. The Resource Manager maintains a global view of the system and is responsible for allocating the multimedia objects to the processors.

The Resource Manager works in concert with the Profilers and the Schedulers. The Profiler on each processor monitors the behavior of the multimedia objects and measures the current load on the processor’s resources. It supplies information to the Resource Manager, which adapts the allocations over changing processing and networking conditions. The Scheduler on each processor exploits the information collected from the Resource Manager to schedule the multimedia objects on the processor.

The multimedia components of the framework are implemented as a set of CORBA objects. The Resource Manager decides the location of those objects across the system based on the utilizations of the processors’ resources, the users’ QoS requirements and the communication among the multimedia objects. The Coordinator uses a multicast group communication system to multic peace the multimedia data to the Consumers and to deliver the data to them reliably and in total order, so as to achieve synchronization of the streams [9].

3. Quality of Service for Distributed Multimedia Applications

Quality of Service (QoS) represents the set of those qualitative and quantitative characteristics that are needed to realize the level of service expected by the users. User QoS parameters (Figure 2) are translated into application-level parameters and are mapped into system-level (processor, network) parameters to control the system resources. The QoS mapping is performed by the resource management components of the framework, which enables the user to specify QoS requirements without having to map the QoS parameters into parameters for the underlying layers. The QoS parameters are expressed as \( \text{name, value} \) pairs, where the values correspond to a name range over a given set.

3.1. User QoS Parameters

To enable users to express their QoS requirements in a simple and convenient manner, the users can set their QoS parameters through a graphical user interface. User QoS parameters are specified in terms of a level of service (such as best effort or best quality) or properties that the user requires. The user must be prepared to pay a higher price when higher Quality of Service is desired. For example, a high-resolution video stream incurs a higher price in terms of increased delivery delay. User QoS parameters can be
changed dynamically when higher Quality of Service is desired.

User QoS requirements are expressed in terms of the media type (i.e., audio, video) and a set of media format parameters such as the color space or the data size (i.e., width and height) of an image, or the compression technique for the frames of a video stream. Users can also specify timing constraints such as start and end delivery times, the desired rate of transmission, the worst-case end-to-end delay and the maximum jitter. The QoS requirements specified by the user includes media-specific parameters, if additional hardware or software constraints are imposed.

3.2. Application Layer

Application QoS parameters describe the characteristics of the media requested for transfer. Some of the user’s parameters (i.e., end-to-end delay, rate of transmission) can be used directly as application QoS parameters, while others are translated into QoS parameters for the application. For example, for a video stream, the frame size is determined by the image height, width and color of an uncompressed image as specified by the user, and is computed as follows:

\[ \text{Frame size} = \text{Width} \times \text{Height} \times \text{Color resolution} \]

A multimedia application has an associated level of service metric, which is explicitly defined by the user or is determined by the resource management components of the framework based on the user’s QoS parameters and the other multimedia applications running in the system. In addition, priority metrics can be associated with the entire multimedia application or with individual frames. For example, in MPEG compression, video I-frames contain the most important information and, therefore, should have a higher priority than P-frames or B-frames. Application QoS parameters may also include media-specific information, such as the format of the video source (i.e., PAL or NTSC), the pixel data type, the compression pattern (i.e., IBP pattern for an MPEG-1 compression), the bit rate and the number of images to be skipped between captures for a video transmission. The rate of transmission can be derived from the \text{IMAGE_SKIP} parameter and the format of the video source. The maximum number of buffers determines the maximum number of images that a video card can store.

3.3. System Layer

While perception of QoS can vary from user to user and from application to application, user and application QoS requirements must be translated into system parameters in order to monitor and control the system resources.

The processor layer determines whether there are sufficient resources to accommodate the user and application requirements. Typical parameters of this layer are the utilization of the CPU, the size of the memory, the available disk space, and the load imposed by special devices used for multimedia processing. This layer also encompasses the scheduling strategy used to schedule the multimedia objects on the processors’ resources.

The network layer specifies the transport requirements for the multimedia application, including the transport protocol to be used for the delivery of packets, and packet-related parameters such as packet size, priority, ordering, transfer rate, round-trip delay, packet error and loss rate. Different multimedia streams experience random delays in the delivery of multimedia data due to the heterogeneity of the underlying communication infrastructure. Ideally, the network would deliver the multimedia data as they are generated with minimal or bounded delay.

4. Developing a Distributed Multimedia Application in CORBA

Significant limitations exist for developing multimedia applications in CORBA. CORBA was originally designed for simple request/reply interactions, rather than real-time or continuous performance-sensitive multimedia interactions. Recently, the Real-Time Platform SIG of the Object Management Group (OMG) has adopted extensions to CORBA \[14\] to support real-time.

The OMG has also recently defined a specification for the control and management of audio and video streams based on CORBA \[12\]. The CORBA A/V streaming specification defines a basic set of interfaces for implementing a multimedia framework that leverages the portability and flexibility provided by the middleware. The framework uses the components of the A/V streaming specification as building blocks for the multimedia components. During the connection establishment phase, the stream endpoints of the multimedia components and the virtual multimedia devices of the framework are defined. The A/V streaming
4.1. Multimedia Components of the Framework

Figure 3 shows the UML representation of the multimedia components of the framework. The multimedia components are based on a three-layered object structure. Multimedia suppliers and consumers are represented by the Supplier and Consumer objects, respectively. Multimedia streams that originate from different Suppliers are transmitted to a Coordinator object so that they can be multiplexed as a single stream before being forwarded to the Consumer.

A key feature of our framework is the introduction of the Coordinator between the Supplier and the Consumer objects. The Coordinator is responsible for the synchronization of the streams that the user wishes to receive so that individual buffers at the endpoints are not required.

4.1.1 The Supplier

The Supplier (Figure 4) represents the stream endpoint from which the multimedia data are derived. The Supplier defines the media to be transferred using the MMDevice interface. Typical configuration parameters of the MMDevice object are the type (i.e., video camera, microphone) or the name (i.e., ‘Star Wars’) of the media. The Supplier is implemented as a CORBA object and, therefore, can be located on any of the processors, but typically is associated with the physical location of the multimedia device. For example, to obtain live images from a camera or to listen to a live conversation, specific physical devices must be selected. On the other hand, to playback a video clip from a file, any of the processors can be chosen.

The Supplier uses the virtual multimedia device (VDev) object to configure the specific flow transfer (i.e., by setting the video format for a video transfer) and the StreamEndPoint object to define the host address where the Supplier is located.

4.1.2 The Coordinator

The Coordinator (Figure 5) multiplexes different streams originating from different sources into a single stream to be transmitted to the Consumers. Specific transport parameters are associated with the Coordinator through the StreamEndPoint interface. These parameters define the host address where the Coordinator is located and the port number to which it listens. To accommodate a large number of consumers, different Coordinator objects can be configured to receive different multimedia streams. The Coordinator is an essential component of the framework and, therefore, is replicated for fault tolerance.

4.1.3 The Consumer

The Consumer (Figure 6) receives a single stream of multimedia data from the Coordinator and separates the flows
in the stream that originate from different sources. These flows are subsequently supplied to video and audio buffers to be viewed or played, respectively. Compressed video images must be decompressed before they are displayed by the Consumer.

The Consumer is associated with the `MMDevice` interface, where multiple `VDev` objects can be created to represent the various flows that the object is expected to receive. Typical parameters of the `VDev` object are image displays and speakers. The host address where the Consumer is located is defined using the `StreamEndPoint` interface.

4.2. Stream Establishment Using the Multimedia Components

As shown in Figure 7, the process of stream establishment between a single Supplier and a single Consumer consists of the following steps:

1. To create a stream between two multimedia devices, the Coordinator instantiates a stream controller object (`StreamCtrl`) and binds it to the multimedia device objects (`MMDev`) to be used as endpoints of the stream. The Supplier of a stream is usually a multimedia device, such as a microphone or a camera, and the Consumer of a stream is typically a display. The Coordinator also describes the QoS parameters for the connection. These parameters derive from the QoS parameters specified by the user and include the type of multimedia device (i.e., video camera, video display) and the Quality of Service to be delivered.

2. A stream endpoint at a multimedia device is described by a `StreamEndPoint` interface which abstracts the transport-specific parameters (such as the name of the host and the port number of the endpoint) of the stream. The device-specific aspects of the stream are represented by a virtual multimedia device (`VDev`), which has associated configuration parameters. A multimedia device can contain different virtual multimedia devices with different characteristics, and different virtual devices can refer to the same physical device.

3. The virtual device objects of the corresponding Supplier and Consumer objects must agree to common parameters for the transfer. For example, for a video transfer, if an endpoint receives more frames per second than it can support, several of the frames must be dropped. Using references to the virtual multimedia devices, the Coordinator establishes common configuration parameters (such as the compression pattern for the video frames) between them.

4. To set up streams between the Coordinator and the two endpoints, the Coordinator object connects to the stream endpoints of the corresponding Supplier and Consumer objects. QoS parameters can be defined for the connection. Each stream endpoint can contain a number of flow endpoint objects for the individual flows within the stream. This gives greater granularity of control over stream establishment and manipulation. For instance, two flows within a stream can share the same endpoints but can differ on the encoding/decoding technique for the data.

5. The actual connection is established when the Coordinator issues `requestConnection()` calls to the corresponding Supplier and Consumer objects. Among the parameters transmitted is the transport address of the Coordinator endpoint. The Supplier and Consumer objects, using the Coordinator’s transport address, respond to the Coordinator with the transport address of the flows listening on their sides.

5. Resource Management

Multimedia applications have high resource requirements, and lack of resource management mechanisms can
lead to transmission problems with multimedia objects competing for limited unmanaged resources. To provide delivery of multimedia data, the framework employs resource management components consisting of a Profiler for each processor and a Resource Manager for the system [7]. Each Profiler continuously measures the load on its processor’s resources and monitors the behavior of the multimedia objects. The Resource Manager maintains a repository of the data obtained from the Profilers to decide how to allocate the multimedia objects to the processors and to determine which actions should be taken to provide the requested QoS.

5.1. The Profilers

The Profiler for each processor measures the current load on the processor’s resources (i.e., CPU, memory, disk) and the bandwidth being used on the communication links. Each Profiler also monitors the behavior of the multimedia objects located on its processor, in terms of the processing and communication times required for the objects to execute and communicate. The Profilers operate on a second-by-second basis, and supply their measurements as feedback to the Resource Manager.

The Profiler can signal changes in the QoS parameters and can detect overloaded resources. The QoS can change either because of an explicit request by the user (for example, when the user desires a higher level of service) or implicitly while the application executes. In both cases, the Profiler reports the monitored change of the QoS parameters to the Resource Manager which determines the action to be taken.

5.2. The Schedulers

The most popular dynamic scheduling algorithm, earliest deadline first, assigns higher priorities to tasks with closer deadlines, but this strategy does not suffice to meet the deadlines in a complex distributed system. It has been shown [4] that least laxity scheduling is an effective strategy provided that the system is not overloaded. In least laxity scheduling, the laxity of a task is defined as:

\[ \text{Laxity} = \text{Deadline} - \text{Remaining Computation Time} \]

where Deadline is the interval within which the task must be completed and Remaining Computation Time is the estimated remaining time to complete the multimedia task. The Resource Manager estimates the Remaining Computation Time to complete the multimedia application, based on its previous executions.

Each processor of the system is equipped with a Scheduler, as shown in Figure 8, that allocates the physical resources on the processor among multiple independent real-time tasks with varying timing and resource requirements [6]. The Resource Manager calculates the deadline and the Remaining Computation Time for each task, thus deriving the initial laxity for the task. The task with the least laxity is assigned the highest priority, and tasks are then scheduled according to the real-time priorities assigned by the Scheduler. If a task is delayed, its laxity diminishes and its priority increases. The invocation of a method on another processor carries the task’s laxity with it, yielding a system-wide scheduling strategy that requires only a small amount of local computation.

5.3. The Resource Manager

The Resource Manager is responsible for coordinating the allocation of the resources and for deciding whether the user’s QoS requirements can be met. Admission control is essential to determine whether a new request can be admitted without degrading the performance of the currently running multimedia applications. Once a user’s request for service is accepted, the Resource Manager selects the most appropriate values of the QoS parameters in order to satisfy the user’s requirements, taking into account the available resources and the other multimedia applications running in the system. Critically important is the ability to use system information, as obtained from the Profilers, to make accurate decisions that increase the likelihood of meeting the QoS requirements specified by the user. For example, if a service requested by the user is available on more than one processor, the Resource Manager selects the service from the most appropriate processor, e.g., the least-loaded processor or the processor located closest to the sender.

The Resource Manager maintains a system profile of the physical configuration, including the various resources along with their specific characteristics (processing speed, memory size, network bandwidth, disk capacity, etc). As new requests for multimedia services are introduced, the Resource Manager determines whether it can satisfy those requests. The Resource Manager translates the QoS properties specified by the user into QoS parameters at the individual layers, as described in Section 3. Once the Re-
source Manager accepts a user’s request, it translates the QoS requirements into application QoS parameters and then reserves and allocates the necessary resources at the nodes along the path between the sender and the receiver.

The Resource Manager is implemented as a set of CORBA objects that are allocated to various processors across the distributed system. The Resource Manager objects are replicated for fault tolerance; however, logically, there is only a single copy of the Resource Manager.

5.4. Dynamic QoS Negotiation and Adaptation
During operation, the Profilers may detect overloaded or insufficient resources to provide the Quality of Service required by the user. If the Profilers report violations of the Quality of Service for some multimedia application, the Resource Manager can initiate negotiation with the user so that alternative QoS parameters can be selected. The QoS parameters can be described in terms of qualitative metrics, such as level of service and level of importance, or quantitative metrics, such as rate of transmission and end-to-end delay. Negotiation, however, entails substantial overhead, which may not be tolerated for real-time applications.

When insufficient resources remain to provide the required Quality of Service, the Resource Manager gradually degrades the Quality of Service for certain multimedia applications. The applications chosen are the ones with the least importance to the system, as defined by the user or computed by the Resource Manager. Alternatively, the Resource Manager attempts to reallocate the multimedia objects dynamically by migrating them to other processors. This dynamic reallocation may free some computing resources and enable the remaining objects to operate. Dynamic reallocation may also be required if a processor or other resource is lost because of a fault or if a multimedia application is not meeting its deadlines. If the quality of the multimedia applications continues to deteriorate, the Resource Manager can drop the least important multimedia applications so that the remaining applications can be accommodated at their desired level of service.

6. Implementation and Experimental Results

6.1. Prototype Implementation
The platform for our implementation and experiments consisted of six 167 MHz Sun ULTRASparscs running Solaris 2.5.1 with the VisiBroker ORB over 100 Mbit/s Ethernet. For the implementation, we used three Supplier objects, two of which transmit live images captured from different cameras, while the third reads a video clip stored in a file. The Supplier objects are located on the processors equipped with the cameras, while the third Supplier object is located on a different processor. Each Supplier object transmits its data stream to a Coordinator object located on a different processor. The Coordinator waits until it receives all of the data streams sent by the Supplier and merges them into a single stream, which it then transmits to the Consumers. The three Consumers receive data streams from the Coordinator and display each of the individual flows within the data stream on a separate display. Figure 9 shows the multimedia application at run-time.

The implementation uses the XIL Imaging Library for image processing and compression. XIL provides an object-oriented architecture and supplies a rich set of basic image processing functions that serve as building blocks for developing complex image processing applications. The implementation also uses the SunVideo subsystem, a real-time video capture and compression subsystem for Sun SPARC-stations. The SunVideo subsystem consists of a SunVideo card, which is a digital video card, and supporting software, which captures, digitizes and compresses unmodulated NTSC and PAL video signals from video sources. MPEG-1 video coding is used for image data compression. The compressed video streams are transmitted over the network, stored on disk, and decompressed and displayed in a window on a workstation.

6.2. Performance Measurements
In our experiments we measured the end-to-end delay experienced by the video frames, i.e., the delay between the time a video frame is captured from the camera at a Supplier, until the time it is transferred to a Consumer and displayed to the user. Of particular interest was the jitter associated with the random delays of the individual frames. Ideally, frames should be displayed continuously with a fixed frame rate. In a distributed system where multiple participants compete for resources, the delays experienced by the various frames varies. Typically, the jitter is eliminated by employing a buffer at the receiver. For example, if the receiver knows a
priori the maximum possible end-to-end delay experienced by a video frame, it can buffer the first frame for this maximum time before displaying it. Alternatively, the display time of the frames at the Consumer can be varied.

In our framework, the end-to-end delay experienced by the video frames depends on the following factors: (1) the time required for the Suppliers to capture the frames from the camera devices and compress them into a compressed frame sequence, (2) the time to transmit the compressed frame sequence from the Suppliers to the Coordinator, (3) the time required for the Coordinator to collect the compressed frame sequences from the individual Suppliers and combine them into a single stream, (4) the time to transmit the single stream from the Coordinator to the Consumers, and (5) the time required for the Consumers to separate the compressed frame sequences from the different Suppliers and decompress them for display.

To compute the end-to-end delay, we assume that the compression/decompression time of frames of the same size is approximately the same; thus, the end-to-end delay is mainly a function of the delay in the transmission of compressed frame sequences from the Suppliers to the Consumers and the delays in the collection of compressed frame sequences from different sources. Figure 10 shows the rates at which the frames are captured and displayed at the Suppliers and Consumers. Our measurements indicate that the frames are captured and displayed at the same rate at both the Suppliers and Consumers, as the introduction of the Coordinator did not result in any irregularity in the compressed frame sequence transmissions and did not introduce any additional delay in the transmissions. Figure 11 shows the delay in the transmission of frames as a function of time, with the load on the processor increasing over time. As the load on the processor increases, the delay becomes larger and more variable.

Figure 10. Frame capture and display rates.

Figure 11. Delay (in ms) for successive frames as the load on the processor is increased.

For our application, the jitter is the variance in the end-to-end delay of consecutive compressed frame sequences when transmitted from the Suppliers to the Consumers. The jitter depends on the load on the processors on which the Suppliers and the Consumers are located. To demonstrate the effect of the processor load on the jitter, we introduced a random increase in the load of the processor on which the Supplier was located and the frames were captured.

We measured the jitter obtained when both a single Supplier and two Suppliers were used to transmit live images to the Consumer. Figure 12 shows that the jitter is larger for a single Supplier than for two Suppliers and that it increases unacceptably when the load on the processor is increased above 0.5. If the load on the processor exceeds 0.5, then the object is migrated to another processor.

7. Related Work

The field of distributed multimedia has recently been the focus of much research [18]. McGrath et al [8] and Mungee et al [10] have developed audio/video streaming frameworks based on the Object Management Group’s A/V Streams model. Their objectives are to support multiple transport protocols and flexible stream controls and to provide multiple concurrent strategies when creating stream endpoints. To demonstrate the flexibility and efficiency of their framework, Mungee et al have implemented a distributed application that uses TAO’s A/V streaming framework [10] to establish and control MPEG video streams. Our prototype can be hosted on top of both of these implementations and can use them during the stream establishment phase to define the multimedia devices and stream endpoints for the connections.
JITTER is on host and network resources that provide mechanisms to manage multimedia computing and communication. Their emphasis on management mechanisms to guarantee end-to-end delivery for multimedia applications with different resource requirements experienced at the host and network levels when executing multimedia applications. Alfano et al [1] have proposed a distributed multimedia system with resource management capabilities. They have defined a generic QoS Management Information Base, which consists of information objects that represent a set of layered QoS parameters, organized into four logical groups: service, application, system, and network. They have also defined a set of QoS management services for monitoring and controlling QoS-related resources, in the context of a distributed multimedia system, called MAESTRO. Unlike MAESTRO, our approach is not focused on building a distributed multimedia collaboration system, but rather, on coordinating and managing the delivery of real-time multimedia data for the development of distributed multimedia applications.

Le Tien et al [17] have proposed a lightweight model that provides end-to-end QoS support for distributed multimedia applications. Similar to our approach, their model uses μ-QoS and resource managers responsible for the QoS mapping and monitoring of the applications and for managing the QoS of specific resources, including admission control, real-time scheduling and monitoring. Their main focus, however, is the structure of the resource manager and the use of their implementation in the context of a video on demand application.

Several researchers have recognized the need to enhance distributed multimedia systems with resource management capabilities. Alfano et al [1] discuss the problems they experienced at the host and network levels when executing a multimedia application with different resource requirements. Nahrstedt et al [11] have employed resource management mechanisms to guarantee end-to-end delivery for multimedia computing and communication. Their emphasis is on host and network resources that provide mechanisms to allocate and control resources for multimedia data transmission and to adapt when system resource overloading occurs. The difference between our work and these two approaches is that we leverage the flexibility provided by the CORBA middleware and benefit from the streaming mechanisms.

Waddington et al [19] have developed distributed multimedia components that extend the CORBA and DCOM architectures to provide additional mechanisms and abstractions for continuous media stream interactions. Unlike us, they have implemented their own stream interfaces to encapsulate and abstract the functionality of multimedia devices and processing entities. They have also developed resource management components that monitor QoS dynamically to support complex distributed multimedia applications over heterogeneous networks and end-systems. Szentiványi et al [16] have provided foundation objects that define different aspects of multimedia information management. Their work differs from ours in that their focus is on the implementation of standardized CORBA services and the integration with their model COMMOTION.

Other researchers have observed that resource management mechanisms are more efficient when information about the applications is known in advance. Chatterjee et al [3] have proposed a model called LASM to capture the structure of distributed applications, their resource requirements, and relevant end-to-end QoS parameters. The difference between LASM and our approach is that LASM conveys application-specific information to the resource manager to structure the applications and allocate the resources initially. When a change in the system state occurs, they reallocate and restructure the applications dynamically. In contrast, our Profilers monitor the behavior of the multimedia components at run-time and record the method invocations to detect any violations in the QoS parameters or to discover overloaded resources so that the allocations can be revised accordingly.

8. Conclusion

We have designed a framework for real-time distributed multimedia applications based on CORBA. The multimedia components of the framework consist of Suppliers that produce streams of multimedia data, a Coordinator that manages the streams generated by the individual Suppliers and combines them into a single stream, and Consumers that receive the single stream of multimedia data and separate the flows within the stream to be viewed or played individually. The resource management components of the framework consist of Profilers that monitor the usage of the resources and the behavior of the application objects, Schedulers that schedule the tasks of the multimedia objects, and a Resource Manager that allocates the multimedia objects to the processors, sharing the resources efficiently.
and adapting the allocations over changing processing and network conditions. Performance measurements show that our framework components provide timeliness, jitter and QoS guarantees, and synchronize the streams of multimedia data efficiently.

References


