Grid Service Architecture for Videoconferencing

Wenjun Wu[1], Ahmet Uyar[1,2], Hasan Bulut[1], Sangyoon Oh[1], Geoffrey Fox[1]

[1] *Community Grids Computing Laboratory, Indiana University, USA*
[2] *Department of Electrical Engineering and Computer Science, Syracuse University, USA*
{wewu, hbulut,, ohsangy ,gcf }@indiana.edu, auyar@syr.edu
Indiana University Research Park, 501 N Morton St. 222, Bloomington, IN, 47408, USA

Abstract

In this paper we present a scalable, integrated and service-oriented collaboration system, namely Global Multimedia Collaboration System, based on the XGSP collaboration framework and NaradaBrokering messaging middleware. This system can provide videoconferencing services to heterogeneous endpoints such as H.323, SIP, Access Grid, RealPlayer as well as cellular phone. This paper discusses the design principle, system architecture and implementation in detail. The extensive performance measurement has been made to evaluate the scalability of the system.

Keywords

Collaboration, Multimedia, Global-MMCS, XGSP, NaradaBrokering, Web-Services,

1. Introduction

Collaboration and videoconferencing systems have become a very important application in the Internet. There are various solutions to such multimedia communication applications, among which H.323 [1], SIP [2], and Access Grid [3] are well-known. It will bring substantial benefits to Internet users if we can build an integrated collaboration environment, which combines these systems into a single easy-to-use, intuitive environment. However, at present they have features that sometimes can be compared but often they make implicit architecture and implementation assumptions that hamper interoperability and functionality.

XGSP (XML based General Session Protocol) [4] is a common, interoperable framework based on Web services technology for creating and controlling multipoint collaborations. XGSP uses a unified, scalable, robust "overlay" network to support audiovisual and data group communication over heterogeneous networking environments. XGSP offers a distributed, flexible conference management mechanism for integration of various collaborations and communities. Furthermore, XGSP specifies a common audiovisual signaling protocol for interactions between different audiovisual collaboration endpoints. Just like the text messages in SIP, XML should be used to describe the XGSP protocol because it makes the protocol easier to be understood and to interact with other Web based components.

Based on this framework, we have developed a system called Global-MMCS (Global Multimedia Collaboration System) [5] to support scalable web-service based interoperable collaborations. Global-MMCS integrates various services including videoconferencing, instant messaging and streaming, and is interoperable with multiple videoconferencing technologies. In terms of multipoint audio video collaboration, Global-MMCS can be regarded as scalable, integrated and service-oriented MCU (Multipoint Control Unit) in H.323 systems. Unlike typical H.323 MCU solution which usually puts everything in a hardware box, Global-MMCS are a pure software service system. It separates the MCU package into media delivery service, media processing service and session management service. On top of these core services, richer multimedia collaborations can be built to meet the demands in different application scenarios. Also several Gateways are developed to enable numerous multimedia clients to interact with the core services.

The paper is organized in the following way. Section 2 introduces related work and our design principle. System architecture is discussed in Section 3. The experience of building audiovisual collaboration is described in Section 4. Section 5 presents the implementation of the system and performance evaluation. Section 6 gives the conclusion.

2. Related Work

The multimedia collaboration framework has been studied over years. The well-known solutions have H.323, SIP and IETF MMUSIC [6]. The IETF's Multi-Party Multimedia (MMUSIC) working group proposed its own solution SCCP (Simple Conference Control Protocol) [7]. However, its main target was lightweight conference management for multicast instead of tightly controlled models. Because multicast

can't be deployed widely in the Internet in near future, in the year 2000 MMUSIC WG gave up and removed conference control from the WG charter. The project Access Grid started from the MBONE tools: VIC and RAT, and is also trying to define its own conference control framework rather than SCCP.

2.1 H.323

H.323 is a communication standard produced by the ITU, initiated in late 1996, and aimed at the emerging area of multimedia communication over LANs. It is widely supported by many commercial vendors and used throughout the world in commercial and educational markets. H.323 is defined as an umbrella standard specifying the components to be used within an H.323-based environment. It provides conference management functionality for audio/video conferences using the call signaling functionality of H.225 [8], H.245 [9]. These protocols provide call set-up and call transfer of real-time connections to support small-scale multipoint conferences. The protocol H.243 [10] defines some commands between the MCU and H.323 terminals to implement audio mixing, video switch and cascading MCU. H.243 commands have been included in H.245. For the data part of a conference, the conference management of the T.120 recommendation [11] is used. This standard contains a series of communication and application protocols and services that provide support for real-time, multi-point data collaborative applications including desktop data conferencing, multi-user applications, and multi-player gaming.

An H.323 conference system for packet switched networks can include one or more of the following functional components: terminals, gatekeeper (GK), multipoint controller (MC), multipoint processor (MP) and multipoint control unit (MCU). The H.323 control messages and procedures define how these components communicate. Figure 1 shows the structure of a typical H.323 system.

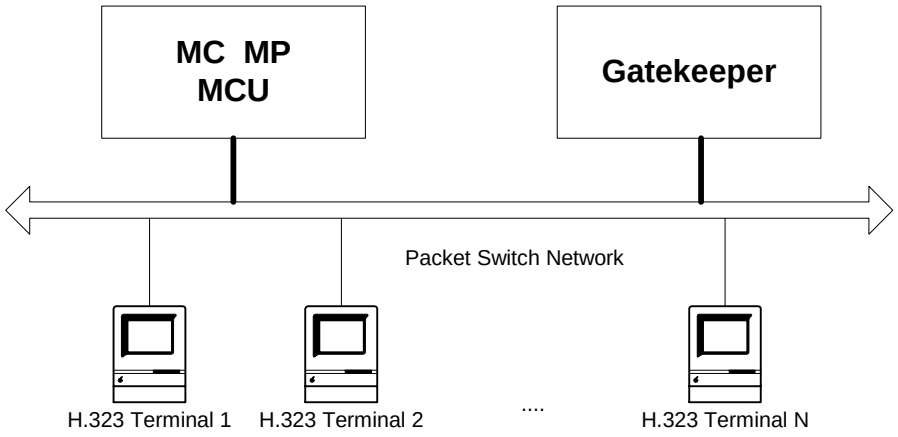

Figure 1. H.323 architecture

The H.323 MC and each H.323 participant in the conference establish an H.245 control connection to negotiate media communication types. The MP provides media switching and mixing functionality, e.g. the MP decides which of the media streams generated by the clients will be forwarded or mixed as a single stream. The H.323 Gatekeeper provides services such as address translation, RAS control, call redirection and zone management to H.323 clients. The gatekeeper may also provide other optional functions such as call authorization and call accounting information.

2.2 SIP

The Session Initiation Protocol (SIP) defines how to establish, maintain and terminate Internet sessions including multimedia conferences. Initially SIP was designed to solve problems for IP telephony. To this end, SIP provides basic functions including: user location resolution, capability negotiation, and call management. All the capabilities are basically equivalent to the service H.225 and H.245 in H.323 protocol. The major difference is that SIP was designed in a text format and took request-response protocol style like HTTP. But H.225 and H.245 were defined in a binary format and kept a style of OSI (Open System

Interconnection). Therefore SIP has some advantages of interaction with web protocols like HTTP in VoIP industry. More importantly, SIP doesn't define the conference control procedure like H.243 and T.120. Additional conference control mechanisms have to be implemented on the base of SIP to support the AV and data collaboration. Recently SIP research group begun to develop their framework and produced a few drafts [12,13]. But SIP work is still in the beginning phase and has not been widely accepted.

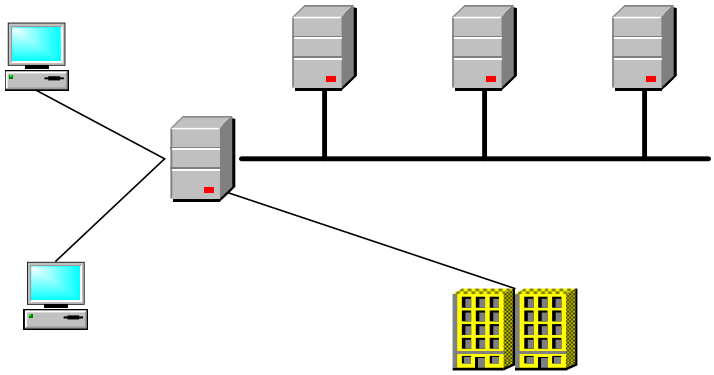

Figure 2 SIP System Architecture

The architecture of SIP based systems is showed in Figure 2. A SIP system usually includes SIP clients, a SIP Proxy Server, a Registrar Server, a Location Server, and a Redirect Server as well as a SIP Multipoint Control Unit (MCU). The SIP Proxy Server primarily plays the role of routing, enforcing policy of call admission. The proxy interprets and if necessary, rewrites specific parts of a request message before forwarding it. The SIP registrar accepts REGISTER requests and places the received information in those requests into the location service for the domain it handles. In addition, the SIP Proxy provides instant messaging service, forwarding SIP Presence Event messages and SIP text messages to SIP clients.

2.3 Access Grid and VRVS

Access Grid is a derivation from MMUSIC conference which uses MBONE tools and can support a large scale audio/videoconference based on a multicast network. Access Grid provides the group-to-group collaborations among 150 nodes connected to Internet 2 world wide. Access Grid improved MBONE audiovisual tools VIC and RAT. Permanent virtual meeting rooms were also introduced in Access Grid as "virtual venue" for the purpose of collaboration services management. Users are allowed to establish their own venue server which hosts the information about the user registration and venue addressing and offers rendezvous service to all the users. Users have to log into the venues server and start the multimedia clients in their nodes for communication through multicast and unicast bridges.

VRVS [14] is a project that extends the service of Access Grid. VRVS builds its collaboration service on top of pure software reflector infrastructure which is a kind of software multicast. It is capable of supporting MBONE tools, H.323 terminal as well as QuickTime player. It also supports some data sharing collaborations, like shared web browsing and shared desktop (VNC). But VRVS is not an open project having few documents for their architecture and conference control framework.

2.4 Kazaa and Skype

Kazaa [15] is one of the most popular and widely used p2p systems with over 85 million downloads worldwide and an average of 2 million users online at any given time. In the beginning, Kazaa only provided file sharing service. Kazaa nodes dynamically elect 'super-nodes' that form an unstructured overlay network and use query flooding to locate content. Regular nodes connect to one or more super-nodes to query the network content and use HTTP protocol to directly download the selected content from

the provider. In 2003, Kazaa extended its service to VoIP world by launching Skype [16] p2p VoIP solution and gained a big success. Skype was trying to address the problems of legacy VoIP solutions by improving sound quality, achieving firewall and NAT transversal and using p2p overlay rather than expensive, centralized infrastructure. It also provided supplemental features like instant messaging service.

Although all of these systems have advantages, they are not sufficient for building more advanced and integrated collaboration systems:

(1) SIP has very limited supported for conference control.
(2) In H.323 framework, AV collaboration and T.120 are not well integrated. Moreover, the AV communication services and T.120 overlay networks don't have very good scalability.
(3) H.323 and T.120 are designed in a relative complicated OSI model. It is not easy to understand and develop in their APIs.
(4) Access Grid heavily depends on multicast service and limited number of unicast bridge servers in the Internet 2. Since it doesn't have overlay substrate to support further deployment in heterogeneous Internet users, most users of Access Grid are institutions connected with Internet 2.
(5) All these frameworks only deal with homogenous video conferencing and can't connect to other collaboration systems.
(6) Kazaa and Skype use their own propriety protocols and can't interoperate with other legacy VoIP clients such as H.323 and SIP. They can only support audio conferencing and have no video service. In our design principle

2.5 Our design principles

Recently, many new technologies in the Internet such as XML, SOAP, Web-Service, Publish /Subscribe messaging as well as peer-to-peer computing have emerged and started to change the Internet applications. These new technologies enable the new architecture for collaboration systems:

(1) A unified, scalable, robust "overlay" network is needed to support AV and data group communication over heterogeneous networking environments. Such an overlay network should be able to go through firewall and NAT, provide group communication service in whatever unicast and multicast networks and offer reliable data delivery in whatever loss network. It also can to be configured as P2P or distributed server-based overlay to provide differential services for VIP and regular users.

(2) A common AV signaling protocol has to be designed to support interactions between different AV collaboration endpoints. For example, in order to get the H.323, SIP and MBONE endpoints to work in the same AV session, we have to translate their signaling procedures into our common procedure and build the collaboration session.

(3) A core conference control mechanism is required for establishing and managing the multi-point conference. The service of this part is quite like T.124 [17] (Generic Conference Control) in T.120 framework. However this mechanism will provide more flexible facilities to describe application sessions and entities. And it can be designed in a more scalable approach based on the powerful publish/subscribe messaging services. All description information for the applications and sessions can be kept in XML format rather than binary format, which will lead to easier development. Furthermore most control messages can be transferred through messaging middleware rather than central servers and the most session information can be distributed in all the participating nodes.

(4) Finally, we'd like to use web-services to integrate collaboration communities in different technologies. Various collaboration systems including Access Grid, H.323 and SIP should be regarded as Web-services components and provide Web-services interface of their conference control protocols to the core conference control components. They can invoke these services to build an integrated community-to-community conference across the communities.

The following table gives a comparison between Global-MMCS and other systems. Although the SIP and Access Grid are trying to add the conference control mechanism, their frameworks haven't been well defined. So we make this comparison according to the current capabilities of their systems.

	H.323	**SIP**	**IETF Access Grid**	**VRVS**	**Global-MMCS**
Conference Management	supported		supported	supported	supported

		No			
Overlay Network Environment	Internet / ISDN Firewall transversal under the support of VPN	No	Need multicast support , No firewall tunneling	Reflector Infrastructure Software Multicast	Publish/Subscribe Firewall & NAT transversal (VPN optional)
Data Collaboration	Limited: T.120 whiteboard, File FTP	No	Limited to (PowerPoint, Chat)	Limited to (Shared browsing and VNC)	allows full integration of all tools
Floor Control Mechanism	H.243 T.120	No Under development	No	No	Chairman based Flexible role setting
Scalability	Not good	Not good	Good	Good	Good
Support heterogeneous clients	No	No	No	H.323, MBONE	H.323, SIP, MBONE, RealPlayer, PDA, Cell Phone
Community-To-Community Collaboration	No	No	No	No	Yes

Table 1 Comparison of XGSP with Competitive Framework

3. Global-MMCS

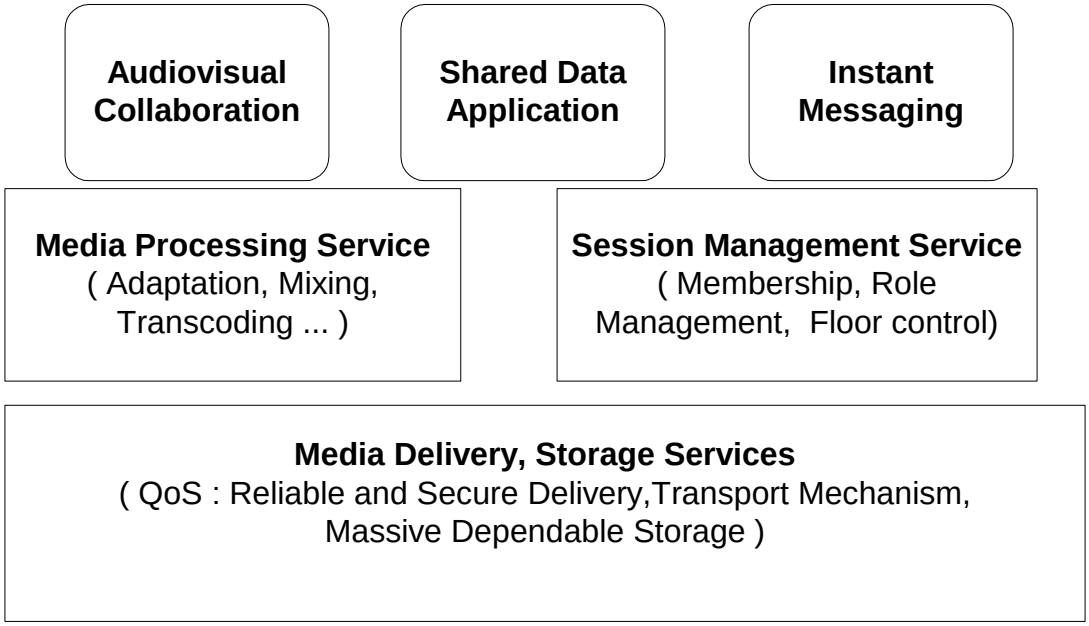

Figure 3 Global Multimedia Collaboration System Architecture

The figure 3 shows a general architecture of Global Multimedia Collaboration System. The top layer lists different kinds of collaborations, like videoconferencing, shared data applications. They have to be

built upon the services provided by the two layers below them. The bottom layer offers the media delivery and storage services to all the collaborations. Media delivery usually involves quality of service of data transmission, reliable and secure group communication and different transport mechanisms. In the middle layer, there are two application oriented services: media processing service and session management service. Media processing service defines the specific data processes necessary for collaborations such as media adaptation, media mixing. Session management service can maintain the membership in the collaboration session, and enforce floor control and role management. Note that if a collaboration session has different clients, there should be some gateway components in the middle layer to adapt heterogeneous clients.

Global-MMCS uses NaradaBrokering [18], a powerful "overlay" network for scalable, reliable and robust media delivery and storage. It is a general event brokering middleware, which supports publish-subscribe messaging model with a dynamic collection of brokers. NaradaBrokering is used be for multipoint data delivery and Web-Services communication substrate. Global-MMCS uses XGSP, a common, interoperable framework based on Web services technology for its session management service. XGSP offers a distributed, flexible conference management mechanism for integration of various collaboration communities. Using the XGSP API, it is easy for developer to create application specific session management.

In terms of multipoint audiovisual collaboration, Global-MMCS can be regarded as a scalable and service-oriented virtual MCU (Multipoint Control Unit) for bridging XGSP native clients, H.323 terminals, Access Grid clients, Real Players as well as cellular phones capable of image uploading. Global-MMCS also provides a general and scalable service platform (Media Servers) that can execute various media processing including video mixing, audio mixing and snapshot generation. Other real-time collaborations like whiteboard, text chat, shared document can also be integrated to this general and web-services based system. The following sections describe NaradaBrokering messaging service and XGSP framework in detail.

3.1 Narada Brokering Grid Messaging System

NaradaBrokering from the Community Grid Labs is adapted as a general event brokering middleware and substrate for deploying Web-Services. It is a distributed publish/subscribe messaging system that provides a scalable architecture and an efficient routing mechanism. It organizes brokers in a hierarchical cluster-based architecture. The smallest unit of the messaging infrastructure is the broker. Each broker is responsible for routing messages to their next stops and also 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 "pathlengths" between brokers that increase logarithmically with geometric increases in network size, as opposed to exponential increases in uncontrolled settings.

NaradaBrokering supports dynamic broker and link additions and removals. While adding new brokers and links, it implements a broker organization protocol to avoid an unstructured broker network which hampers the development of efficient routing strategies. This lets broker network to grow or shrink dynamically.

NaradaBrokering has a flexible transport mechanism. 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.

JMS API [19] is very good for developing scalable collaboration applications. The publish/subscribe interaction paradigm make it possible to build a peer-to-peer and loosely coupled distributed system. And publish/subscribe topics, which represent keywords for publisher and subscriber, can be used to describe hierarchy and complicated collaboration groups. Fault tolerance in JMF style communication is also introduced by using WSRM [20] semantic between replicated publishers or subscribers.

3.2 XML-based General Session Protocol

Figure 5 shows the important components in XGSP framework:

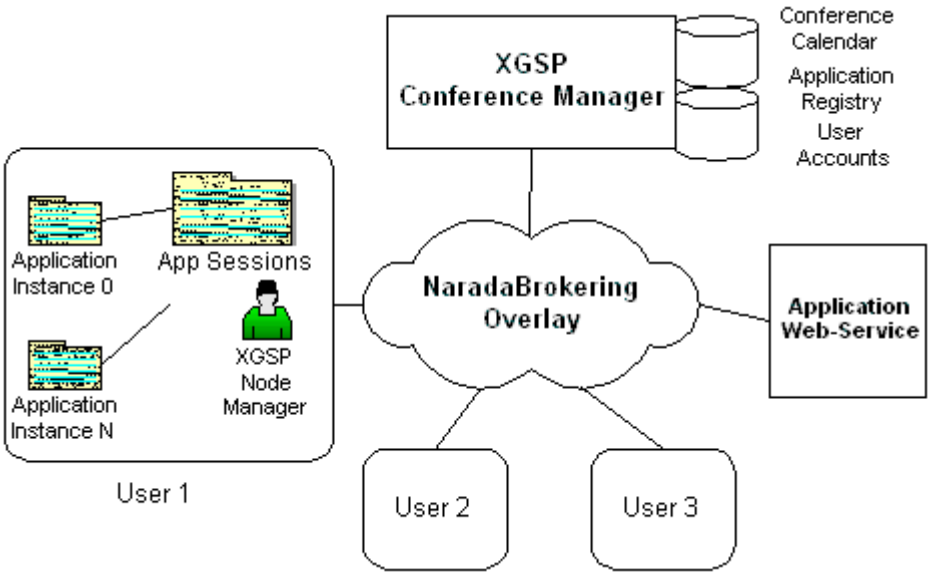

Figure 5. XGSP Conference Control Architecture

The conference manager is the server keeping the important information for all the conferences. Through the manager, users can create a new conference or terminate an old one. The meta-data repositories in the conference manager includes: a conference description set, application registry set as well as user accounts. The conference description set contains the registries of all the scheduled conferences and is organized as conference calendars. Each conference registry includes the fields: Conference ID, Conference Name, Conference Mode, Conference Time, and Available Application Session Templates. The application registry set has all the registries of the collaboration application such as audio/video, chat and whiteboard. An application registry usually contains the entries like application identification, role systems definition as well as specific attributes.

A node manager is the user interface for the XGSP conference management service in each user. An application instance refers to a client of the collaborative applications. Because a node manager has the factories for all kinds of applications, it can create new application instances, and invoke start, stop, and set-role methods in them.

The XGSP conference control includes three services: conference management, application session management and floor control. The conference management supports user sign-in, user create/terminate/join/leave/invite-into XGSP conferences. The application session management provides users with the service for creating/terminating application sessions. And the floor control manages the access to shared collaboration resources in different application sessions.

3.3 Build Collaboration Web-Services using XGSP and NaradaBrokering

Developers should follow the following approach to build their own collaborative applications based on NaradaBrokering and XGSP.

(1) Use NaradaBrokering for control communication and data distribution

NaradaBrokering provides JMS-style publish/subscribe API for group communication. Developers who are familiar with JMS can use the API for all the communication in the application. There are also more advanced data transmission services in NaradaBrokering like reliable delivery and ordered delivery. Some collaboration may need them for maintaining complicated consistency in distributed sharing.

To distinguish different sessions in the collaboration, developers have to specify the topic scheme. Each application session should have its own topic name space inside NaradaBrokering. We can define the naming schema: /xgsp/conferenceID/Application-Session-ID. The conferenceID field is generated by the conference manager and determined when the conference is activated. The Application-Session-ID field is generated when the application session is created. This field can have three kinds of forms: the default public session can use the application type identification like av, chat, whiteboard. The public application sessions take the format of < application type, sequence number >. The sequence number represents the last

number of the application sessions. The private application sessions can be < application type, initiator-ID, sequence number>. In the following, we give a few examples: suppose the conference named ourtestroom is created. And it has two default application sessions with the topic names: /xgsp/ourtestroom/av and /xgsp/ourtestroom/chat. If the chairman in the conference creates two whiteboard sessions, their topic names should be: /xgsp/ourtestroom/whiteboard-0, /xgsp/ourtestroom/whiteboard-1. For a private whiteboard session initiated by the user with the user ID: "testuser", its topic name should be /xgsp/ourtestroom/whiteboard-testuser-0.

(2) Define roles and its capability in XGSP

Each collaborative application defines its role system in a XML registry. A role description includes the role name and the role capability. The conference manager keeps the database of all these definition registries. It copies the database to a user node manager when the user joins the conference.

The conference chairman has the right of setting the roles. It can send a "SetApplicationRole" message to the application instance running in other users. A "SetApplicationRole" message tells the conference participants which user should be changed to this role. All the application instances have to parse the message and take appropriate actions. For example, in the chess application, the conference manager has its application registry defining three different roles: black, white and observer.

```
< ApplicationRegistry>
< ApplicationID>  chess </ApplicationID>
<roles>
  <role>
     <roleName> black </roleName>
     <capabilities> player-first </capabilities>
     <roleName> white </roleName>
     <capabilities> player-second </capabilities>
     <roleName> observer </roleName>
     <capabilities> non-player </capabilities>
  </role>
</roles>
….
</ApplicationRegistry>
```

(3) Define application specific processing services and hooked it into NaradaBrokering

Collaborations usually need media processing services. For example, conferencing needs audio mixing and video mixing. These services can be regarded as collaboration web-services, and described in WSDL. Collaboration clients can communicate with the services through SOAP and JMS messages.

NaradaBrokering Grid messaging system can work as a substrate of SOAP transport and SOAP routing so that Web-Services can be deployed on top of it. By using handler chain approach and service advertisement mechanism, it enables clients to discover, invoke services and interact with them. SOAP handlers of Web-services can automatically generate service advertisements on service startup. And then Web-services connect directly to NaradaBrokering and expose their capability through advertisements. Clients use XPath or Regular expressions-like query to search for this advertisement.

XGSP framework provides session control mechanism for creating and managing the instances of the Web-Services. It can work as service factory where session related media services can be created and attached to the active application session.

(4) Implement the interface between the XGSP NodeManager and application clients

Collaboration client instances have to be created by the XGSP Node Manager. XGSP specifies the interface between the Node Manager and application clients. The application client must implement the following interface at first to be hooked with the NodeManager and report the heart-beat information to it.

```
public interface Application{
void setServiceMgr(NodeManager provider);
boolean isAlive();
}
```

Through the node manager, the application client can obtain a session handler which allows the client to join/leave the session and listen to the events in the session. XGSP interface defines two listeners: Session EventListener and Role EventListener.

```
public interface SessionListener extends EventListener{
  public void sessionJoined(SessionEvent evt);
  public void sessionLeft(SessionEvent evt);
  public void sessionInvited(SessionEvent evt);
  public void sessionExpelled(SessionEvent evt);
  public void sessionDestoryed(SessionEvent evt);
}
public interface RoleEventListener extends EventListener{
  public void RoleChanged(RoleEvent evt);
}
```

The above code segment shows the interface of the listeners. The application client must implement the listener interface to handle the events. Also Application Registry has to be defined to load the main class of the application. For the chess application, the conference manager has its application registry defining three different roles: black, white and observer. And the MainClass field tells the NodeManager which java class it should load.

```
< ApplicationRegistry>
< ApplicationID>  chess </ApplicationID>
<MainClass>games.SVGGame</MainClass>
<roles>
  <role>
     <roleName> black </roleName>
     <capabilities> player-first </capabilities>
     <roleName> white </roleName>
     <capabilities> player-second </capabilities>
     <roleName> observer </roleName>
     <capabilities> non-player </capabilities>
  </role>
</roles>
….
</ApplicationRegistry>
```

4. Audiovisual Collaboration in Global-MMCS

In this section, we discuss the experience of building an audiovisual collaboration system using XGSP framework and NaradaBrokering messaging system. This system can connect numerous audiovisual endpoints including H.323, SIP, Access Grid and cellular phones.

Although different AV endpoints have quite different implementation and capabilities, they share a common AV collaboration pattern. Assume in an AV session, there are video streams (VS) and audio streams (AS): VS $\{v_1, v_2, \ldots v_m\}$, AS $\{ a_1, a_2, \ldots a_n \}$. And we also have AV endpoints: $E_1, E_2, \ldots E_n$. Each endpoint may send a single or multiple video streams, but only send an audio stream. In this AV collaboration session, each endpoint adds new AV streams into VS and AS by sending audio and video RTP packets, or removes streams by sending RTCP "BYE" packets to leave the session. At the same time, each endpoint gets a subset from VS and AS by receiving RTP streams and rendering them.

Different types of AV endpoints have different collaboration capabilities. Multicast endpoints are able to receive multiple video and audio streams, display all the video streams in their screens, and mix all the audio streams by themselves. For example, Access Grid endpoints receive all the streams in VS and AS to create the duplications of VS and AS, and allow users to make selection of rending video and mixing audio. On the other hand, unicast endpoints like Polycom ViaVideo can only receive and play a single video and audio stream. So they can't attend the collaboration with Access Grid endpoints. This issue may be addressed by introducing some middleware, which offers the description data of the streams to the users of these endpoints, allows them to switch among the video streams, and mix audio streams for them.

NaradaBrokering publish/subscribe messaging middleware is a perfect candidate for this purpose. An AV RTP stream is regarded as a "topic" and each RTP packet from this stream as an "event" for this topic. Only the sender of this stream can "publish" AV events to this topic. Other endpoints need to subscribe to

this topic in order to receive the stream data. In this way, the Publish/subscribe middleware provides a group communication service for those unicast endpoints. Besides of the RTP events for the stream topic, some major events for each stream topic have to be defined to describe the change in this stream. They include five major events: *NewStreamEvent, ByeEvent, TimeOutEvent, Active-to-Passive, Passive-to-Active*.

Event Name	**The change in the status of the stream**
NewStreamEvent	This stream has been created
ByeEvent	The stream gives a BYE RTCP packet, indicating it has left the session
TimeOutEvent	This stream has not send any RTCP packet for a long time, indicating it may have left the session
Active-to-Passive	The stream has stopped sending RTP packet
Passive-to-Active	The stream resumed sending RTP packet

Table 2 major events of an AV stream

Based on these events, Unicast endpoints can maintain the duplications of VS and AS list just like multicast endpoints. Furthermore Active-to-Passive and Passive-to-Active events notify the endpoints whether the streams have new data at that moment, which sometimes are very important to AV collaboration. For example, since the endpoints only get a mixed audio for all the audio streams, it is very useful for the endpoints to know who is speaking in this mixed audio. These events help the users to get the "focus" of the AV collaboration.

Figure 6 shows the architecture of implementing audiovisual collaboration mentioned above. The following sections give the detailed description of the components in the figure. Section 4.1 discusses the media delivery of NaradaBrokering. Section 4.2 and 4.3 describe media processing. Section 4.4 illustrates the audiovisual session management in XGSP. The topic on how to adapt other multimedia clients to Global-MMCS is covered from Section 4.5 to Section 4.7.

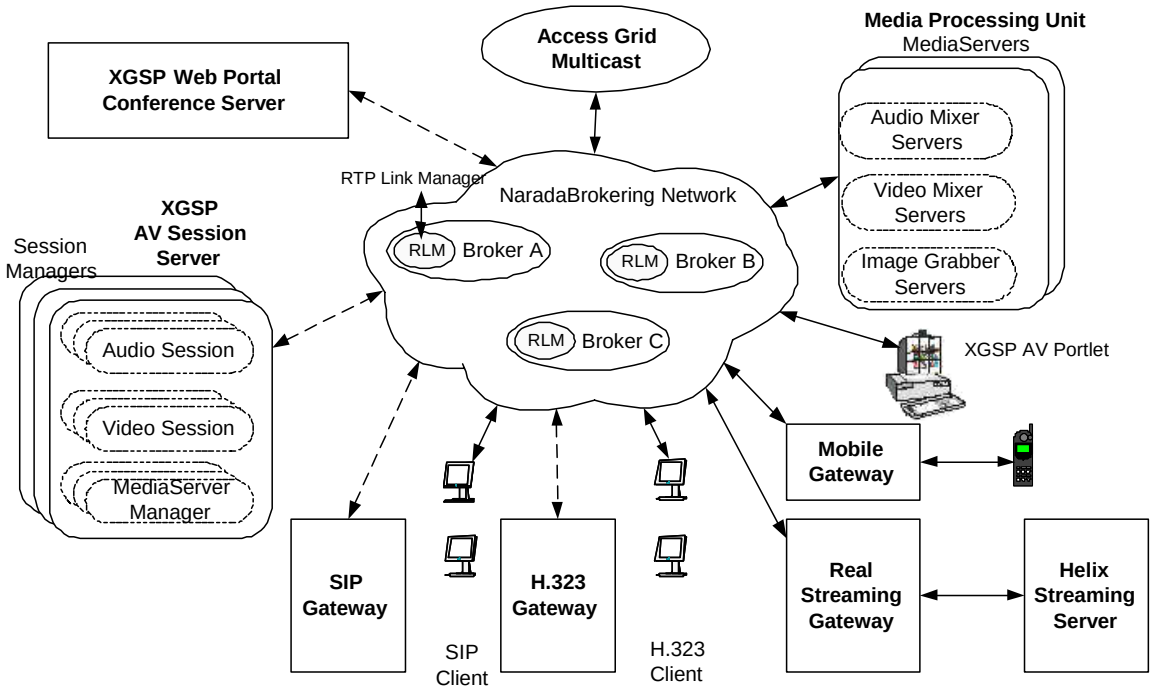

Figure 6. Audiovisual Collaboration in Global-MMCS

4.1 Media Delivery

An efficient audio/video distribution system should have a number of characteristics. First of all, the best possible routes must be chosen from sources to destinations when delivering the content. This is important both not to load extra traffic on the network and not to add extra transit delays to packages. Secondly, audio and video streams need to be replicated only when it is needed along the path from sources to destinations. This saves significant bandwidth. Usually there are multiple participants in a videoconferencing session, when the sender of a stream sends one copy and distribution network replicates it when necessary, that user is removed from the burden of sending many copies. Thirdly, since audio and video streams are composed of many small sized packages, it is important to add minimum headers to each of these packages. Otherwise, there can be substantial increase in the amount of data transferred. Lastly, unreliable transport means should be used whenever possible. Contrary to many data applications, audio and video transfer can tolerate some package loss to some extend. These protocols usually provide lower transit delays, since they do not have the extra cost associated with error correction and package retransmission.

Traditionally, multicast is used to deliver audio and video streams for videoconferencing sessions. But the lack of widespread use and the problems with firewalls/NATs discouraged many people from using it. Publish/subscribe systems provide a messaging middleware that decouples producers and consumers on time, space and synchronization. Since publish/subscribe messaging systems provide reliable group communication services, in addition to audio and video delivery, they can also be used to deliver the control messages exchanged among the distributed components in the system. On the other hand, since publish-subscribe systems are not designed to serve real-time multimedia traffic. They are usually used to deliver guaranteed messages by employing reliable transport protocols. In addition, they do not focus on delivering high bandwidth traffic or reducing the sizes of the messages they transfer. It is more important for them to provide more services than saving bandwidth. Each message tends to have many headers related to the content description, reliable delivery, priority, ordering, distribution traces, etc. Many of these services are not important for audio and video delivery. Therefore some additions need to be made to deliver multimedia traffic.

4.1.1 Implementing a distributed topic number generation mechanism

NaradaBrokering implemented a string based topic mechanism. Although this was very useful for other applications, it was not adequate for media delivery. Since media streams are composed of many small sized packages and they are bandwidth intensive, it is very important to add minimum headers to each message. When strings are used as topic names and a topic name is added to each media package, this may result in significant increase in the required bandwidth and it adds more load on the brokers and links. When strings are serialized, each character takes at least one byte, depending on the size of the string topic name; tens of bytes can be added to each message. Since media packages can be as low as 20 bytes, it would not be efficient to add tens of bytes to each message as the topic name.

One way of solving this problem is to impose a limit on the size of the topic string. We can require each topic to have at most 8 characters, but this would limit the number of possible topic names significantly. In addition, the collision of topic names would increase. On the other hand, increasing the maximum size of the topics would result in more bandwidth and load, though it would provide more options. Therefore, we have decided to implement a topic mechanism which can provide more options and take less space.

Although one aspect of the topics is their size, another aspect is the way they are created and their uniqueness is insured in a distributed setting. Here we outline three conditions to meet for a distributed topic management system:

1. A topic generator must be able to create topics without interacting with other topic generators in the system. We avoid centralized solutions for fault tolerance and speedy execution.
2. A topic generator should be able to fail and start over without requiring saving its state to a stable storage. Namely, topic generators must be stateless.
3. Each topic should have a predetermined size and their size should be as small as possible.

The first condition requires the spatial independence of a topic generator. This can be achieved by assigning a unique topic generator id to each topic generator in the system. Then this unique id can be added to every topic constructed by that generator to provide system wide uniqueness. In NaradaBrokering network, each broker is assigned a unique id. We can utilize this mechanism to provide unique ids for each

topic generator. We require that a topic generator runs in each broker. Clients ask this topic generator to construct a new topic for them. The broker id in NaradaBrokering system is 16 bytes. This is obviously too long to add to each topic. On the other hand, this broker id is not designed to use the minimum space, it is designed to provide the best performance. Therefore, it does not utilize the 16 bytes range efficiently. Instead, it is possible to generate a 20 bit id from this 16 bytes broker id with a simple conversion. This 20 bit is small enough to add to each topic. In this way each topic generator in brokers will be able to generate topics independent of other brokers in the system.

The second condition requires the temporal independence of a topic generator. This can be achieved by adding a timestamp value to each topic. Since NaradaBrokering brokers are synchronized [15] with high accuracy clocks using Network Time Protocol, it simplifies the problem considerably. This eliminates the conditions where the clock of a computer can be changed backward. With this synchronization mechanism in use, we can assume that the time always flows forward, though it may stall for short periods of time when synchronizing. Therefore, we can add a timestamp to each topic to provide temporal independence of topics without requiring state savings to file system when restarting a broker.

The third condition requires that topics should have minimum size. We can construct a topic number by combining the topic generator id with a timestamp. Since topic generator id is 20 bits, we need to determine the size of the timestamp. If we set the total topic number size as 32 bits (4 bytes), remaining 12 bits would provide 2^{12}=4096 different options for the timestamp. Therefore, it is too small. When we set the total topic number size as 64 bits (8 bytes), 44 bits are left for the timestamp. This would provide 2^{44}=17592186044416 distinct timestamp values. If we increment timestamp value by one in every millisecond, this would be enough for 557 years. Therefore 64 bits topic number (Figure) is a good choice. It is small enough to add to each audio and video package.

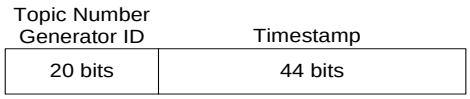

Figure 7: 8 bytes topic number for RTPEvent

In conclusion, this mechanism provides a fast and scalable solution to generate unique topic numbers with 8 bytes. Since a broker does not interact with other brokers, or keeps track of the unused topic numbers, it can generate topic numbers very quickly. In addition, this mechanism lets brokers fail and start over without interacting with other brokers or using a stable storage device. Moreover, it guarantees to generate unique topic numbers.

4.1.2 Designing a New Event

In publish/subscribe messaging systems, messages tend to have many headers, most of which are related to the quality of services provided. Since audio and video streams do not require quality of services such as persistence and reliability, many of these headers are unnecessary. For example, a message in JMS API has at least 10 headers. Many of them are redundant in the context of audio and video delivery. These headers take around 200 bytes when they are serialized to transfer over the network. If these 200 bytes are added to each audio and video package, it results in substantial increase in the bandwidth of the audio or video streams. For example, a ULAW audio package for 20 ms has a size of 172 bytes including the RTP header and entails 64 kbps network bandwidth. Padding an extra 200 bytes of header to each audio package results in the bandwidth requirement of up to 148 kbps. Then, there is the cost associated with serializing and de-serializing the multimedia content. Therefore, we need to design a new event type with minimum headers and minimum computational overhead.

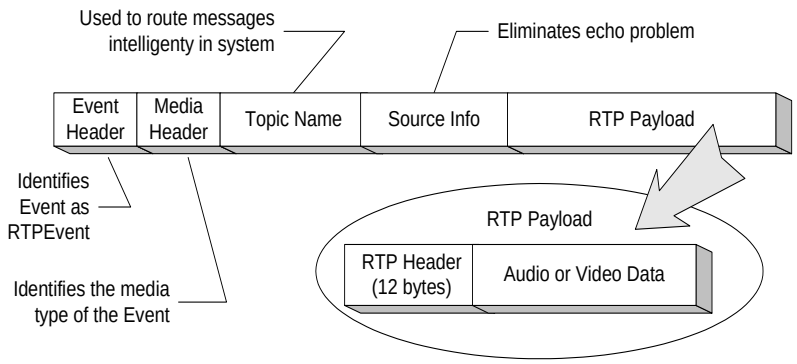

Figure 8. Serialized RTPEvent

RTPEvent encapsulates media content that comprises of 4 elements. There are two headers identifying the event type. Both headers are 1 byte. Event header identifies the event as RTPEvent among other event types in NaradaBrokering system. Media header identifies the type of the RTPEvent such as audio, video, RTCP, etc. To eliminate echo problems arising from the system routing content back to the originator of the content, information pertaining to the source is also included. This information can be represented in an integer, which amounts to 4 bytes. Finally, there is the media content itself as the payload in the event. Although, in **Error! Reference source not found.**8 an RTP package is seen as the payload, it can be any data type. Therefore, the total length of the headers in an RTPEvent is 14 bytes. 14 bytes is small enough to add to each audio and video package transferred in the system.

We should also note the fact that when an RTPEvent package is traveling through multiple brokers, a 16 bytes dissemination trace value is added in the first broker and it is removed again in the last broker before sending it out to the destination client. Although, this adds extra load on the links among brokers, it should not affect the quality of the communications seriously, since these links are supposed to be high bandwidth. On the other hand, the links between clients and brokers will not have this extra load. These links are usually the most vulnerable links in the communication path from sources to destinations. Therefore, the addition of the dissemination trace value should be tolerable.

4.1.3 Adding support for legacy RTP clients

Although, we have developed a XGSP AV native client which can join a videoconferencing session and send/receive audio/video streams using RTPEvent messages, a lot of other RTP based clients from H.323, SIP and Access Grid, also need to be supported. These clients exchange RTP packages and use UDP or multicast as the transport mechanism. Therefore, we have developed a specialized implementation of the NaradaBrokering transport framework called RTPLink for both UDP and Multicast. This process entailed an implementation of the Link interface which abstracts the communication link between two entities. The RTPLink can receive raw RTP packages over UDP or multicast from legacy systems, wrap them in RTPEvents and propagate through the protocol layer in the broker node. Once it reaches the protocol layer, the event is routed within the distributed broker network.

An RTP media stream is composed of two different kinds of packages: RTP and RTCP packages. RTP packages carry the audio or video data along with the RTP headers which are used to provide a host of services such as payload type identification, sequence numbering, timestamp, source identification, etc. RTCP packages carry the control messages to monitor the timely delivery of real-time data. RTP and RTCP packages are exchanged on different ports. RTCP packages are exchanged on the port number following the RTP port number. Therefore, the RTPLink implementation starts two sockets on these two ports. In addition, it publishes the RTP and RTCP messages on different topic numbers. Similar to RTP protocol, RTCP packages are published on the topic number following the RTP topic number. Therefore, a pair of topic numbers is used to publish an RTP stream.

The RTPLink deals with the initialization, registration and data processing on the communication link. During the initialization process, the RTPLink is provided a port number to listen for packages from the legacy client at the other end, and also the IP address and the port number of the legacy client to be able to send packages. For registration purposes, the RTPLink is assigned a NaradaBrokering-ID and the RTPLink subscribes to the topic corresponding to its meeting. In the data processing part, the RTPLink constructs the RTPEvents for processing within the broker network when it receives media packages. On the other hand, when an RTPEvent is ready to be sent to the legacy application, the RTPLink retrieves the RTP payload

from the RTPEvent and sends it to the legacy application based on the parameters specified during initializations.

Different RTP sessions lead to different implementation of the RTPLinks. On unicast sessions, usually a user is in direct communication with another user and only one audio/video stream is exchanged through one unicast RTPLink. On the other hand, in multicast sessions which have many audio or video streams from many participants, the Multicast RTPLinks should handle the communication with a group of users.

A multicast RTPLink has two options: either it can publish all these streams to the same topic pair or it can publish each RTP stream on a different topic pair. When all streams are published on the same topic pair, all participants who subscribe to this topic receive all the streams. Namely, they do not have any choice to select any specific stream in the group. On the other hand, if every stream is published on a different topic, then each user can select any stream of its choice. In practice, usually it makes sense to publish all audio streams on the same topic, since a user usually wants to hear all speakers in a session. However, it might be better for video streams to be published on different topics, since some users might only want to receive some video streams in a session. A multicast RTPLink can examine the headers of the received RTP and RTCP packages and determine the packages that belong to the same RTP stream. Each RTP stream in a real-time session has a unique SSRC number. Then, it can publish all packages belonging to the same stream to the same topic number. Since many streams require many topics, the multicast RTPLink should either be given a list of topic numbers to use when it is started, or it should ask for a new pair of topics whenever it receives a new RTP stream. It can also examine the content of RTCP packages to garbage collect the RTP streams when the session is ended by sending a bye message. In addition, it can generate events to mark the arrival and departure of RTP streams.

4.2 Media Processing Service

Global-MMCS supports multiple copies of the same service providers in a distributed fashion. Since there are a number of different service providers in our system, it would be better to have a unified framework for distributing the service providers. We assume that distributed copies should be able to run both in a local network and in geographically distant locations with different network connections. Each media 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: a public topic for all the service providers and a private topic for itself.

4.2.1. 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 of this approach. Another difficulty is that since in our system the status of the service providers changes dynamically, it is not reasonable to update a centralized registry frequently.

In our 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 decide from which 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.

4.2.2 Service Selection

When a consumer receives the *ServiceDescription* messages from service providers, it compares the service providers according to the service selection criteria set by users. 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. This policy can be set by a configuration file to provide more flexibility.

4.2.3. 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. 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 from the other, 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. This simplifies the distribution and management of service providers significantly.

4.2.4. Advantages of this service distribution model:

- **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 we have services to provide them. The fact that initially a consumer sends a message to all service providers, and they all respond back to the consumer, limits the number of the supported service providers. However, this can be eliminated by limiting the number of service providers who respond to specific inquiry messages. 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 these 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.

4.3. Media Processing Units

We provide media processing services at server side to support a diverse set of clients. Some clients have high network bandwidth and advanced processing and display capacity. They can receive process and display multiple concurrent audio and video streams. Therefore, they can receive all audio and video streams in a meeting. Some other 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 a RTP stream monitoring service. All these services require real-time processing and usually high computing resources.

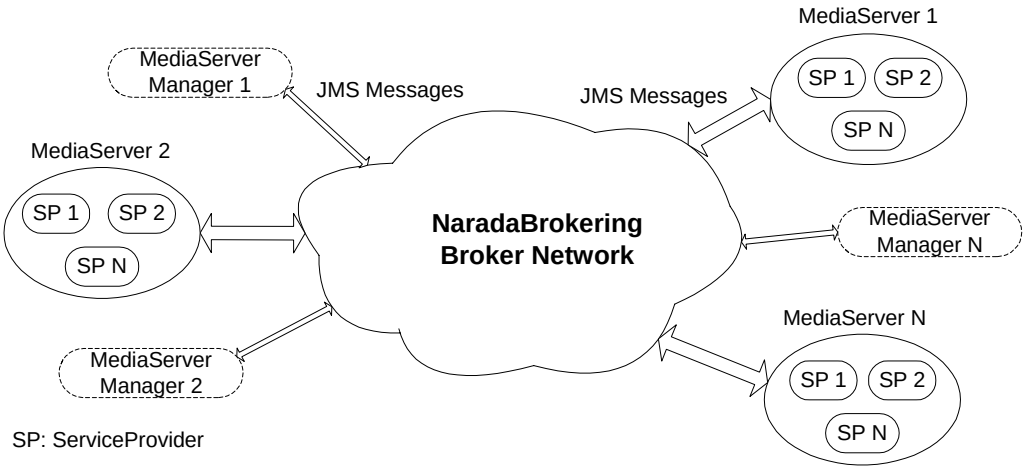

Figure 9 Media Processing Framework

Media processing framework (Figure 9) 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 the 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 media processing service providers: 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 MediaServerManagers. MediaServerManagers implement the semantics to talk to MediaServers.

4.3.1 Audio Mixing

Audio mixing is very important to those clients which can't receive multiple RTP audio streams and mix them. An AudioMixerServer creates an audio mixer for the clients in a XGSP audio session. Any number of audio mixers can be created in the AudioMixerServer as long as the host machine can handle. Each speaker is added to the mixer as they join the session, and special mixed streams are constructed for them. Audio mixer receives the streams from the broker network and publishes back the mixed streams 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 will be dropped and result in the breaks in audio communications. Therefore, the load on audio mixing machines should be kept as low as possible.

4.3.2 Video Mixing

Video mixing service improves the visual collaboration especially for those limited clients which can only handle a single video stream. 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 way of video mixing. 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.

4.3.3 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, and 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.

4.3.4 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. As mentioned above, there are five types of events in the status of a stream: NewStreamEvent, ByeEvent, TimeOutEvent, ActiveToPassiveEvent and PassiveToActiveEvent. Each of these events gives information regarding a particular steam. These events also provide information about the identity of senders of these streams.

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.

4.3.5 Media Processing Service Distribution

We use the previously explained service distribution model to distribute the media processing tasks. A Media Server Manager usually running inside the XGSP AV session server implements the logic to talk to server containers and also for selecting the best available service providers. Currently, we use simple distribution logic for small settings. But we are working on more complete scalable algorithms. We plan to include prediction of necessary resources for a meeting and schedule accordingly. Users can be asked to provide more information about the meeting when they are scheduling. For example, they can provide the expected number of participants. This can be very helpful when scheduling audio mixers and image grabbers.

4.4. Session Management

XGSP framework divides the session advertisement information actually into two levels: one is the collaborative conference calendar, the other is the detailed information for audiovisual clients to join the conference, for example the session identification in the system and transport addresses associated with the session.

XGSP audiovisual sessions have five states: created, canceled, activated, deactivated and finished. The states are managed by the XGSP conference server and the XGSP AV session server. Public XGSP audiovisual sessions are initially created when a XGSP meeting is created. Based on the meeting schedules specified by users, the XGSP conference manager will activate/deactivate XGSP meetings and ask the AV session server to activate/deactivate the associated audiovisual sessions. Users are only allowed to join/leave an activated XGSP AV session. The XGSP conference manager also provides administration web pages through which an administrator or conference chairs could ask the AV session server to manage the media processing services like adding some new streams into video mixers.

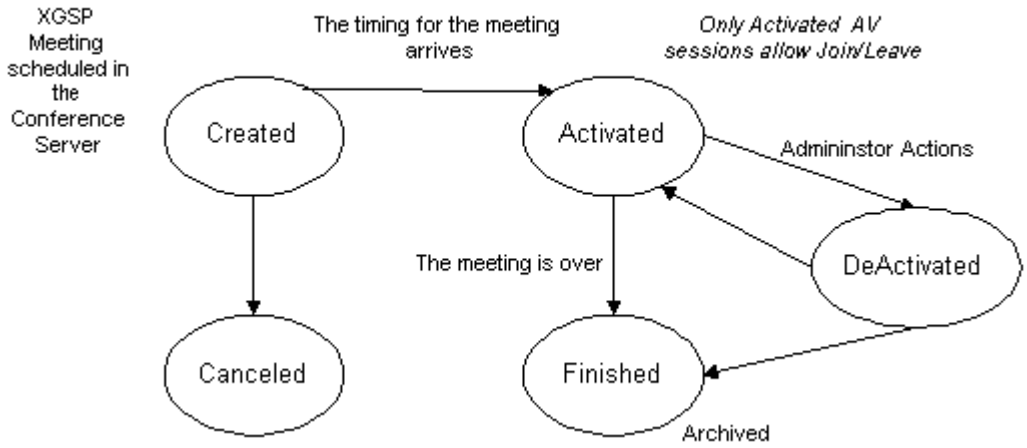

Figure 10 the state transition of XGSP audiovisual sessions

The XGSP AV session server implements the session management logics specified by the XGSP framework including: how to activate/deactivate XGSP audiovisual sessions, how to join/leave sessions, how to subscribe/unsubscribe AV streams and how to control multimedia service elements.

The AV Session Server has two important components: Session Manager and Media Server Manager. Session Manager handles starting/stopping/modifying XGSP AV sessions, and Media Server Manager can locate and start/stop media processing servers. Session Manager has AudioSession and VideoSession objects respectively for audio and video sessions. The objects keep the membership list of AV participants, the NaradaBrokering AV topic sets for publishing and the list of AV streams generated by these endpoints. They also handle the XGSP Join/Leave-AV-Session requests from the endpoints. To handling these requests, they usually talk to other system components to create the associated resources in media processing units and RTPLinks.

4.4.1 Activate and deactivate XGSP AV sessions

The Conference Scheduler in the XGSP Web Server sends an activation command to the AV Session Server to activate an AV session. The Session Manager in the Session Server creates the associated audio and video session objects. Then it uses a Media Server Manager to locate an AudioMixerServer and an ImageGrabberServer to start the AudioMixerSession and the ImageGrabberSession, respectively. 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. The Session Manager 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 and pushes the state of the session from "Created" into "Activated". If the administrator makes a request for creating a video mixing stream, a video mixer can also be added by exchanging messages with the VideoSession object.

4.4.2 Allow users to join and leave XGSP AV Sessions

After the session becomes activated, users can join the session by sending Join messages to the Session Manager. The manager handles join request by calling services in the audio and video objects separately.

When a speaker joins an audio session, 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 video session, it is assigned a topic number to publish its video stream. At the same time an image grabber is started to construct the snapshots of its video stream. This user is 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.

4.4.3 Provide XGSP AV sessions to different gateway servers

The AV Session Server also involves the management of XGSP sessions for other legacy clients like H.323, SIP and RealPlayer. Global-MMCS has H.323, SIP, RealStreaming Gateways for adapting H.323, SIP terminals and RealPlayers. The XGSP AV Session Server needs to collaborate with these Gateway Servers to deal with the session control layer problems in this heterogeneous collaboration system. The H.323 and SIP gateway transform H323 and SIP messages into XGSP signaling messages. The RealStreaming Gateway gets the encoding jobs from the Session Server and generates the RealMedia streams from the selected conferencing AV streams. The Section 4.5 and 4.6 give the description of these Gateway Servers.

4.5 Adapting H.323, SIP Endpoints: H.323 and SIP Gateway

To support H.323 and SIP audiovisual endpoints in XGSP AV sessions, bridging mechanism is necessary in both control layer and data transport layer. As discussed in Section 4.1.3, unicast RTPLinks are created for media transport of H.323 and SIP terminals. Some H.323 and SIP terminals like video phones which can only handle a single video/audio stream, need mixing services provided by MediaServers (Section 4.3). In the control layer, the H.323 and SIP gateway enable H.323 and SIP terminals to interact with other clients, and provides them the complete H.323 / SIP conference control services. Working with the XGSP AV session server, two functional components in the H.323 Gateway: H.323 Gatekeeper and H.323 MC provide the services including session call service, session registration service, and session control service to bridge H.323 terminals into XGSP audiovisual sessions. SIP Gateway also has the similar function components.

4.5.1 Session Registration Service

The H.323 Gatekeeper keeps the registration of H.323 terminals and the alias name of active sessions. This session alias list keeps the calling transport address of the H.323 MC for H.323 terminals. When a new AV session is created and activated at any time by the conference participants using the XGSP protocol, the H.323 MC registers the Session ID of this activated session as the session alias in the H.323 Gatekeeper. Any H.323 terminal joining this XGSP AV session must call the H.323 MC with the session alias. The H.323 Gatekeeper translates this session alias into the calling address and route the conference call to the H.323 MC.

4.5.2 Session Call Service

The session connection service enables H.323 terminals to join or leave XGSP audiovisual sessions. A H.323 terminal first make a H.225 call to the H.323 gateway with the *session ID*, which is routed to the registered H.323 MC. Three pieces of information are needed for establishing a call between two endpoints, namely the signaling destination address, media capabilities and media transport addresses at which the endpoints of both side can receive the media packets. H.323 MC processes the call, parses the information and exchanges XGSP XML messages with the AV Session Server when necessary. Figure 11 illustrates the translation between the H.323 call and the XGSP join-session procedure, which actually has three important steps: H.225 call setup, H.245 capability exchange and audiovisual logical channel creation.

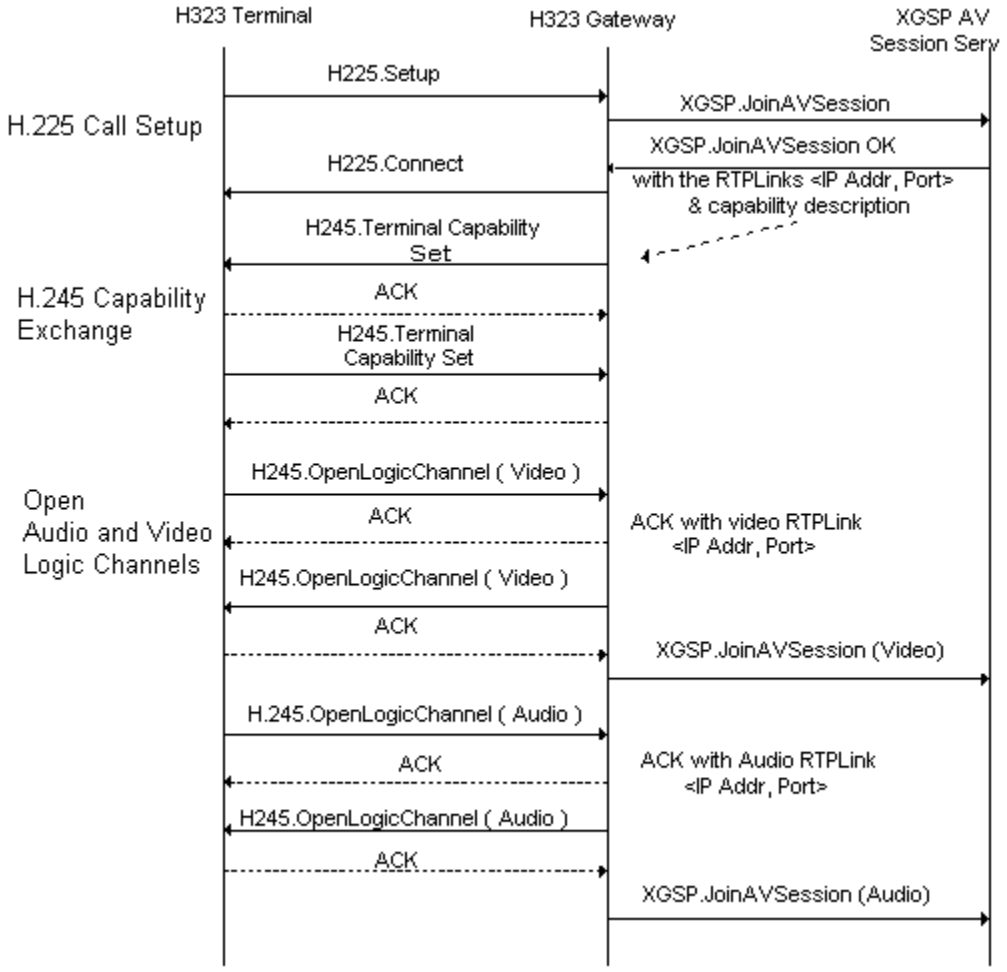

Figure 11. The translation of H.323 call into XGSP procedure

A H.323 call starts with a H.225.Setup message with the signaling destination address which actually is the SESSION ID. H.323 MC parses this message, and sends XGSP.JoinAVSession message with the SESSION ID to tell the XGSP AV Session Server that a H.323 terminal is connecting. If the Session Server decides to permit the joining request, it fills the information of audiovisual RTPLinks of the broker in the XGSP.JoinAVSession-OK message. As long as the H.323 gateway gets the *"OK"* response, it sends H.225.Connect back to the terminal to complete the H.225 procedure and keeps the RTP channel information for the phase of audiovisual logical channel creation.

The establishment of a H.323 conference involves the procedure of common media capability negotiation based on the H.245 media capability description of every connected terminal. XML is used for the capability description of Global-MMCS media services, which can easily be mapped into H.245 media capability description. The H.323 MC can keep two types of capability descriptor: a global capability descriptor for the whole system and specific descriptors for XGSP audiovisual sessions. In H.245 capability exchange procedure, H.323 MC set this common capability information in the connected terminal to force it uses the right media codec.

Creating logical channels for the AV streams just follows the exchange of capabilities and master-slave determination. The H.323 MC retrieves the terminal's transport address from the H.245.OpenLogicChannelACK, and sends XGSP.JoinAVSession (video and audio) with the address information to the Session Server. Since the H.323 Gateway has obtained the transport address of the receiving RTPLinks from XGSP.JoinSession-OK in the first phase, it can include the information into H.245.OpenLogicalChannelACK to notify the requesting H.323 terminal.

4.5.3 Session Collaboration Control

After the session call is over, a point-to-point H.245 control channel exists between this H.323 terminal and the H.323 Gateway. On the top of this H.245 control channel, our XGSP service allows H.323 users to vote for meeting chair, request floors, and make audio/video mixing and switching. The XGSP Node Manager can launch a H.323 Console, a basic XGSP AV control client for H.323 terminals. The XGSP session control services are achieved by interaction among the node managers, the H.323 consoles and the XGSP AV session server. The H.323 Gateway intercepts the XGSP messages and translates the procedure into H.245 control procedures.

Video switching enables a H.323 terminal with the limited render capability to display any number streams from the XGSP session. The H.323 console can copy a video stream list of the activated session from the XGSP AV session server and allows the user to make the video selection. Upon the selection, the H.323 console sends a XGSP.VideoSelection command to the session server to subscribe a video stream. The H.323 Gateway also receives the command and follows the procedure of H.243 to start video switching for the connected terminal.

Each user in a XGSP AV session may have different roles for floor control, like speaker and listener in an audio sessions, sender and viewer in a video session. The XGSP conference chair is allowed to assign roles dynamically in a public AV session by sending XGSP.SetRole messages to regular users through its XGSP Node Manager. Whoever gets this message must check the identity and notifies the local H.323 console of the action of role setting. At the same time, the H.323 Gateway also understands the message and starts a H.245 “Request Open/Close Logical Channel” procedure to open or close the AV logical channels of the associated H.323 terminals.

4.5.4. SIP Gateway

The services provided by the SIP Gateway are similar to the services of the H.323 Gateway. SIP Gateway can translate the SIP Messages including INVITE and BYE into XGSP messages in a more straightforward way because of the simplicity of SIP protocol. The procedure is: when the SIP gateway receives an INVITE request from a SIP client, it will send a Join Session message to the XGSP AV Session Server, and get the media description by parsing the SDP body in the INVITE message. After getting an admission response from the XGSP AV Session Server, the SIP gateway will reply to the SIP client with an OK message holding the media description of RTPLinks. And when the SIP gateway receives a BYE message, it will send a XGSP.Leave-AV-Session message to the AV Session Server and reply to the SIP client.

4.6 Adapting RealPlayer: RealStreaming Gateway

Streaming provides a framework to deliver media stream across Internet. A streaming client connects to a streaming server, primarily using Real Time Streaming Protocol [22] (RTSP), to establish a session and receive the stream. Streaming is primarily used for Media-On-Demand, receiving media that resides on a streaming server whenever a client wants to play. It is also possible to make a live streaming broadcast. The connection between the client and the server is stateful. RTSP is a client-server multimedia presentation control protocol. It provides VCR-like control, so that clients can pause, fast forward, reverse, and absolute positioning etc. Streaming media aims to improve the quality of service of the stream, by using various streaming codec to improve stream quality and a buffering mechanism to reduce the jitter.

The implementation of XGSP RealStreaming Gateway is different from other gateways, i.e. H.323 and SIP, H.323 and SIP gateways transform H.323 or SIP communication signals into XGSP control messages or vice versa so that H.323 and SIP endpoints can join XGSP AV sessions. After joining the sessions, they can use common conferencing codecs to interact with Access Grid nodes and XGSP AV portlets through RTPLinks. But a RealPlayer doesn’t need to communicate with those conferencing clients directly. It communicates with the Real Streaming server by using RTSP control channels and establishing a RTP data channels to receive stream in Real native AV format. Therefore, the essential job of the Streaming Gateway is to work as a customized RealStreaming Producer, converting H.261 video streams and Ulaw audio streams from XGSP AV sessions into RealMedia streams and uploading them as live streams onto the Streaming Server.

We used Helix Streaming Server, which is an open source version of the RealStreaming Server. Commercial and open source versions can be used interchangeably. The XGSP Streaming Gateway uses Helix Producer API to produce the RealMedia Stream for Helix Streaming Server. Components of XGSP

Streaming Gateway are composed of the following logical components: stream conversion handler, stream engine and SMIL file generator.

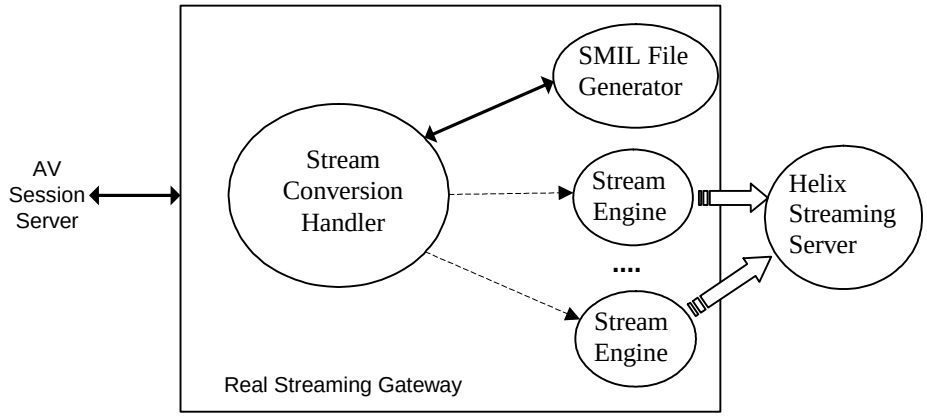

Figure 12 RealStreaming Gateway

4.6.1. Stream Conversion Handler

Stream Conversion Handler handles the communication between the XGSP AV Session Server and the Streaming Gateway. It keeps an internal database for the streams being converted. This database is updated when the streaming jobs are started or deleted. In order to start a streaming job, it initiates a Stream Engine for the requested stream and passes required parameters such as conversion format, helix server address, stream name, etc to the Stream Engine.

4.6.2. Stream Engine

This component can be considered as the most fundamental component of the Streaming Gateway. Stream Engine is responsible for converting the received audio or video streams into a specified RealStreaming format and pushing the converted stream to Helix Streaming Server. Streaming Engine is composed of two parts, RTP Handler and HXTA Wrapper. HXTA is a conversion engine provided by Helix Community [23] and converts raw audio and video data into RealStreaming formats.

RTP Handler receives audio and video packets from a local port provided by Stream Conversion Handler. The purpose of this unit is to transform the received packets into a format that HXTA can understand and be able to make the conversion to RealStreaming format. Raw audio and video data can be passed to HXTA Wrapper. There are several color spaces to video representation. But two of the most common are RGB (red/green/blue) and YCrCb (luminance/red chrominance/blue chrominance). HXTA accepts different formats of RGB and YCrCb. As the first conversion step, RTP Handler decodes the received video H.261 frames into YCrCb frames. We prefer YCrCb, because RGB requires more memory to represent video images compared to YCrCb. RTP Handler passes these decoded frames to HXTA Wrapper over a buffer. Audio ULaw packets are decoded into the raw WAV format before passed to HXTA Wrapper. RTP Handler also makes sure that packets are processed in a sequent order by dropping those late-arrival packets. RTP Handler gets the media type of the stream from the input provided by Stream Conversion Handler. Based on that information it either converts received packets into raw video or audio format. Each stream, whether it is audio or video, is converted independently from each other.

4.6.3. SMIL File Generator

Streaming Engine receives only one stream, whether it is audio or video, and produces one stream. In order to enable streaming clients to receive audio and video together, there is need for a SMIL file, which resides on the Helix Streaming Server side. Because of this, Stream Conversion Handler provides RTSP links of audio and video streams to SMIL file generator and SMIL file generator produces a SMIL file that includes those RTSP links. In our current implementation we have only one audio stream per session, which is mixed of all available audio streams in that session. So when a video stream is to be converted into streaming format, the RTSP link for the mixed audio stream is included to the generated SMIL file.

4.6.4. Streaming Gateway User Interfaces

Because the production of a RealMedia stream, especially for video streams, takes considerable CPU percentage, regular users should not be allowed to start and stop streaming production jobs. Two types of user interfaces to the XGSP Streaming Gateway are developed for administration users and regular users:

respectively. The Streaming Admin is implemented for administrative purposes and Streaming Client is for normal RealPlayer users. The Streaming Admin enables the administrator to choose active video streams from XGSP AV sessions and ask the Streaming Gateway to produce the associated RealMedia streams. The Streaming Client shows the list of available RealMedia streams and allows users to select one of them and play it from RealPlayer.

The XGSP AV specifies the messages among the streaming administrator, streaming client, XGSP AV Session Server and Streaming Gateway. The AV Session Server keeps the list of active RealMedia streams and notifies the user interfaces of the change in the list. The streaming admin interface sends the request from the administrator to the AV Session Server, and Session Server forwards the commands to the XGSP Streaming Gateway.

(1) Initialization : When streaming administrator first connects to the Session Server, it requests a list of the available streams and RealStream streams in the session. Streaming administrator sends RequestStreamList, to request all of the available audio/video streams and RealStreams, to request all of the available RealStream streams consecutively. In the reply, Session Server replies with RequestAllStreamsReply and RealStreamsReply. Streaming client only sends RealStreams message to receive available RealStream streams list.

(2) Create new RealMedia Streams: streaming administrator sends "JoinStream" for one of the streams chosen. Session Server adds some other fields to the same message and forwards it to Streaming Gateway. Streaming Gateway replies with "JoinStreamReply" and as a result Session Server generates "RealStreamEvent" messages with NewRealStream mode and sends them to streaming administrator and streaming clients.

(3) Delete new RealMedia Streams: Only instead of sending RealStreamEvent with NewRealStream mode, Session Server sends RealStreamEvent with ByeRealStream mode. For the case of InitializeRealGateway, Session Server forwards the message to Streaming Gateway and also sends RealStreamEvent with ByeRealStream mode for each of the streams removed.

4.7 Adapting Mobile clients

Mobile devices have limited capabilities such as limited bandwidth, processing and memory capability, small size screens. Due to this reason we cannot expect them to function like an AV client on a desktop PC. Although 3G wireless network fully supporting multimedia application hasn't been widely used, limited multimedia services for cellular users are already available such as Real Mobile Streaming and MMS. Appropriate gateway has to be introduced to enable cellular phone devices join a XGSP AV session to interact with other desktop clients. This Mobile Gateway can work with RealStreaming Gateway and Helix Streaming Server to provide suitable RealMedia streams for mobile clients. And it also transcodes the still images sent by a phone cameras into a video stream. Nokia 3650 [24], a popular GSM phone which has RealPlayer and Java MIDP 1.0 [25] installed, is a good candidate for our testing purpose.

4.7.1 Streaming to Cellular Phones

RealNetworks provides RealMedia formats for cellular phones as well. This RealMedia format is named as General Mobile Streaming which is 20 Kbps with 5 fps. The image size is also 160x120 pixels. RealPlayer on Nokia 3650 is capable of playing this streaming format.

The administration interface of the Streaming Gateway can initiate streaming jobs for cellular phone clients as well. AdminUI specifies General Mobile Streaming format when he/she selects the video to be converted. Streaming Gateway receives the selected video stream and the session audio. Both audio and video are converted and combined into one RealMedia stream with General Mobile Streaming format. Since the generated streams sometimes have too long RTSP URLs to be understood by cellular phone users, the Mobile Gateway generates .ram files specific for RealPlayer and provides links to those files in a XHTML page. Cellular phone users can visit this page through browser and launch RealPlayers by clicking one of the links. If the administrator stops mobile streaming jobs, the corresponding stream URLs are removed from the page.

4.7.2 Streaming from Cellular Phones

We developed a camera application using MIDP 1.0 and deployed it on Nokia 3650. The images produced by that application are 160x120 pixels. In order to send them to a XGSP AV session, we need to present these still pictures in a temporal order as continues video streams. The transcoding module in the mobile gateway is developed to achieve image-to-stream conversion which also resizes the images by a factor of 2 to produce a video stream with 320x240 pixels in size. The images are transported to this

gateway over an HTTP connection with a interval of 7 ~8 seconds, encoded into a H.263 stream with 2 fps and then pushed to the XGSP AV session. During the procedure, the conversion module reuses the same image until the next image is received. When the images uploading from cellular phone stops, that gateway simply sends an end-of-stream packet to the session and terminates.

5. Implementation and Performance

5.1 Software Release

In May, 2004, we have release the initial version of Global-MMCS. It has the software components including: Media Server package, Session Server package, Web Server package, H.323 Server package, Real Streaming package.

The whole system uses NaradaBrokering package for communication overlay. All the audiovisual processing services, including the video mixer, audio mixer, the image grabber servers as well as the front end of RealStreaming Gateway are developed using Java Media Framework [26]. Java Media Framework (JMF) package, which can capture, playback, stream and transcode multiple media formats, gives multimedia developers a powerful toolkit to develop scalable, cross-platform technology. Based on the codecs provided by JMF, our system can support ULaw audio codec, H.261 and H.263 video codec for all the AV endpoints. In the development, we also fixed some minor bugs inside JMF, integrated H.261 encoder and the video capture of shared display into it.

To implement the H.323 gateway, we use the protocol stacks from the open source projects, including OpenH323 [27] project. OpenH323 is an open source project providing H.323 protocol stack library.OpenH323 applications are built on top of the powerful network, inter-process communication and threading library PWLib. The source package also provides an ASN.1 parser that automatically generates code for parsing and generating H.323 PDUs from their standardized ASN.1 description. The handling and processing of signaling messages is encapsulated in an object oriented way which provides simple but powerful means for extension. The stack is organized using method calls and callbacks that are associated with the protocol and state transitions. We follow JAIN SIP [28] stack specified by Sun for SIP Gateway development. NIST [29] is an open source project and implements SIP JAIN library. The NIST package also includes the examples of Proxy-Registrar Server and Instant Messageing client. We use the NIST library to develop the SIP Gateway and build our own Proxy Server based on the source codes. Helix community open source project supported by Real Company enables the development of RealStreaming Gateway. We uses Streaming Server, which is an open source version of the RealStreaming Server. Commercial and open source versions can be used interchangeably. And we developed, RealStreaming Gateway based on Helix Real Producer SDK.

The AV Session Server is built to manage real-time AV sessions, receiving messages from gateways and the web server through different control topics on the NaradaBrokering. The XGSP web server based on Apache Tomcat provides an easy-to-use web interface for users to make meeting schedules, join XGSP conferences and for administrators to perform the tasks of the system management. The XGSP conference manager is implemented as an embedded server in the web container. It can create/destroy conferences, activate/deactivate AV application sessions and generate the active conference directory to all the users. Users should log into Global-MMCS through their web browsers and select active conferences.

The XGSP node manager is implemented in an applet running inside the Global-MMCS portal browser. The node manager will show up when the user joins the conference. Right now the node manager can report the membership in the XGSP conference. In addition, it has a few buttons for the available application endpoints, including: a Unicast JMF, H.323, Real Streaming and chat application portlet. The chat portlet provides the text chat collaboration. Depending upon XGSP AV servers, the Unicast JMF portlet can build up their AV stream list in a videoconference, allow the user to choose any number of video streams for receiving and rendering and is able to send audio and video streams.The H323 and Real Streaming portlets are the wrappers for H.323 terminals and RealPlayers, supporting a single video selection and rendering in their particular clients.

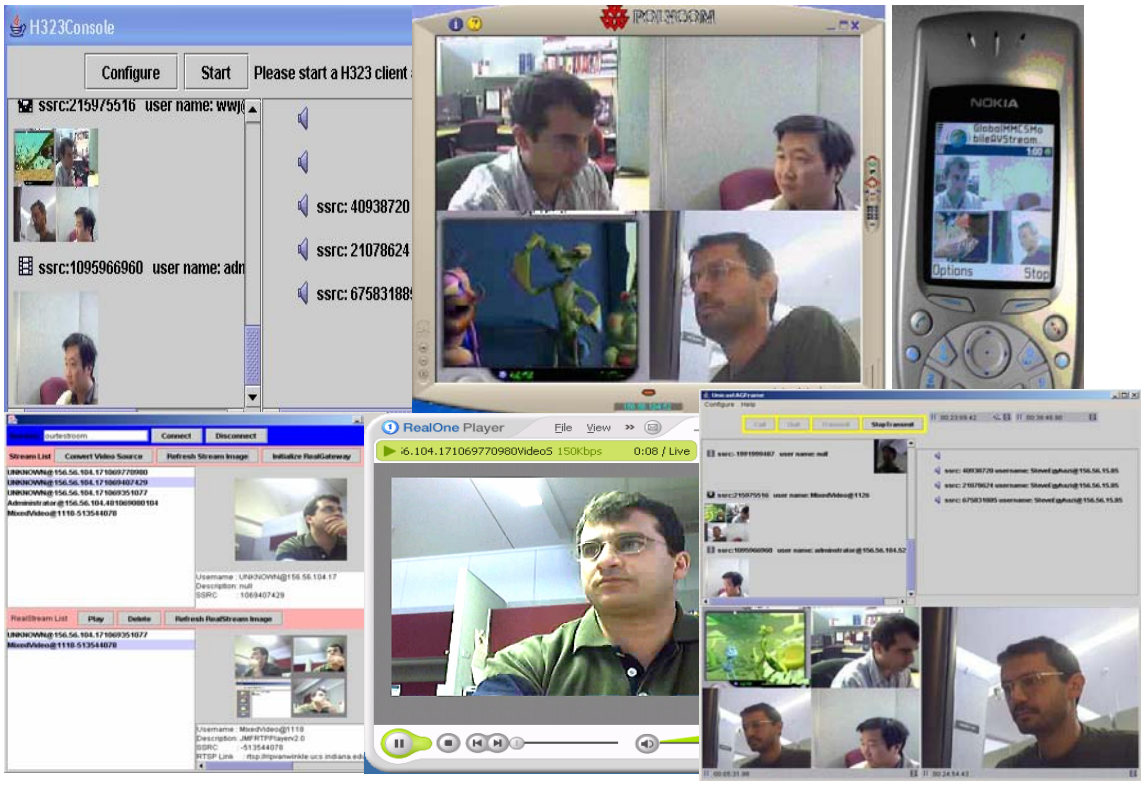

Figure 13 Global-MMCS User Interface

Right now, the Unicast JMF portlet is able to send AV data through NAT boundary by using NaradaBrokering UDP connection. By opening some specific ports, it can also go through firewall. Further work like AV data transmission over HTTP tunnel is still going on. We also tried to integrate our Global-MMCS portal and XGSP node manager into OGCE portlet server. There are still some issues including unifing user authentication mechanism and applet reloading. We are planning to use Web-start mechanism to replace the applet launching.

The whole Global-MMCS package is still under extensive test and improvement. JMF framework is being enhanced in its performance and codec plugins and extended to other platform including Linux and Mac OSX. We also working hard to introduce the archiving and replay service to build a more dependable and robust collaboration system. P2P style needs to be applied for multimedia services to achieve more general scalability even without enough server-side computation resource.

5.2 Performance of NaradaBrokering for Audio/Video Delivery

We conducted extensive tests to evaluate the performance and the scalability of the NaradaBrokering broker network in the context of audio/video delivery. We investigated the performance of both a single broker and the distributed broker network.

5.2.1 Performance Tests for One Broker

Since the building blocks of the distributed broker network are brokers, it is essential to know thoroughly the capacity and the limits of a single broker. Knowing the capacity and the performance of a single broker helps us to predict the performance of the broker network in distributed settings. In addition, it helps us to identify the bottlenecks and problems in multi broker environments. We tested two cases thoroughly. The first one is single large scale meetings. The second one is multiple small scale meetings.

(1) Single Meeting Tests

We tested the performance and the scalability of a single broker for three types of single meetings: single audio meetings, single video meetings, and audio and video combined meetings. It supported 1500 users on a single audio meeting with one speaker. The audio stream was 64kbps ULAW. The machine was a Linux machine that had Double Intel Xeon 2.4GHz CPUs, and 2GB of memory.

It also supported 400 users in a single video meeting on the same machine. The video stream was an H.263 stream with 280kbps bandwidth on the average. On the other hand, when there were one audio and one video meeting at the broker, it supported close to 400 users in both meetings. This was due to the better utilization of the broker when there are two meetings. **Table** shows the summary of the test results from the video meeting when there are one audio and one video meeting on a single broker. Each row of the table shows a test case. First column shows the number of users on each audio and video meeting. The second column shows the average latency of video packages delivered to the first user in the meeting. Since the broker routes packages in the first-come-first-serve manner. Therefore, the order of subscription of the clients is important. The broker delivers each package first to the client who joined the meeting first, and it delivers it last to the client who joined the meeting last. Sixth column shows the percentages of packages that arrive more than 100ms later to the last client. We require the broker not introduce more than 100ms of latency to provide good quality. When the percentages of late arriving packages go beyond 1.0%, we assume the quality of the communication is not good enough. Therefore, in this case, the broker supports 300 users in both audio and video meetings with excellent quality. However, there are more than 1.0% late arriving packages for 400 users. This shows that the broker can support up to 400 users in audio and video combined meetings.

number of clients	first latency (ms)	middle latency (ms)	last latency (ms)	Avrg. Latency (ms)	Avrg. Jitter (ms)	Last Late arrivals	Avrg. Late arrivals	In BW Mbps	Out BW Mbps
12	1.3	1.4	1.5	1.4	0.5	0	0	344	4.13
50	1.7	2.2	2.7	2.2	0.8	0	0	344	17.2
100	3.7	4.6	5.7	4.7	2.1	0	0	344	34.4
200	8	10.1	12.2	10.1	5.4	0	0	344	68.8
300	11.3	14.2	17.2	14.2	7.5	0	0	344	103.2
400	18.1	22.1	26.1	22.1	10.9	% 1.37	% 1.25	344	137.6
500	26.7	31.7	36.8	31.8	13.5	% 5.5	% 4.7	344	172.0
600	169.2	175.3	181.4	175.3	16.2	% 59.9	% 58.2	344	206.4

Table 3 Video test results for single audio and video combined meetings

(2) Multiple Meeting Tests

We also tested the performance of a single broker with multiple smaller meetings. The most important outcome of these tests was the fact that the broker was utilized much better for multiple smaller meetings than single large size meetings. It supported higher number of participants with smaller latency and jitter values. **Figure** 14 shows the average latency values of single video meeting tests and multiple video meeting tests for the same number of participants. In multiple video meeting cases, all video meetings had 20 participants. As the latency values show the average latencies of multiple video meetings are much smaller.

Similarly, the jitter values and loss rates are also much smaller. Therefore, the broker was able to provide services to 700 participants in 35 video meetings with very little late arriving packages. It was able to support only 400 participants in the single video meeting test.

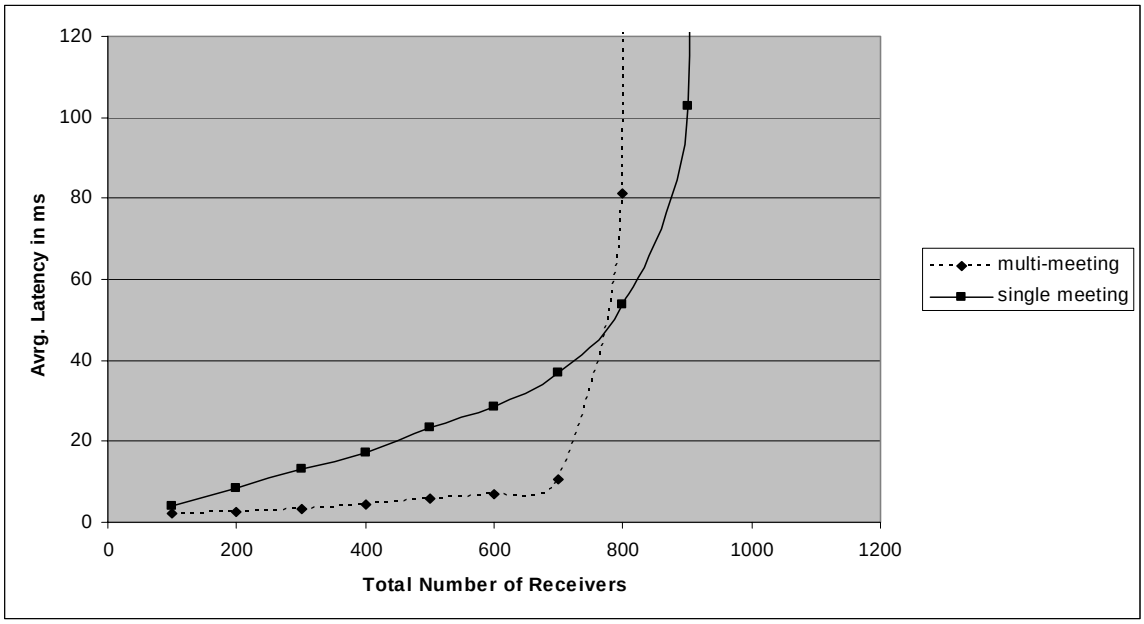

Figure 14 Average latencies of single and multiple video meetings

We also tested multiple audio and video meetings concurrently. All meetings had 20 participants and one transmitter. There were equal numbers of audio and video meetings for each test. **Table** shows the summary of the results from audio meetings and **Table** shows the summary of the results from video meetings. First columns of these tables show the total number of participants in these meetings. Half of these participants are audio meeting participants and the other half are video meeting participants. Similarly the second columns show the total number of concurrent meetings. Half of them are audio meetings and the other half are video meetings. The latency values of audio meetings are a little smaller than the latency values of video meetings, because we give priority to audio package routing at the broker. There are no late arriving packages until 1000 participants. These results demonstrate that 40 meetings (20 audio and 20 video meetings) can be conducted simultaneously on this broker with excellent quality. In addition, these tests show that the routing of audio packages do not delay the routing of video packages significantly.

Number of Clients	number of meetings	Avg. latency (ms)	Avg. Jitter (ms)	Avg. Late arrivals	In BW (Mbps)	Out BW (Mbps)
200	10	1.7	0.5	0	1.755	35.1
400	20	2.5	0.9	0	3.51	70.2
600	30	3.3	2	0	5.265	105.3
800	40	4.9	2.2	0	7.02	140.4
1000	50	46.8	2.8	%16	8.775	175.5
1200	60	9287	6.6	%100	10.53	210.6

Table 4 Audio results from audio and video combined multi meeting tests

Number of Clients	number of meetings	Avg. latency (ms)	Avg. Jitter (ms)	Avg. Late arrivals	In BW Mbps	Out BW Mbps
200	10	2	0.7	0	1.755	35.1
400	20	2.62	0.85	0	3.51	70.2
600	30	5.25	1.3	0	5.265	105.3
800	40	6.5	1.56	0	7.02	140.4
1000	50	76.2	1.96	%23	8.775	175.5
1200	60	9421	4.12	%100	10.53	210.6

Table 5 Video results from audio and video combined multi meeting tests

5.2.2 The Performance Tests for Distributed Brokers

Similar to single broker tests, we evaluate the performance and the scalability of the brokering network for both single large size meetings and multiple smaller size meetings. We conducted these tests in controlled settings to measure the performance of the broker network accurately.

(1) Single Meeting Tests

Inter-broker package delivery is very limited in broker network when there is a single meeting. Only one stream is exchanged among the brokers. This simplifies the analysis greatly. The most important factor that affects the performance of the broker network in distributed settings is the amount of overhead put to packages that travel to other brokers. We minimized this overhead by giving priority to packages that travel to other brokers. Each broker routes packages first to other brokers and then to local clients. In addition, when multiple packages arrive simultaneously, later packages do not need to wait the earlier ones to be routed to all local clients. Instead, there are two layer queues at the broker. First queue is used to hold arriving packages and route them to the other brokers. Then, packages are placed into the second queue, to be routed to the local clients. This mechanism ensures that the broker puts minimum overhead to packages that travel to other brokers.

We setup a broker network with four brokers in two Linux clusters Figure **1**15. We tested the performance of this broker network for a single video meeting. On user published the video stream through the first broker and equal numbers of users received that stream through each broker. The nodes of the first cluster had 2.4GHz Dual Intel Xeon CPU and 2 GB of memory. The nodes of the second cluster had 2.8 GHz Dual Intel Xeon CPU and 2GB of memory. Both clusters had gigabit network bandwidth among its nodes.

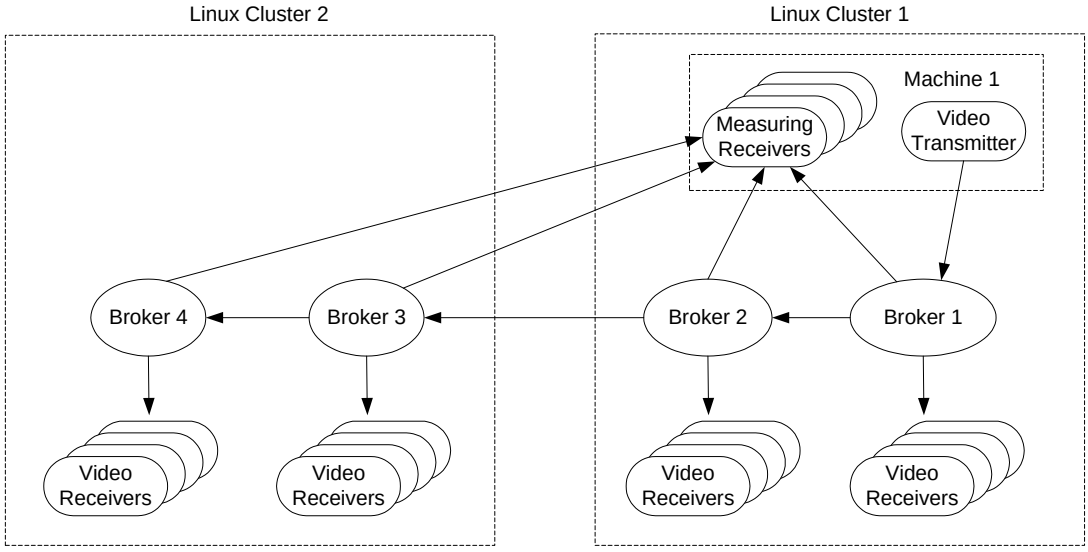

Figure 15 Single video meeting tests with four brokers

Table shows the latency results of these tests. Two blocks of latency results show the average latency values of the first and the last receivers from each broker for the 5610 video packages exchanged. We have results from all four brokers. The latency values of broker1 and broker2 are very similar to each other. Similarly, the latency values of broker3 and broker4 are very similar. Last two brokers perform better than the first two brokers, because the machines in the second Linux cluster have superior CPU power.

As the latency values show, adding new brokers increases the capacity of the broker network as much as the capacity of the added machines. In this case, since all brokers have very similar computing power, each broker increases the capacity of the broker network almost linearly. Table shows the percentages of late arriving packages. For the first two brokers, the percentage of late arriving packages is %1.9 when there are 400 participants. Therefore, they can support up to 400 users. For the last two brokers, the rate of late arriving packages are less than %1.0 for the same number of participants. They support 400 users comfortably. In total, four brokers support close to 1600 participants in a single video meeting.

Users per broker	Users in total	Latencies of first receivers (ms)				Latencies of last receivers (ms)			
		B1	B2	B3	B4	B1	B2	B3	B4
200	800	8.06	8.44	8.58	9.03	12.57	12.93	12.39	12.79
300	1200	12.04	12.5	11.85	12.37	18.78	19.25	17.55	18.03
400	1600	16.45	16.94	15.26	15.95	25.47	26.02	22.84	23.55
500	2000	21.43	22.38	19.07	19.92	32.61	33.72	28.53	29.38
900	3600	665.2	798.4	86.9	72.9	685.7	818.9	104.53	90.17

Table 6 latency results for single video meeting tests with four brokers

Users per broker	Users in total	Broker1 late arrivals (%)	Broker2 late arrivals (%)	Broker3 late arrivals (%)	Broker4 late arrivals (%)
200	800	0	0	0	0
300	1200	0.29	0.27	0	0.02
400	1600	1.87	1.92	0.73	0.73
500	2000	4.01	4.43	2.41	2.4
900	3600	93.54	93.85	37.84	31.31

Table 7 Percentages of the late arriving packages for the last users in single video meetings

In summary, these tests demonstrate that NaradaBrokering broker network scales well in distributed settings when delivering audio and video streams to high number of participants in large scale meetings. The scalability of the broker network increases almost linearly by the number of brokers. The overhead of going through multiple brokers for a stream is not significant, since inter-broker routing has priority over local client routings.

(2) Multiple Meeting Tests

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. As we have seen in the single broker tests with multiple video meetings in the previous section, conducting multiple concurrent meetings on a broker can increase both the quality of the service and the number of supported users. This can also be achieved in multi broker setting as long as the size of these meetings and the distribution of clients among brokers are managed properly. 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 significantly. 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. To investigate this, we conducted multiple video meeting tests with two different meeting sizes.

We used the same broker organization scheme as the single video meeting tests in the previous section. There were four brokers connected as a chain. In this case, all brokers were running in the same Linux cluster that has 8 identical nodes with 2.8 GHz Dual Intel Xeon CPU and 2GB of memory.

We tested with multiple video meetings each having 20 receivers. One client was publishing the video stream on a broker and 20 clients were receiving it. We distributed the clients of each meeting among brokers evenly. 5 clients joined each meeting through each broker. We also distributed the video transmitters of all meetings evenly among the brokers. There were equal numbers of transmitters publishing video streams to each broker. We collected the performance data from three meetings. The publishers of these three meetings were publishing their streams through the first broker. For each of these three

meetings, we collected the results from 4 receivers, each one getting the stream from a different broker. **Table** shows the average latency and jitter values of three meetings. **Table** shows the percentages of lost packages and the percentages of late arriving packages.

Number of meetings	Total Users	Latencies from 4 brokers (ms)				Jitters from 4 brokers (ms)			
		B1	B2	B3	B4	B1	B2	B3	B4
24	480	3.30	4.24	5.32	5.87	1.39	1.44	1.65	1.72
48	960	2.98	5.04	6.90	8.20	1.84	2.53	2.82	3.00
72	1440	4.83	13.66	17.03	17.52	1.70	3.04	3.52	3.51
96	1920	5.76	25.55	52.77	47.34	1.69	3.70	5.28	5.52

Table 7 Average latency and jitter values for multiple video meeting tests.

Number of meetings	Total Users	Loss rates from 4 brokers (%)				Late arrivals from 4 brokers (%)			
		B1	B2	B3	B4	B1	B2	B3	B4
24	480	0.01	0.00	0.00	0.00	0.00	0.00	0.03	0.03
48	960	0.00	0.06	0.11	0.22	0.00	0.11	0.09	0.17
72	1440	0.02	1.20	1.60	1.63	0.00	0.01	0.03	0.04
96	1920	0.13	6.41	19.62	19.68	0.00	0.13	2.79	0.91

Table 8 Average loss rates and late arrivals for multiple video meeting tests

Since the publishers are publishing the streams through the first broker, the latency values for the first broker are the smallest. Latency values increase when the streams travel more hops along the way from the first broker to the last. The broker network provides excellent quality communication when there are less than 72 meetings. The latency values and jitters for all brokers are very small. There are minor package losses and late arriving packages. For 72 meetings, the latency values and jitters are still very small. There is also very few late arriving packages. However, there are a little more than %1.0 lost packages. When there are 96 meetings, significant amount of packages are lost. Therefore, the broker network can support up to 72 meetings or up to 1440 participants in total. This number is slightly smaller than the single video meeting case, in which the broker network was able to support up to 1600 participants.

When we compare the scalability of the broker network with the scalability of the single broker in multiple video meeting tests, the number of supported participants increased two times. The single broker supported 700 participants in 35 video meetings, each having 20 users. In this test, four brokers supported around 1440 participants in 72 meetings, each having 20 users. As we can see, the increase on the number of brokers did not result in a linear increase on the number of supported participants. There are two reasons for this. First one is the overhead of inter-broker stream delivery in distributed setting. Now, the brokers deliver streams not only to clients but also to other brokers. The second one is the smaller number of participants in each broker for each meeting. Each incoming package is delivered to only 5 users in the distributed setting, while it was delivered to 20 users in the single broker case.

Since the small number of participants joining meetings through each broker reduced the scalability and the quality of the service provided, we tested with a larger meeting size to observe the difference. This time, 10 participants joined each meeting through each broker. Therefore, the sizes of meetings were 40. All other aspects of the test were the same as the previous test. In this case, the number of supported clients increased significantly. 48 meetings with 1920 participants in total are supported with excellent quality, compared to 1440 participants in the previous test with meeting sizes of 20. In addition, the quality of the service provided by the broker network also increased considerably. The average latency and jitter values are much lower. The late arriving packages and losses are very small, too. The main reason for the better performance is the better utilization of the broker network. Now, there is less stream exchange among brokers and each incoming stream is delivered to more participants by every broker.

5.3 Media Service Performance

We investigate the performance of media services in controlled experiments. All the service providers running in a single server are demanded to create more service instance so that the overhead increase in the terms of CPU and memory usage in the server could be illustrated. Although there is some difference in the computation overhead generated by services, they are all computation intensive tasks. Since individual service instances can run independently and be attached to different XGSP AV sessions, it is quite easy to distribute them into hosting servers. Provided enough media service resources, the scalability of the Global-MMCS can be guaranteed.

5.3.1 Audio Mixing

Number of audio 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	Some loss, Negligible

Table 9 Audio mixer performance test

The performance of an AudioMixerServer has been tested 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 Windows XP machine with 512 MB memory and 2.5 GHz Intel Pentium 4 CPU. The broker was running on another machine in the same subnet.

The number of speakers in the test in a mixer is not less than the average number of speakers in meetings. Usually there is one active speaker in a session and there are less than 6 speakers. Therefore, we assume that this setting represents at least average meetings. Table 9 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. Therefore, on the average one machine may support around 10 mixing sessions.

5.3.2 Video Mixing

Video mixing is a computing intensive process. One video mixer decodes four received video streams and encodes one video stream as the output. In Table 10, it shows that a Linux machine with 1 GB memory and 1.8GHz Dual Intel Xeon CPU, can serve 3 video mixing streams 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.

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

Table 10 Video mixer performance test

5.3.3 Image Grabbering

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

40	60	192
50	70	232

Table 11 Image grabber performance test

Image grabbing is also a computing intensive task. Each image grabbing includes decoding, resizing and encoding of a video stream. Though, resizing and encoding do not have to be done continually. They can only be performed when it is time to get the snapshot. Table 11 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. Although the number of video streams and the format of the streams change, if we assume that meeting have 10 video streams on the average, Table 11 shows that one machine can serve 5 meetings.

5.3.4 RealStreaming Conversion Performance

Stream conversion is a CPU intensive application. In order to see the CPU and memory usage of this conversion we also observed the effect of number of streams converted on CPU and memory usage. Streaming Gateway is running on the XP machine with 512 MB memory and 2.26GHZ Intel Pentium 4 CPU. Table 12 provides approximate CPU and memory usage for producing RealMedia streams. As the number of streams converted increases the CPU usage and memory usage also increases. In this specific machine we could successfully convert 4 streams without causing quality of services decrease. If we increase the number of streams, other streams are also affected and some time later the conversion performance reduces much. In this test we kept the number of frames per second high. This frame rate is normal in an Access Grid session.

Number of Streams	Frame rates of streams involved	CPU Usage Range	Memory Usage Range
1	23 fps	%10 - %25	29 MB
2	23 fps, 25 fps	%22- %40	34 MB
3	23 fps, 25 fps, 26 fps	%50 - %75	43 MB
4	23 fps, 25 fps, 26 fps, 16 fps	%65- %95	53 MB

Table 12 RealStreaming Gateway performance test

5.4 Scalability Discussion

On the basis of measurement in controlled experiment environment which consists of at most 4 brokers, we can discuss the scalability of the system in a more general way and larger scale. Consider scenarios where users are connected in collaborative sessions such as supported in Access Grid rooms or text chat sessions. All communication is performed by NaradaBrokering installed as a mixture of standalone brokers, handlers/plug-ins for clients or Web services.

The software routing is performed by the handlers and the brokers. The routers fall into two groups: Geographical routers handle traffic in a particular region and Functional routers cope with particular capabilities – in this case the different rooms. Hardware multicast is one approach to geographical multicast. However note that we ignore the possibility of hardware multicast below and assume software multicast where this is needed. Note that we have very efficient communication between institutions and hardware multicast would only be useful internal to each organization where it could add quite a bit of value and be easier to implement than globally across institution. NaradaBrokering supports hybrid protocols that mix hardware unicast and multicast. The network architecture is designed to make communication traffic as reliable and low bandwidth as possible.

Note we replicate the routers for each room at each institution so as to minimize traffic between institutions by using software multicast only within each institution. We give a more detailed analysis below ignoring any Geographical routers. These could be implemented in P2P fashion using NaradaBrokering handlers on the clients.

Assume there are N Institutions, R XGSP AV sessions and M participants per "session". Let each person send out one native AV stream and receive $4+X$ streams. Let each client mix the (first) 4 streams to make one composite mixed stream. The other X streams can either be other mixed streams or native webcam streams. Clients are also capable of generating these thumbnails of the received streams and

publish over the NaradaBrokering. Note we are mixing and image grabbing in P2P mode as more CPU power on clients. One could mix on servers but currently low end Linux server can produce up to 4 mixed streams where each mixed stream made up of 4 native streams. Thus one needs ***M*R/4*** servers for mixing to make the MR composite streams imagined here. The clients can display any of received streams and/or the mixed stream that it makes itself. And it can send out 2 streams including the native stream from its webcam and mixed one it makes.

Assume each person can only in one room. We configure S servers at each institution each handling ***R/S*** rooms. So the traffic at each server is receiving ***2*M*** streams and dispatching ***(N-1)* 2*M/N*** streams to remote sites. Here we assume uniform distribution ***M/N*** people per session per site, so each session dispatches ***(4+X)*M/N*** streams to local clients. Thus total traffic per session for the server is ***4*M +(2+X)*M/N*** streams.

We can illustrate the result in a particular example: let R= 50, S= 25, N=2, M=20**,** X=4. Assume the bandwidth of each stream is 1/3 Mbps. Therefore each Server has 93 Mbps traffic summing input and output bandwidth which has basically reach the peak rate in a server with 100 Mbps network interface. Each client needs to handle up to 3.3 Mbps traffic. Each Institution has 50 servers and 500 clients, so we get totally 50 servers and 1000 clients.

Based on the measurement result in Section 5.3, one regular server can usually run up to 20 audio mixers. We need other 3 or 4 audio servers for 50 sessions. Many rooms can share a server for their session control Web Service and we can estimate that a safe number is 500 users per session server. Two session servers and two H.323/SIP gateways should be enough for 1000 users. If for each session we need to generate a RealMedia stream simultaneously, at least 12 servers are necessary for 50 sessions. To support thousands of streams and dozens of sessions, media services demands the computation resources in dozens.

6. Conclusion and Future Work

In this paper we have presented the design principle and experience of building a scalable and integrated Web-Services Collaboration system. This collaboration system is developed based on the XGSP collaboration framework and NaradaBrokering messaging middleware. Such a service-oriented collaboration environment greatly improves the scalability of traditional videoconferencing system, benefits customers using diverse multimedia terminals through different network connections and simplifies the further extension and interoperability.

Further works include: enhancing the performance of media processing, extending the scalability of the system to 10,000 users, adding archiving and replay services, customizing the system for different application scenarios like collaborations in e-sports and e-science.

7. Reference

[1] ITU. Recommendation H.323 (1999), Packet-base multimedia communications systems.

[2] J. Rosenberg et al. (2002) “SIP: Session Initiation Protocol”, RFC 3261, Internet Engineering Task Force, http://www.ietf.org/rfc/rfc3261.txt.

[3] Access Grid (2003), http://www.accessgrid.org

[4] W. Wu, G. C. Fox, H. Bulut, A. Uyar, H. Altay, “Design and Implementation of A Collaboration Web-services system”, Journal of Neural, Parallel & Scientific Computations, Volume 12, 2004.

[5] Global Multimedia Collaboration System (Global-MMCS), http://www.globalmmcs.org

[6] Handley, M., Crowcroft, J., Bormann, C. and J. Ott (2002) The Internet Multimedia Conferencing Architecture, Internet Draft, draft -ietf-mmusic -confarch-03.txt.

[7] Bormann, C., Kutscher, D., Ott, J., and Trossen, D. (2001). Simple conference control protocol service specification. Internet Draft, Internet Engineering Task Force, Work in progress.

[8] ITU. Recommendation H.225(2000), Calling Signaling Protocols and Media Stream Packetization for Packet-based Multimedia Communication Systems.

[9] ITU. Recommendation H.245(2000), Control Protocols for Multimedia Communication.

[10] ITU. Recommendation H.243(2000), Terminal for low bit-rate multimedia communication.

[11]] ITU. Recommendation T.120(1995),Multipoint Data Conferencing and Real Time Communication Protocols, 1995

[12]Koskelainen P., Schulzrinne H. and Wu X.(2002), A SIP-based Conference Control Framework, *NOSSDAV’02*, May 12-14, 2002, Miami Beach, Florida, USA.

[13] Wu, X., Koskelainen P., Schulzrinne H., Chen C (2002). Use SIP and SOAP for conference floor control.Internet Draft, Internet Engineering Task Force, Feb. 2002. Work in progress.

[14] Virtual Rooms Video Conferencing System (2003), www.vrvs.org

[15] Kazaa, http://www.kazaa.com
[16] Skype, http://www.skype.com/
[17] ITU. Recommendation T.124 (1995) Generic conference control, 1995.
[18] Geoffrey C. Fox and Shrideep Pallickara (2002). "The Narada Event Brokering System: Overview and Extensions", proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)
[19] Java Message Service (JMS) , http://java.sun.com/products/jms/
[20] Web Services Reliable Messaging, http://www.oasis-open.org/committees/wsrm/charter.php
[21] Hasan Bulut, Shrideep Pallickara and Geoffrey Fox Implementing a NTP-Based Time Service within a Distributed Brokering System ACM International Conference on the Principles and Practice of Programming in Java, June 16-18, Las Vegas, NV
[22] Real Time Streaming Protocol (RTSP), http://www.ietf.org/rfc/rfc2326.txt
[23] Helix Community Project (2002), http://www.helixcommunity.org
[24] Nokia 3650 Phone, http://www.nokia.com/nokia/0,,2273,00.html
[25] Sun Microsystems, Mobile Information Device Profile 1.0 http://java.sun.com/products/midp/
[26] Sun Microsystems, Java Media Framework 2.1, (2001), http://java.sun.com/products/javamedia/jmf/2.1.1/index.html.
[27] OpenH323 Project (2001) , http://www.openh323.org
[28] JAIN SIP, http://jcp.org/en/jsr/detail?id=125
[29] NIST SIP (2001), http://snad.ncsl.nist.gov/proj/iptel/.