Service-Oriented Architecture for Building a Scalable Videoconferencing System

Ahmet Uyar[1,2], Wenjun Wu[2], Hasan Bulut[2], Geoffrey Fox[2]
[1]Department of Electrical Eng. & Computer Sci. Syracuse Unv.
[2]Community Grids Lab, Indiana University
{auyar, wewu, hbulut, gcf}@indiana.edu

Abstract

The availability of increasing network bandwidth and the computing power provides new opportunities for videoconferencing systems over Internet. On one hand, broadband Internet connections are spreading rapidly. Even cell phones will have broadband internet access in the near future with the implementations of 3G standards. On the other hand, the usage of webcams and video camera enabled PDAs and cell phones are increasing by many millions every year. This requires universally accessible and scalable videoconferencing systems that can deliver thousands of concurrent audio and video streams. In addition to audio and video delivery, such systems should provide scalable media processing services such as transcoding, audio mixing, video merging, etc. to support increasingly diverse set of clients.

However, developing videoconferencing systems over Internet is a challenging task, since audio and video communications require high bandwidth and low latency. In addition, the processing of audio and video streams is computing intensive. Therefore, it is particularly difficult to develop scalable systems that support high number of users with various capabilities. Current videoconferencing systems such as IP-Multicast and H.323 can not fully address the problem of scalability and universal accessibility. These systems designed to deliver the best performance and lacks flexible service oriented architecture to support increasingly diverse clients with various network and device capabilities. We believe that with the advancements in computing power and network bandwidth, more flexible and service oriented systems should be developed to manage audio and video conferencing systems. In this paper, we propose a service oriented architecture for videoconferencing, GlobalMMCS, based on a publish/subscribe event brokering network, NaradaBrokering.

***Keywords**: service oriented architecture, videoconferencing, publish/subscribe systems.*

1 Introduction

The availability of increasing network bandwidth and the computing power provides new opportunities for distant communications and collaborations over Internet. On one hand, broadband internet connections are spreading rapidly. Even cell phones will have broadband internet access in the near future with the implementations of 3G standards. On the other hand, the usage of webcams and video camera enabled PDAs and cell phones are increasing by many millions every year. Therefore, it is not inconceivable to imagine that the trend in the increasing usage of videoconferencing systems will continue. This will require universally accessible and scalable videoconferencing systems that can deliver thousands or tens of thousands of concurrent audio and video streams. In addition to audio and video delivery, such systems should provide scalable media processing services such as transcoding, audio mixing, video merging, etc. to support increasingly diverse set of clients.

However, developing videoconferencing systems over Internet is a challenging task, since audio and video communications require high bandwidth and low latency. In addition, the processing of audio and video streams is computing intensive. Therefore, it is particularly difficult to develop scalable systems that support high number of users with various capabilities. Current videoconferencing systems such as IP-Multicast [1] and H.323 [2] can not fully address the problem of scalability and universal accessibility. These systems designed to deliver the best performance and lacks flexible service oriented architecture to support increasingly diverse clients with various network and device capabilities. We believe that with the advancements in computing power and network bandwidth, more flexible and service oriented systems should be developed to manage audio and video conferencing systems.

The first step when building a videoconferencing system is to analyze and identify the tasks performed in videoconferencing sessions. Then, independently scalable components can be designed for each task. It is also important to coordinate the interactions among these components in an efficient and flexible manner to add new services and computing power when necessary. We identified that there are three main tasks performed in videoconferencing sessions: audio/video distribution, media processing and meeting management. We proposed using a publish/subscribe event brokering system as the audio and video distribution middleware [3]. In this paper, we propose a service oriented architecture to develop a videoconferencing system, GlobalMMCS [4], that is scalable, flexible and universally accessible, based on a publish/subscribe event brokering network, NaradaBrokering [5, 6, 7].

The content of this paper is organized as follows. First, we analyze the tasks performed in videoconferencing sessions to determine the criteria to develop videoconferencing systems. In the next two sections, we give an overview of this architecture and a brief summary of NaradaBrokering. In following sections, we provide the details of messaging mechanisms and service distribution framework in this system. We evaluate other videoconferencing systems briefly in related work section before we conclude the paper.

2 Task Analysis in Videoconferencing Systems

There are three main tasks performed in videoconferencing sessions on server side.

1. Audio/video distribution: This includes transferring audio and video streams from source clients to destinations in real-time. This is a challenging task, since those streams require high bandwidth and low latency. ITU recommends [8] that the mouth-to-ear delay of audio should be less than 300ms for good quality communication. Therefore, it is essential to provide an efficient media distribution mechanism that will route media streams through best possible routes from sources to destinations. Otherwise, unnecessary network traffic might be generated and additional transit delays might be added. In addition, audio and video streams should be replicated only when it is needed along the path from sources to destinations. This saves significant bandwidth and provides scalability. The sender publishes one copy of a stream and the distribution network delivers it to all participants by replicating it whenever necessary. Thirdly, since audio and video streams are composed of many small sized packages, minimum headers should be added to all packages. Otherwise, there can be substantial increase in the amount of data transferred. Lastly, users should be able to receive a stream with various transport protocols.

2. Media Processing: Media processing is another very important task performed in videoconferencing sessions on server side. Although in a homogenous videoconferencing setting, where all users have high network bandwidth and computing power, media processing might not be necessary at server side, it is crucial in videoconferencing sessions which have users with various network and device capacities. For example, AccessGrid [9] provides room based group-to-group videoconferencing services to multicast enabled high bandwidth sites that can receive/send/display tens of audio/video streams concurrently. They do not provide any media processing services. However, videoconferencing systems that aim to support diverse set of users with various network bandwidths and endpoint capabilities must provide media processing services to customize the streams according to the requirements of users. Some users might have very limited network bandwidth. For those users, multiple audio and video streams should be mixed to save bandwidth, or some streams should be transcoded to produce low bandwidth streams. Some other users might have limited display or processing capacity. For those users, multiple video streams can be merged or larger size video streams can be downsized.

Media processing usually requires high computing resources and real-time output. Therefore, they can limit the scalability of a videoconferencing system severely when implemented poorly. More importantly, they can affect the quality of audio and video distribution if they share the same computing resources with media distribution units. Therefore, the media processing units should be separated completely from the media distribution units to provide scalability. In addition, it should be possible to add new computing resources dynamically to support high number of sessions with more users. Moreover, a flexible media processing framework should be designed to allow the implementation of new media processing services.

3. Session management: Session management includes starting/stopping/modifying videoconferencing sessions. It also includes determining and assigning system resources for these sessions. For example, it includes finding out the right audio mixing unit to be used by a meeting. In addition, it includes the mechanisms for participants to discover/join/leave sessions. Contrary to the media distribution and media processing tasks, session management requires little bandwidth and computing resources. However, it is very important to coordinate and distribute the tasks in such sessions. Therefore, it is crucial to design a flexible and scalable session management mechanism.

3 GlobalMMCS Architecture

Global Multimedia Collaboration System (GlobalMMCS) is designed to provide scalable videoconferencing services to a diverse set of users. The architecture is flexible enough to support users with various network bandwidth requirements and endpoint capabilities. It supports users behind firewalls, NATs, and proxies. It also allows the system to grow or shrink dynamically by adding or removing computing resources.

There are three main components of this architecture (Figure 1): media and content distribution network, media processing unit and meeting management unit. NaradaBrokering event broker network is used to deliver both media and data packages. It provides a unified scalable middleware for all communications. We provided the rationale to use a publish/subscribe middleware to use for real-time audio/video delivery in [3]. We also give a brief overview of NaradaBrokering in this paper. The architecture separates media processing from media distribution completely to provide a flexible and scalable system.

There are many types of service providers in this system. *MediaServers* provide media processing services such as audio mixing, video mixing and image grabbing. *MeetingManagers* provide meeting management services such as starting and stopping audio and video sessions. *AudioSession* and *VideoSession* components provide user join and leave services to meeting participants. We provide a unified framework to manage the interactions among system components and distribute service providers. We avoid centralized solutions to provide fault tolerance and location independence. Addition and removal of service providers are handled dynamically to allow the system to grow or shrink. The service provider distribution framework provides the mechanisms to discover and select service providers, and execute tasks.

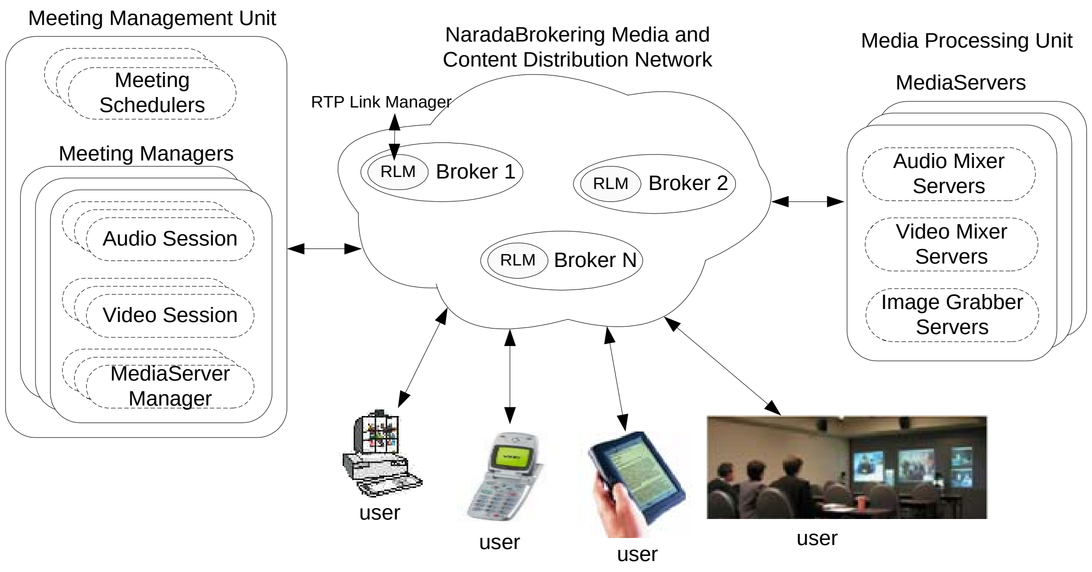

Figure 1 GlobalMMCS Architecture

4 NaradaBrokering

NaradaBrokering [5, 6, 7] is a distributed publish/subscribe messaging system that provides scalable architecture and an efficient routing mechanism. It organizes brokers in a cluster-based hierarchy. The smallest unit of the messaging infrastructure is the broker. Each broker is responsible for routing messages to their next stops and handling subscriptions. In this architecture, a broker is part of a base cluster that is part of a super-cluster, which in turn part of a super-super-cluster and so on. Clusters comprise strongly connected brokers with multiple links to brokers in other clusters, ensuring alternate communication routes. This organization scheme results in the average communication “path lengths” between brokers that increase logarithmically with geometric increases in network size, as opposed to exponential increases in uncontrolled settings.

Each broker keeps a broker network map of its own perspective to efficiently route the messages to their destinations with a near optimal algorithm [6]. Messages are routed only to those routers that have at least one subscription for that topic. This prevents unnecessary message traffic on the system. Messages are duplicated on brokers when they are to be sent to more than one destination. This saves significant bandwidth when delivering audio and vide streams. Moreover, messages are routed only to the intended destinations and they are prevented from being routed back to the producers.

NaradaBrokering has a flexible transport mechanism [10]. Its layered architecture supports addition of new protocols easily. In addition, when a message traverses through broker network, it can go through different transport links in different parts of the system. A message can be transported over HTTP while traversing a firewall but later TCP or UDP can be used to deliver it to its final destinations. Therefore, it provides a convenient framework to go through firewalls and support clients with differing transport needs.

Another important feature of NaradaBrokering is the performance monitoring infrastructure [11]. The performance of the links among brokers is monitored and problems are reported on real-time. In addition, NaradaBrokering supports dynamic broker and link additions and removals, so that the broker network can grow or shrink dynamically.

Since NaradaBrokering provides JMS compliant publish/subscribe messaging service, it can also be used to deliver the reliable messages among the distributed components in the system. It can be used to deliver the messages for real-time collaboration applications [12] such as chat, file sharing, application sharing, display sharing, etc. Therefore, NaradaBrokering provides a unified content delivery mechanism that simplifies the design and management of the videoconferencing system significantly.

On the other hand, publish/subscribe systems in general and NaradaBrokering in particular are not designed to deliver real-time audio and video streams. Therefore, we made some additions to better support audio and video transfer [3].

A. We added an unreliable transport protocol (UDP) to the transport layer.
B. We added a compact message type which adds 14 bytes headers to packages. This process entailed the implementation of a distributed unique id generation mechanism with 8 bytes long.
C. We implemented proxies for legacy RTP clients and multicast groups.
D. We made some changes in the routing algorithm of NaradaBrokering. We gave priority to audio package delivery [13] since audio communication is the fundamental part of a videoconferencing system. We also modified the routing algorithm [14], so that minimum delay is added to packages that are traveling to other brokers in the system.

4.1 Performance Tests of NaradaBrokering

We conducted extensive tests to evaluate the performance of NaradaBrokering broker network in the context of audio and video stream delivery. We investigated both the performance of a single broker and the performance of the broker network. We presented the results of the single broker tests in [13] and the results of the broker network tests in [14]. These tests demonstrated that a single broker can support up to 400 participants both in single large size meetings and multiple smaller size meetings with very good quality audio and video delivery. Therefore, a small size organization can deploy this system with one broker.

The broker network tests showed that the capacity of the broker network can be increased significantly by adding new brokers. Having multiple brokers increases the quality of the stream delivery considerably by providing smaller latency, jitter and loss rates. These performance tests with multiple brokers demonstrated that the number of supported participants can be increased linearly in large size meetings by adding new brokers. While one broker supported up to 400 participants in one large size meeting, 4 brokers supported up to 1600 participants. On the other hand, the behavior of the broker network is more complex when there are multiple concurrent meetings compared to having a single meeting. Having multiple meetings provide both opportunities and challenges. If the sizes of meetings are very small and the clients in meetings are scattered around the brokers, then the broker network can be utilized poorly. Inter-broker stream delivery can reduce the number of supported users. The best broker utilization is achieved when there are multiple streams coming to a broker and each incoming stream is delivered to many receivers. If all brokers are utilized fully in this fashion, multi broker network provides better services to higher number of participants. Our tests showed that 4 brokers can support up to 72 video meetings each having 20 users, 1440 users in total. A similar test with a larger size meeting showed that the same four brokers can support 48 meetings each having 40 users, 1920 users in total.

In summary, the broker network provides very good audio and video delivery services. It can be configured both for small and large size organizations with brokers distributed geographically.

5 Messaging Among System Components

We use NaradaBrokering-JMS [15] publish/subscribe system to distribute the control messages exchanged among various components in the system. This simplifies building a scalable solution, since messages can be delivered to multiple destinations without explicit knowledge of the publisher. Service providers can be added dynamically. Moreover, it provides location independence for each component, since a component is only connected to one broker and it exchanges all its data and media messages through this broker. In addition, using the same middleware for both data and media delivery reduces the overall system complexity considerably.

JMS [16] provides a group communication medium. It uses topics as the group address. When a message is published on a topic, all subscribers of that topic receive that message. In our system, while some messages are sent to a group of destinations, some others are destined to one target. Therefore, an efficient and scalable message exchange mechanism should be designed among system components. Messages should only be delivered to intended destinations. In addition, topics should be organized in an orderly fashion.

First, we will examine the various messaging types that take place in our system. Then we will provide the topic naming convention to handle these messaging types.

5.1 Messaging Semantics

There are three different messaging types in this videoconferencing system:

1. Request/Response messaging: This messaging semantic is used when a consumer requests a service from a service provider in the system. It sends a request message to the service provider to execute a service. The service provider processes the received message and sends a response message back to the sender. Since both the request and response messages are destined to one entity, it is important not to deliver these messages to unrelated components. Therefore, all service providers and consumers should have unique topics to receive messages destined to them only.

2. Group messaging: This messaging semantic is used when an entity wants to send a message to a group of entities in the system. It publishes a message to a shared topic and all group members receive it. In some cases, receiving components send a response message back to the sender. In some other cases, no response message is assumed. There are two types of applications of this messaging semantic in our system. First one is to discover service providers. An entity sends a request message to the group address of some service providers. Then, each one of them sends a reply message including the information asked. Another application is to execute a service on a group of service providers. In this case, an entity sends a service execution request message to the group address, and all service providers in that group execute that service.

3. Event based messaging: Event based messaging is used when an entity wants to receive messages from another entity regarding the events happening on that component during a period of time, such as over the course of a meeting. All interested entities subscribe to the event topic and receive messages as the publisher posts them. A typical application of this event based messaging in our system is to deliver events related to audio and video streams. All participants subscribe to the event topic and monitoring service publishes the events as they happen.

5.2 Topic Naming Conventions

To meet the requirements of the messaging semantics explained above, two types of topics are needed; group topics and unique component topics. We use a string based directory style topic naming convention to create topic names in an orderly and easy to understand fashion. All topic names start with a common root. We use our project name as the root name GlobalMMCS. However, it is possible for an institution to change this root name and all topic names change accordingly. This lets installing more than one copy of this system on the same broker network. Group topic names are constructed by adding the component name to the root by separating with a forward slash. Groups are formed by the multiple instances of the same components. For example, all instances of *MediaServers* running in the system belong to the same group.

- GlobalMMCS/MeetingManager
- GlobalMMCS/AudioSession
- GlobalMMCS/VideoSession
- GlobalMMCS/MediaServer
- GlobalMMCS/RtpLinkManager

These strings are used as the component group addresses. For example, all *AudioSession* objects listen on GlobalMMCS/AudioSession topic to receive messages which are destined to all *AudioSession* objects. Similarly, all other objects listen on their group addresses to receive group messages.

Unique component topic names are constructed by adding a unique id to these component group addresses:

- GlobalMMCS/AudioSession/<sessionID>
- GlobalMMCS/VideoSession/<sessionID>
- GlobalMMCS/MediaServer/<serverID>
- GlobalMMCS/RtpLinkManager/<brokerID>

These unique topic names are used to communicate directly with a component. The messages sent to these topics only received by the component which has that id. When an instance of a component is initiated, it gets an id from the broker it is connected. Then it constructs its private topic name by following the above structure and starts listening on that topic for the messages destined to it. In addition to using the component id for constructing a private topic name, this id is also used to identify components from others in the system.

One of the additions which we made to NaradaBrokering is the mechanism to generate unique ids on time and space. A unique id generator runs in every broker and it can generate an id for every millisecond. This id will be unique for 557 years. Each broker generates unique ids without interacting with any other broker.

Sometimes a component communicates with many different components; in that case, we use extra one more layer to distinguish these communication channels:

- GlobalMMCS/AudioSession/<id>/RtpLinkManager
- GlobalMMCS/AudioSession/<id>/AudioMixerServer
- GlobalMMCS/AudioSession/<id>/RtpEventMonitor

In the above example, an *AudioSession* component communicates with three different entities: *RtpLinkManager*, *AudioMixerServer* and *RtpEventMonitor*. It uses different topics for each component. Using different topics simplifies logging and detecting the problems. It also simplifies developing codes to handle various types of messages exchanged with each component.

With this naming convention, we provide a unified mechanism to generate group and individual component topic names. It is easy to understand and debug.

6 Service Distribution Framework

In our system, we support multiple copies of the same service providers in a distributed fashion. Since, there are many types of service providers; we provide a unified framework (Figure 2) for distributing them. We assume that distributed copies should be able to run both in a local network and in geographically distant locations.

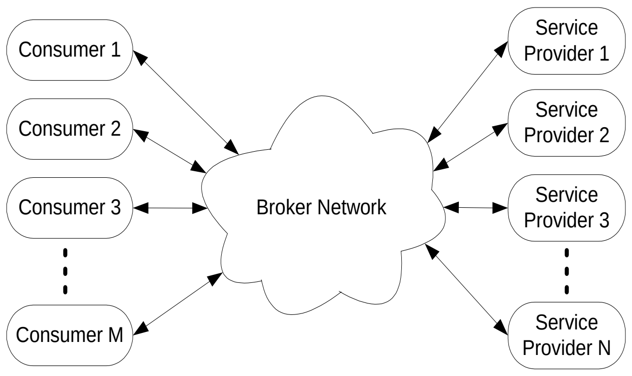

Figure 2 Service distribution model

As we mentioned above, each service provider and the consumer is assigned a unique id. This id is used both to identify an instance of this component from others and to generate its unique topic name to communicate with others in the system. A service provider listens on two topics. One is the service provider group topic on which it receives messages destined to all service providers. Another is its private topic on which it receives messages sent only to itself.

6.1 Service Discovery

Instead of using a centralized service registry for announcing and discovering services, we use a distributed dynamic mechanism. One problem with centralized registry is the failure susceptibility. Another difficulty is that since in our system the status of the service providers change dynamically, it is not reasonable to update a centralized registry frequently.

In this approach, a consumer sends an *Inquiry* message to the service provider group address. In this message, it includes its own topic name, so that service providers can send the response message back to it only. When service providers receive this message, they respond by sending a *ServiceDescription* message, in which they include the current status of that service provider. The information provided in this *ServiceDescription* message depends on the nature of the service being provided. But, it must be helpful for the consumer to select the service provider to ask for the service. The consumer waits for a period of time for responses to arrive, and evaluates the received messages. Since a consumer does not know the current number of the service providers in the system, after waiting for a while it assumes that it received responses from all the service providers.

6.2 Service Selection

When a consumer receives *ServiceDescription* messages from service providers, it compares the service providers according to the service selection criteria set by user. This criteria can be as simple as checking the CPU loads on host machines and choosing the least loaded one or it can take into account more information and complicated logic. For example, users can be given an option to set the preferences over the geographical location of the service providers. This can be particularly useful for systems that are deployed worldwide.

6.3 Service Execution

When the consumer selects the service provider on which it intends to run its service, it sends a *Request* message to the service provider for the execution of the service. If the service provider can handle this request, it sends an *Ok* message as the response. Otherwise, it sends a *Fail* message. In the case of failure, the consumer either starts this process from the beginning or tries the second best option. A service can be terminated by the consumer by sending a *Stop* message.

In our system, a service is usually provided for a period of time, such as during a meeting. Therefore, the consumer and the service provider should be aware of each others continues existence during this time period. Each of them sends periodic *KeepAlive* messages to the other. If either of them fails to receive a number of *KeepAlive* messages, it assumes that the other party is dead. If the consumer is assumed dead, then the service provider deletes that service. If the service provider is assumed dead, then consumer looks for another alternative.

In our system, each service provider is totally independent of other service providers. Namely, service providers do not share any resources. Therefore, there is no need to coordinate the service providers among themselves. This simplifies the distribution and management of service providers significantly.

6.4 Advantages of this Framework

Fault tolerance: There is no single point of failure in the system. Even though some components may fail, others continue to provide services.

Scalability: This model provides a scalable solution. There is no limit on the number of consumers to support as long as there are service providers to serve them. The fact that initially a consumer sends a message to all service providers, and they all respond back to the consumer, may limit the number of the supported service providers. However, this can be eliminated by limiting the number of service providers who respond to an *Inquiry* message. This selection can be based on the location of the service providers or some other criteria depending on the nature of the services provided. For example, already fully loaded service providers might ignore inquiry messages.

Location independence: All service providers are totally independent of other service providers and all consumers are also independent of other consumers. Therefore, a service provider or a consumer can run anywhere as long as they are connected to a broker.

7 Media Processing

We provide media processing services at server side to support a diverse set of clients. Some clients have limited network bandwidth, processing and display capacity. Either they can not receive multiple audio and video streams or they can not process and display them. Therefore, server side components should generate combined streams for them. The services which we have implemented include audio mixing, video mixing and image grabbing. We also have an RTP stream monitoring service. All these services require real-time processing and usually high computing resources.

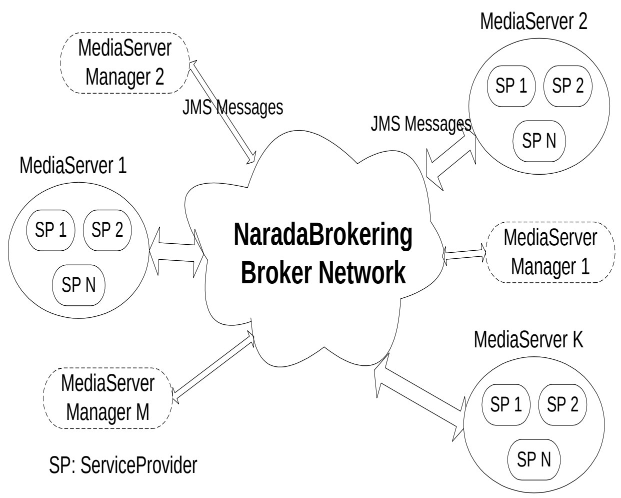

Figure 3 Media Processing Framework

Media processing framework (Figure 3) is designed to support addition and removal of new computing resources dynamically. A server container, *MediaServer*, runs in every machine that is dedicated for media processing. It acts as a factory for service providers. It starts and stops them. In addition, it advertises these service providers and reports the status information regarding the load on that machine. All service providers implement the interface required by the server container to be able to run inside. Each *MediaServer* is independent of other *MediaServers* and new ones can be added dynamically.

Currently, there are three types of service providers for media processing: *AudioMixerServer*, *VideoMixerServer*, and *ImageGrabberServer*. More service providers can be added by following the guidelines and implementing the relevant interfaces. These service providers can either be started from command line when starting the service container, or they can be started by using the *MediaServerManager*. *MediaServerManager* implements the semantics to talk to *MediaServers*.

7.1 Audio Mixing

AudioMixerServer provides audio mixing services for a meeting, *AudioMixerSession*. An *AudioMixerServer* can have any number of audio mixers as long as the host machine can handle. Each speaker is added to the mixer as they join the meeting, and special mixed streams are constructed for them. An audio mixer receives the streams from the broker network and publishes the mixed streams back on the broker network. Clients receive the mixed streams by subscribing to the mixed stream topics.

While some audio codecs are computing intensive, some others are not. Therefore the computing resources needed for audio mixing change accordingly. Audio mixing units need to have prompt access to CPU when they need to process received packages. Otherwise, some audio packages can be dropped and result in the breaks in audio communications. Therefore, the load on audio mixing machines should be kept at as low as possible.

Table 1. Audio mixer performance test

Number of mixers	CPU usage %	Memory usage (MB)	Quality
5	12	36	No loss
10	24	55	No loss
15	34	73	No loss
20	46	93	Negligible loss

We have tested the performance of an *AudioMixerServer* for different number of mixers on it. There were 6 speakers in each mixer. Two of these speakers were continually talking and the rest of them were silent. There were also one more audio stream constructed which had the mixed stream of all speakers. Therefore, 6 streams were coming into the mixer and 7 streams were going out. All streams were 64kbps ULAW. Mixers were receiving the streams from a broker and publishing the output streams back on the broker. The machine that was hosting the mixer server was a winXP machine with 512 MB memory and 2.5 GHz Intel Pentium 4 CPU. The broker was running on another machine in the same subnet.

Error! Reference source not found. shows that a machine can support around 20 mixing sessions. But we should note that, in this test all streams are ULAW. This is not a computing intensive codec. When we had the same test with another more computing intensive codec, G.723, one machine supported only 5 mixing sessions.

7.2 Video Mixing

There are a number of ways to mix multiple video streams into one video stream. One option is to implement a picture-in-picture mechanism. One stream is dedicated as the main stream and it is placed in the background of the full picture. Other streams are imposed over this stream in relatively small sizes. Another option is to place the main stream in a relatively larger area than other streams. For example, if the picture area is divided into 9 equal regions, main one can take 4 consecutive regions and remaining regions can be filled with other streams. In our case, we choose a simpler mechanism. We divide the picture area into four equal regions and place a video stream into each region. This lets a low end client to display four different video streams by receiving only one stream. *VideoMixerServer* can start any number of *VideoMixers*. Each video mixer can mix up to 4 video streams. Therefore, in large meetings more than one video mixing can be performed.

Table 2. Video mixer performance test

Number of Video mixers	CPU usage %	Memory usage (MB)
1	20	42
2	42	54
3	68	68
4	94	80

Video mixing is a computing intensive process. One video mixer decodes four received video streams and encodes one video stream as the output. **Error! Reference source not found.** shows that a Linux machine with 1 GB memory and 1.8GHz Dual Intel Xeon CPU, can serve 3 video mixers comfortably and 4 at maximum. Therefore, video mixing is a very computing intensive process. In this test, we used the same incoming video stream for all mixers. The incoming video stream was an H.261 stream with an average bandwidth of 150kbps. The mixed video stream was an H.263 stream with 18fps.

7.3 Image Grabbing

The purpose of image grabbing is to provide users with a meaningful video stream list in a session. Without the snapshots of the video streams, users are often confused to choose the right video stream for them. Snapshots provide a user friendly environment by helping them to make informed decisions about the video streams they want to receive. Therefore, it saves a lot of frustration and time by eliminating the need for trying multiple video streams before finding the right one.

An image grabber is started for each video stream in a meeting. This image grabber subscribes to a video stream and gets the snapshots of this stream regularly. It first decodes the stream, then reduces its size to save CPU time when encoding and transferring the image. Then it encodes the picture in JPEG format. Either the newly constructed image can be saved in a file and served by a web server, or published on the broker network and accessed by subscribing to relevant topics.

Table 3. Image grabber performance test

Number of image grabbers	CPU usage %	Memory usage (MB)
10	15	66
20	35	110
30	50	148
40	60	192
50	70	232

Image grabbing is also a computing intensive task. Each image grabbing includes decoding, resizing and encoding of a video stream. However, resizing and encoding do not have to be done continually. They can be performed only when it is time to get the snapshot. Table 3 shows the performance tests for image grabbers. All image grabbers subscribed to the same video stream on a broker. That video stream was in H.261 format with an average bandwidth of 150kbps. Image grabbers saved a snapshot every 60sec to the disk in JPEG format. The host machine was a Linux machine with 1 GB memory and 1.8GHz Dual Intel Xeon CPU. These results show that 50 image grabbers can be supported on one machine. However, the number of supported image grabbers can change depending on the bandwidth of the video streams and the computing power of the underlying machine.

7.4 RTP Stream Monitoring

Stream monitoring service monitors the status of audio and video streams in a meeting, and publishes the events happening on dedicated topics. The entities interested in these events subscribe to these topics and receive them as the monitoring service publishes them. For example, all participants in a meeting subscribe to audio and video stream events to receive them. This allows them to know the identities of the current participants in the meeting and their status. Currently, there are four types of events: *StreamReceivedEvent*, *ByeEvent*, *ActiveToPassiveEvent* and *PassiveToActiveEvent*.

Contrary to other media processing services, stream monitoring is not implemented as a stand alone application. Instead, audio stream monitoring is implemented along with audio mixing service and video stream monitoring is implemented along with image grabbing service. Since all audio streams in a meeting are received by the audio mixer, and all video streams are received by image grabbers, we embedded the stream monitoring services into them to avoid extra audio and video stream delivery.

7.5 Media Processing Service Distribution

Media processing unit can be configured according to the needs of both small and large size organizations. For small organizations that will have only one or two concurrent meetings, one machine can be sufficient to run all media processing units. However, larger organizations need to run media processing servers on multiple machines. When distributing the servers, each machine can be dedicated to run one type of media processing service such as audio mixing. It is particularly important to run audio mixer servers on separate machines, since audio mixing is very sensitive and they should have prompt access to computing resources to provide best quality.

We use the previously explained service distribution model to distribute the media processing tasks. *MediaServerManager* implements the logic to talk to server containers and select the best available service providers. Currently, we use simple distribution logic for small number of settings. However, we plan to develop more complete scalable algorithms.

8 Meeting Management

Meeting management unit handles starting/stopping/modifying videoconferencing sessions. It also manages the media processing unit resources by using *MediaServerManagers*. In addition, it manages participant joins and leaves.

A videoconferencing session has two independent parts: an audio and a video session. *AudioSession* object manages the audio sessions and *VideoSession* object manages the video sessions. This management includes two main functions. First one is to manage the topics used for a meeting. They keep the list of users and the topics they publish their media. The second one is to provide session management services to participants, such as user joins and leaves. While handling these requests, they usually talk to other system components, such as media processing units and RTP link managers. *MediaServerManagers* are used by *MeetingManagers* to locate and to start/stop media processing servers. On the other hand, *MeetingSchedulers* are used to initiate and to end *AudioSession* and *VideoSession* instances. *MeetingSchedulers* can run either as independent applications or as embedded components in web servers. When they are used with web servers, an administrator or a privileged user initiates meetings through a web browser.

Although, session management components are lightweight entities and they can handle a large number of concurrent users, we still distribute *AudioSession* and *VideoSession* objects to provide fault tolerance. We use the service distribution model outlined in the previous section. *MeetingManagers* act as service providers and *MeetingSchedulers* act as consumers.

Here we explain the message exchanges that take place when creating a videoconferencing session. A *MeetingScheduler* sends an *Inquiry* message to *MeetingManagers* in the system. After receiving the responses, it selects a *MeetingManager* to ask for the service. It sends two request messages to the selected manager: *CreatAudioSession* and *CreateVideoSession*. This *MeetingManager* uses a *MediaServerManager* to locate an *AudioMixerServer* and an *ImageGrabberServer*. Then, it starts an *AudioSession* instance while providing the selected *AudioMixerServer*. This *AudioSession* object asks the given *AudioMixerServer* to start an *AudioMixerSession* to be used during this meeting. *MeetingManager* also initiate a *VideoSession* instance while providing the identified *ImageGrabberServer*. This *VideoSession* also asks the given *ImageGrabberServer* to start an *ImageGrabberSession* to be used during this meeting. This completes the initialization of the session. Users can join the session by sending *Join* messages directly to *AudioSession* and *VideoSession* components. A *VideoMixer* can also be added by exchanging messages with the *VideoSession* object. Usually administrators have the right to add and remove video mixers. We should also note that *MeetingManager* accesses *MediaServerManager* directly by calling its methods.

Here we also would like to explain briefly the messaging that takes place when users join meetings. When a speaker joins an *AudioSession*, a topic number is assigned for this user to publish its audio stream. Another topic number is also assigned to publish the mixed audio stream for this user by the audio mixer component. This user is also added to the *AudioMixerSession*. The mixer constructs a new stream for this user and publishes it in the given topic number. The interaction between the *AudioSession* and *AudioMixerSession* components are transparent to the user. If the joining user is a listener, in that case it is only given the mixed stream topic number to receive the audio of all speakers in the session. Since it will not publish any audio, it is neither assigned a topic number, nor added to the mixer.

When a speaker joins a *VideoSession*, it is assigned a topic number to publish its video stream. Then, an image grabber is also started to construct the snapshots of its video stream. This user is also given the list of available video streams in the meeting. He/she can subscribe to these streams by sending subscribe/unsubscribe messages to the *VideoSession* object.

9 Related Work

Currently, there are videoconferencing systems based on two main standards: IP-Multicast [1] and H.323 [2]. SIP [17] is another standard which is used to establish real-time sessions. It can also be used to implement videoconferencing systems, but it does not propose any architecture for building video conferencing systems.

IP-Multicast is a set of transport level protocols which provide group communications over the Internet. It provides services such as group formations and management, package delivery mechanisms, inter-domain interactions, etc. All these protocols are implemented on routers. Multicast has two main advantages. First one is its minimal usage of bandwidth. A sender sends one copy of a stream and it is duplicated along the way from sources to destinations when necessary. It avoids sending multiple copies of the same stream on the same link. Another advantage of multicast is its ease-of-use. A group of users need to know only the group address to start a meeting. This simplifies the management of meetings significantly. On the other hand, multicast tries to provide a group communication infrastructure for all Internet users. That results in the scalability and manageability problems [1]. In addition, it lacks widespread support from Internet routers and its traffic is blocked by almost all firewalls. Broadband service providers to homes and small offices usually do not provide Multicast support. Therefore, it is not suitable for systems that serve all internet users.

H.323 [2] is a videoconferencing recommendation from International Telecommunications Union (ITU) for package based multimedia communications systems. It defines a complete videoconferencing system including audio and video transmission, data collaboration and session management. It is heavily influenced by telephony industry and provides a binary protocol. Many h.323 based systems are hardware based such as Polycom, the most dominant player in the market. The scalability of h.323 based systems is very limited, since media processing and media distribution are not separated. They recommend MCU cascading for large scale conferences, but it is a very limited approach to support high number of users. An MCU connects to another MCU as a client. Therefore, multiple concurrent meetings can not utilize the same MCUs. Moreover, it is very difficult for h.323 based systems to go through firewalls. Each client uses many ports and they can not be changed.

VRVS [18] is another videoconferencing system that uses software routers to deliver audio and video streams. They have routers across United States and Europe. However, they are not an open source project and we do not know the details of their system.

10 Conclusion

In this paper, we proposed a service oriented architecture to implement scalable videoconferencing systems. This system utilizes a publish/subscribe messaging middleware to transfer both multimedia and data traffic. It implements a service oriented framework to manage and distribute system components efficiently. It allows new computing resources to be added dynamically and provides guidelines to add new services easily. Our performance tests show that this approach can deliver significant performance. However, we still need to develop algorithms that would allow global distribution of various media processing components.

11 References

[1] K. Almeroth, “The Evolution of Multicast: From the MBone to Inter-Domain Multicast to Internet2 Deployment”, IEEE Network, Jan 2000, Volume 14.

[2] ITU-T Recommendation H.323, “Packet based multimedia communication systems”, Geneva, Switzerland, Feb. 1998.

[3] A. Uyar, S. Pallickara, G. Fox, “Towards an Architecture for Audio/Video Conferencing in Distributed Brokering Systems”, The proceedings of The IC on Communications in Computing, June 2003, Las Vegas, Nevada, USA.

[4] Global Multimedia Collaboration System. globalmmcs.org

[5] http://www.naradabrokering.org.

[6] S. Pallickara and G. Fox. NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003.

[7] G. Fox and S. Pallickara. An Event Service to Support Grid Computational Environments. Journal of Concurrency and Computation: Practice & Experience. Volume 14(13-15) pp 1097-1129.

[8] ITU-T Recommendation G.114, One Way Transmission Time. (05/2003).

[9] The Access Grid Project. http://www.accessgrid.org/

[10] S. Pallickara, G. Fox, J. Yin, G. Gunduz, H. Liu, A. Uyar, M. Varank. A Transport Framework for Distributed Brokering Systems. Proceedings of PDPTA. June 2003, Las Vegas, Nevada, USA.

[11] G. Gunduz, S. Pallickara and G. Fox. A Framework for Aggregating Network Performance in Distributed Brokering Systems. Proceedings of the 9th International Conference on Computer, Communication and Control Technologies. Volume IV pp 57-63.

[12] Geoffrey Fox et al. “Grid Services For Earthquake Science”. Concurrency & Computation: Practice and Experience. Special Issue on Grid Computing Envronments. Volume 14:371-393.

[13] A. Uyar, G. Fox. Investigating the Performance of Audio/Video Service Architecture II: Single Broker. Submitted to The International Symposium on Collaborative Technologies and Systems. May 2005, Missouri, USA.
[14] A. Uyar, G. Fox. Investigating the Performance of Audio/Video Service Architecture II: Broker Network. Submitted to The International Symposium on Collaborative Technologies and Systems. May 2005, Missouri, USA.
[15] G. Fox and S. Pallickara. “JMS Compliance in the Narada Event Brokering System”. Proceedings of the International Conference on Internet Computing. June 2002. pp 391-402.
[16] Mark Happner, Rich Burridge and Rahul Sharma. Sun Microsystems. Java Message Service Specification. 2000. http://java.sun.com/products/jms
[17] J. Rosenberg et al., “SIP: Session Initiation Protocol”, RFC 3261, Internet Engineering Task Force, June 2002, http://www.ietf.org/rfc/rfc3261.txt
[18] Virtual Rooms VideoConferencing System. http://www.vrvs.org/