Service-Oriented Architecture for 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 computing power provides new opportunities for videoconferencing systems over Internet. Multimedia capable devices with broadband Internet connections are spreading rapidly. Even cell phones will have broadband internet access in the near future. This requires universally accessible and scalable videoconferencing systems that can deliver thousands of concurrent audio and video streams. 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 lack flexible service oriented architecture to support increasingly diverse. We believe that with the advancements in computing power and network bandwidth, more flexible and service oriented systems should be developed. In this paper, we outline a service oriented architecture for videoconferencing, GlobalMMCS, based on a publish/subscribe event brokering network, NaradaBrokering.

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

1 Introduction

The availability of increasing network bandwidth and computing power provides new opportunities for videoconferencing and collaborations systems over Internet. On one hand, broadband internet connections are spreading rapidly. Even cell phones will have broadband internet access in the near future. On the other hand, there are excellent quality audio and video add-ons (video cameras and microphones) for desktops and PDAs. Therefore, we can imagine that the trend in the increasing usage of videoconferencing systems will continue. This requires 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 diverse 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 lack 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.

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 in [3] and presented extensive performance test results at [4, 5]. In this paper, we outline a service oriented architecture to build a scalable and universally accessible videoconferencing system, GlobalMMCS [6], based on a publish/subscribe event brokering network, NaradaBrokering [7].

2 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.

There are three main components of this architecture (Figure 1): media and content distribution network, media processing unit and meeting management unit. The media processing unit is separated from media distribution completely to provide flexibility and scalability.

NaradaBrokering [7] event broker network is used to deliver both media and data packages. It provides a unified middleware for all communications. This reduces overall system complexity significantly.

Media Processing Unit provides services at server side to support diverse set of clients. Some clients can not receive multiple audio and video streams or they can not process and display them. Therefore, server side components generate combined streams for them. Currently, we implemented audio mixing, video mixing and image grabbing services.

Meeting management unit handles starting/stopping/modifying videoconferencing sessions. It manages the media processing unit resources and handles participant joins and leaves. *AudioSession* component manages the audio part of a session and *VideoSession* component manages the video part. *MeetingSchedulers* are used to start and end *AudioSession* and *VideoSession* instances.

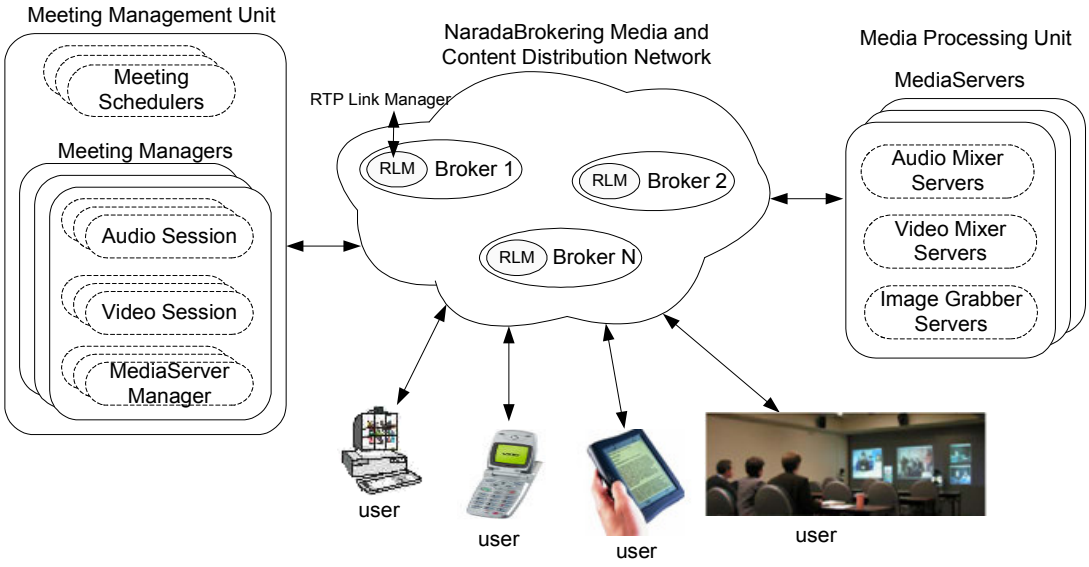

Figure 1. GlobalMMCS Architecture

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 services to participants for joining and leaving meetings. Therefore, 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.

3 Messaging Among System Components

We use NaradaBrokering-JMS publish/subscribe system to distribute the control messages exchanged among the components in the system. This simplifies building a scalable solution, since messages can be delivered to multiple destinations without explicit knowledge of the publisher. However, JMS [8] provides a group communication medium. It uses topics as the group addresses. In our system, while some messages are sent to a group of destinations, some others are destined to only one target. Therefore, an efficient group formation and message exchange mechanism should be designed. Messages should only be delivered to intended destinations. First, we examine various messaging types that take place in this system.

3.1 Messaging Semantics

A. Request/Response messaging: This messaging semantic is used when a consumer requests a service from a service provider. 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. Both the request and response messages are destined to one entity. Therefore, all service providers and consumers should have unique topics to receive messages destined to them only.

B. 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. The first one is to discover service providers and the second one is to execute a service on a group of service providers.

C. 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. 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 interested participants subscribe to the event topic and monitoring service publishes the events as they happen.

3.2 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 to change this root name and all topic names change accordingly. This allows installing more than one copy of the system on the same broker network. 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. Group topic names are constructed by adding the component name to the root by separating with a forward slash:

- GlobalMMCS/MeetingManager
- GlobalMMCS/AudioSession
- GlobalMMCS/MediaServer

Unique component topic names are constructed by adding a unique id to these component group addresses. When an instance of a component is initiated, it gets an id from the broker that is connected:

- GlobalMMCS/AudioSession/<sessionID>
- GlobalMMCS/MediaServer/<serverID>

We implemented a unique id generation mechanism at the broker network to provide unique ids on time and space [9]. An id generator runs in every broker and it can generate an id for every millisecond for 8 bytes long.

Sometimes a component needs to communicate 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* communicates with three different components: *RtpLinkManager*, *AudioMixerServer* and *RtpEventMonitor*. Using different topics simplifies logging and detecting the problems. It also simplifies developing codes to handle message exchanges with multiple components.

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

4 Service Distribution Framework

We provide a unified framework (Figure 2) to distribute many types of service providers. This framework supports running multiple copies of the same service providers in a distributed fashion. We assume that distributed copies should be able to run both in a local network and in geographically distant locations.

A. Addressing: Each service provider and consumer is identified by a unique topic name. This unique topic name is used to communicate with each entity privately. This topic name is generated as explained in the previous section. In addition, each service provider listens on the service provider group topic to receive messages destined to all group members.

B. 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 the fact that the status of the service providers change dynamically in our system. Therefore, it is not practical to update a centralized registry frequently.

First, a consumer sends an *Inquiry* message to the service provider group address. In this message, it includes its own private topic name, so that service providers can send the response messages back to it only. When service providers receive this message, they respond by sending a *ServiceDescription* message, in which they include their current status information and their private topic name. The status information depends on the nature of the service being provided. However, it must be helpful for the consumer to select the best 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 consumers do not know the current number of service providers in the system, after waiting for a while they assume that they received responses from all service providers.

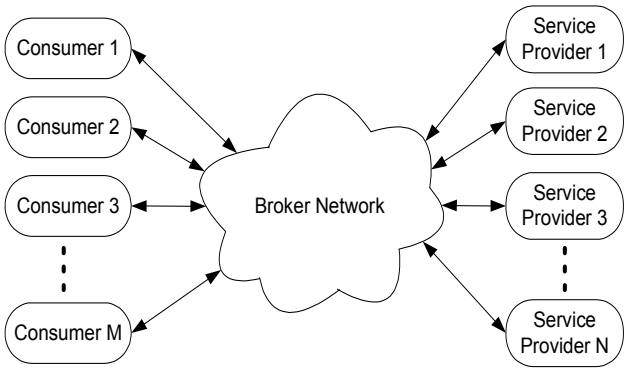

Figure 2. Service distribution model

C. 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.

D. Service Execution: When the consumer selects a service provider to run its service, it sends a *Request* message to the private topic of that 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.

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. 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.

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.

4.1 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. More service providers can be added easily to support more users.

Location independence: Service providers and consumers are totally independent of others in the system. A component is only connected to one broker and it exchanges all its data and media messages through this broker. Therefore, it can run anywhere as long as it is connected to a broker.

5 Conclusion

We presented service oriented architecture to build a scalable videoconferencing system. This system utilizes publish/subscribe messaging middleware to transfer both media and data traffic. It implements a service oriented framework to manage message exchanges among distributed components efficiently. It allows new computing resources to be added and removed dynamically. It also provides location independence to all system components.

6 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] A. Uyar, G. Fox. Investigating the Performance of Audio/Video Service Architecture II: Single Broker. The International Symposium on Collaborative Technologies and Systems. May 2005, Missouri, USA.

[5] A. Uyar, G. Fox. Investigating the Performance of Audio/Video Service Architecture II: Broker Network. The International Symposium on Collaborative Technologies and Systems. May 2005, Missouri, USA.

[6] GlobalMMCS Project. http://www.globalmmcs.org.

[7] NaradaBrokering project. http://www.naradabrokering.org.

[8] Mark Happner, Rich Burridge and Rahul Sharma. Sun Microsystems. Java Message Service Specification. 2000.

[9] Ahmet Uyar. Scalable Service Oriented Architecture for Audio/Video Conferencing. Ph.D. Thesis. Syracuse University. May 2005.