Malaysian Journal of Computer Science, Vol. 10 No. 2, December 1997, pp. 1-16

A FLEXIBLE AND RELIABLE DISTRIBUTED MULTIMEDIA SYSTEM FOR MULTIMEDIA INFORMATION SUPERHIGHWAYS

James Won-Ki Hong, Tae-Hyoung Yun, Ji-Young Kong and Young-Mi Shin
Dept. of Computer Science and Engineering
POSTECH
Pohang Korea
{jwkhong, thyun, konga, dry}@postech.ac.kr

ABSTRACT

Multimedia applications are being developed and used for many aspects of our lives today. New high-speed, broadband networks have emerged and made the operation of these high-bandwidth requiring applications readily feasible. However, the development of distributed multimedia applications and efficient and reliable operation of these applications are still very difficult. This paper presents a flexible and reliable distributed multimedia system that provides a rich multimedia API and that provides distributed multimedia services which can be used to develop a variety of multimedia applications easily. The system as well as applications running on it are managed and controlled in order to provide a reliable and efficient multimedia operations environment. We validate our claim by developing a number of multimedia applications using our distributed multimedia system and by using them for supporting distributed collaborative research experiments on top of a high-speed, broadband information superhighway.

Keywords: ***distributed multimedia system, multimedia applications, information superhighway, distributed multimedia services, multimedia communication, collaborative research, operation and management, CORBA, application management***

1.0 INTRODUCTION

Multimedia applications are beginning to play an important role in various aspects of our lives, including education, business, health care, publishing, and entertainment. The recent advances in computing and networking technologies have fueled the emergence of these applications requiring real-time processing and high bandwidth. In order for these applications to be truly useful and effective, they must be able to operate in a distributed fashion, covering users possibly located in geographically distant locations. Many national and international information superhighways [1, 2, 3, 4, 5, 6] being developed today can be used to support collaboration of distant users using multimedia applications, exchanging various multimedia data. The real problem in making the national or international users collaborate using multimedia applications, as if they are in a same physical location, is providing a system that can be used to develop multimedia applications easily and supporting the operation of these applications in an efficient and reliable way.

One of the major problems in supporting the development and operation of such applications lies in the operating systems of the computers being used. Most conventional operating systems running on these computers have not been designed to support multimedia systems that impose stringent constraints. Multimedia applications also require various forms of communication (such as conferencing, collaboration, multicasting and so on). Further, most of these operating systems have not been designed to support applications operating in a distributed fashion. In order for the system and applications to operate efficiently and reliably, they must be monitored and controlled properly at run time. As the number of user increases, the underlying system must be able to cope with the increased resource requirements. What is needed is a solution that is easily scaleable.

Much research work related to multimedia systems has been done [7, 8, 9, 10, 11]. Most of these systems fall short of meeting the requirements mentioned above. In this paper, we present a distributed multimedia system called MAESTRO [12, 13, 14] that is flexible enough to support the development of a variety of multimedia applications easily and that can support efficient and reliable operation of the applications. MAESTRO has been designed to support a wide variety of multimedia applications (such as video/audio conferencing, multimedia-on-demand, whiteboard, electronic notebook, telemedicine, teleshopping, etc.) that can operate in a distributed environment in mind.

The heart of MAESTRO lies in the multimedia application programming interface (API) and multimedia services that can be used to develop, operate and manage various multimedia applications. The multimedia API uses the set of distributed multimedia services MAESTRO provides for the operation and management of distributed multimedia applications. MAESTRO multimedia services include *communication service* for supporting various forms of communication (1-to-1, 1-to-N, N-to-M), *session service* for managing multimedia sessions, *naming service* for naming and locating objects, *storage/retrieval service* for storing and retrieving multimedia data, and

management service for monitoring and controlling services and applications. The API as well as the underlying services, has been modeled and implemented using the object-oriented approach. OMG CORBA [15] has been used as the middleware for mainly supporting interactions between distributed application components.

As a validation of our claim, we have developed a number of distributed multimedia applications using MAESTRO for a national information superhighway project in Korea. In this project, distant researchers (mechanical engineers, chemical engineers and chemists) collaborate in performing various research experiments simultaneously and collect, analyze and view the results using the multimedia applications developed using and running on MAESTRO.

The remainder of this paper is organized as follows: Section 2 presents the architecture of MAESTRO. Section 3 presents the multimedia application programming interface (API). Section 4 describes the distributed multimedia services provided by MAESTRO. Section 5 presents the implementation details of MAESTRO. Section 6 describes the national information superhighway project that uses MAESTRO briefly and presents the applications we have developed for the project. We summarize our work and discuss some possible future work in Section 7.

2.0 MAESTRO ARCHITECTURE

In this section, we present the layered architecture of MAESTRO as illustrated in Fig. 1. At the top layer, there exist various multimedia applications (video/audio conferencing, video-on-demand, telemedicine, whiteboard, electronic notebook, etc.) which use the multimedia API.

The second layer provides multimedia API that is used by various multimedia applications. This layer has been designed using the object-oriented approach and consists of class definitions that can be used to generate and transfer multimedia data between multimedia devices. This layer is used by multimedia application components, which are scattered throughout a distributed environment and collectively form a distributed multimedia application.

The third layer consists of distributed multimedia services that are needed to support the operation and management of distributed multimedia applications. This layer has also been designed using the object-oriented approach and consists of services, such as Communication Service, Session Service, Naming Service, Storage/Retrieval Service and Management Service. These services are used by multimedia applications at run time such that they allow multimedia applications to discover their source or destination associations, create/join/destroy sessions, exchange multimedia data, store or retrieve multimedia data and be monitored and controlled.

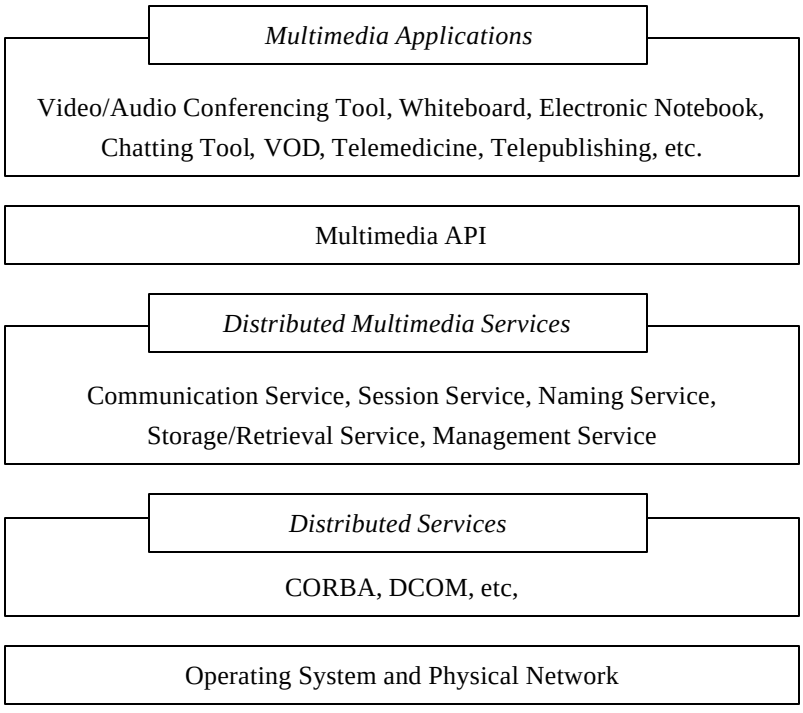


Fig. 1: Layered architecture of MAESTRO

At the fourth layer, there exist distributed services that provide transparency to distributed multimedia services and applications. Although any existing middleware service can be used in this layer, distributed object technologies such as CORBA [15] or DCOM [28] may be suitable. At the lowest layer, there exist various operating systems (such as Solaris, HP-UX, AIX, Digital Unix, IRIX, Linux, Windows NT) and networking technologies (such as ATM, FDDI, SONET, Fast Ethernet, Ethernet, Token Ring) that physically connect the parties involved in a distributed multimedia application.

The heart of our work is the multimedia API of the second layer and the distributed multimedia services of the third layer. We will present more details of those two layers in the following two sections.

3.0 MULTIMEDIA API

In this section, we describe the purpose of defining multimedia API and present the objects that are defined in the multimedia API layer.

3.1 Purpose of Multimedia API

The purpose of defining Multimedia API is as follows:

- To provide a wide range of multimedia-related class definitions so that multimedia applications can deal with multimedia-related problems easily and flexibly.
- To provide a common programming interface so that the developers of multimedia applications can develop new multimedia applications easily.
- To reuse codes, which are common to many multimedia applications.
- To support the processing and presentation of various kinds of media such as video, audio, image, and text so that multimedia applications can deal with various kinds of media easily.
- To reduce software development time.

3.2 Object Classes in Multimedia API

The multimedia API is composed of three object class hierarchies, which are media class hierarchy, component class hierarchy and device class hierarchy. The media class hierarchy models various multimedia data being exchanged in multimedia applications. The component class hierarchy models media producers and consumers, which generate and transmit media from a sender and receive and output the media to the receiver devices. Finally, the device class hierarchy models various devices (such as camera, microphone, monitor display, speaker) involved in multimedia applications.

There are two main classes in the media class hierarchy. The first class is *Media* which is the root class and has common interfaces for all kinds of media type. The second class is *Temporal* which is a subclass of *Media* and has common interfaces for the temporal media types. Here, we define a temporal media type as one that is much affected by time factor when they are presented. The subclasses of *Temporal* include *Audio*, *Video*, *Music* and *Animation* classes. Fig. 2 shows the media class hierarchy.

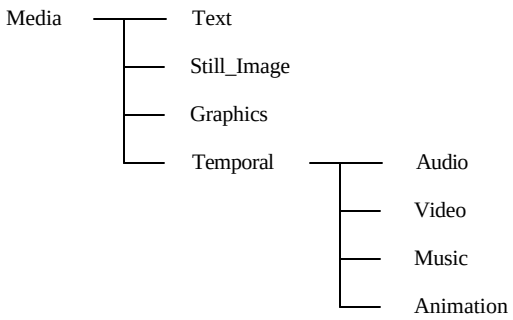


Fig. 2: Media Class Hierarchy

In MAESTRO, multimedia applications use the concept called **component objects** [27] to transmit and receive various kinds of data. Component objects are objects of a special form. That is, they have their own thread of execution.

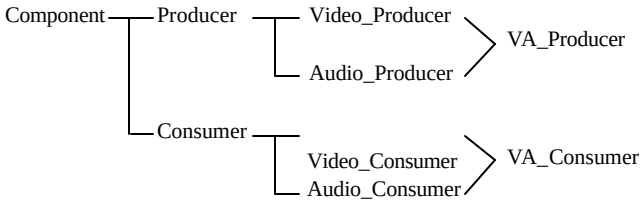


Fig. 3: Component Class Hierarchy

Fig. 3 shows the component class hierarchy. *Component* is the root class` of the component class hierarchy. *Producer* is a subclass of *Component* and models application components that generate data. *Consumer* is also a subclass of *Component* and models application components that consume data generated by *Producer* objects. *Video_Producer* is a subclass of the *Producer*, which is used for generating *Video*. *Audio_Producer* is also a subclass of the *Producer* but it is used for generating *Audio*. Similarly, *Video_Consumer* and *Audio_Consumer* are used to consume *Video* and *Audio*, respectively. The *VA_Producer* inherits both *Video_Producer* and *Audio_Producer* and is used to generate both *Video* and *Audio*. The *VA_Consumer* inherits both *Video_Consumer* and *Audio_Consumer* and is used to consume both *Video* and *Audio*. Video/audio conferencing tools typically use *VA_Producer* and *VA_Consumer* objects for generating video and audio data in the sender and consuming them in the receiver. Although the component class hierarchy (illustrated in Fig. 3) shows the hierarchy for video/audio-related producers and consumer, any type of producers and consumers can be modeled and developed.

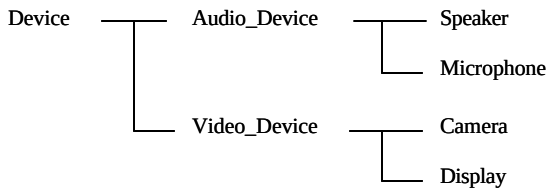


Fig. 4: Device Class Hierarchy

Fig. 4 shows the device class hierarchy. There are three main classes in this class hierarchy. The first class is *Device* that is the root class. It has common interfaces for all multimedia-related devices. The second class, *Audio_Device*, is a subclass of *Device* and has common interfaces for the audio-related devices. The third class, *Video_Device*, is also a subclass of *Device* and has common interfaces for the video-related devices. *Speaker* is a subclass of *Audio_Device*, which is used to control audio-output device. *Microphone* is also a subclass of *Audio_Device*, which is used to control audio-input device. Similarly, *Camera* and *Display* are subclasses of *Video_Device*, which are used to control video-input and video-output devices, respectively.

4.0 DISTRIBUTED MULTIMEDIA SERVICES

In this section, we present MAESTRO distributed multimedia services that are essential in supporting the operation and management of distributed multimedia applications. There are currently five distributed multimedia services defined, which include *Communication Service*, *Session Service*, *Naming Service*, *Storage/Retrieval Service* and *Management Service*. These distributed multimedia services support multimedia applications to transmit or receive multimedia each other, to do a collaborative work in a session, to find a service in a distributed environment, to store/retrieve multimedia, and to monitor and control services and multimedia applications.

Although, we present only five important distributed multimedia services, a new service can be easily defined and incorporated since the services have also been modeled using the object-oriented approach. Now, we describe each distributed multimedia service.

4.1 Communication Service

In this section, we describe Communication Service that is needed by multimedia applications to exchange various kinds of media. The functional requirements of the Communication Service for supporting distributed multimedia communication are as follows:

- It should be possible to exchange various media including temporal and non-temporal media.
- Multi-point communication should be possible to support conferencing, multicasting and collaboration.
- It should control the flow of temporal media and synchronize two or more types of media if necessary.
- It should allow multimedia applications to control the quality of service (QoS).

The Communication Service in MAESTRO has been designed with the functional requirements mentioned above. The unit of data exchange in MAESTRO communication is media object, which can be of any size or type defined using the media class hierarchy. Main features of the communication service include the support of multi-point communication (i.e., one-to-one, one-to-many, many-to-many communication) and intra-domain and inter-domain communication which are essential for supporting distributed multimedia applications. Below, we present the details of the communication service.

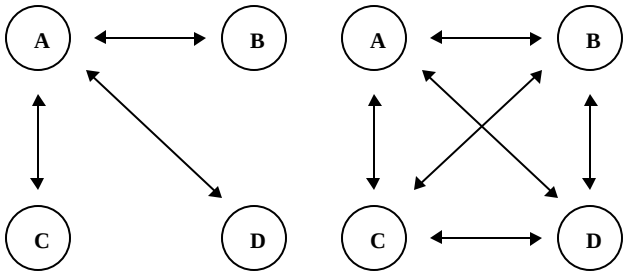
4.1.1 Objects in Communication Service

The objects that are defined and used in the Communication Service are *CommunicationFactory*, *ConnectionManager*, *Port*, *Connector* and *Channel* objects. The role of *CommunicationFactory* is to create, manage or destroy *Port*, *Connector* and *Channel* objects that are used by multimedia applications in a multi-point communication. If a multimedia application wants to communicate, it must first create a *Port* using a *CommunicationFactory*. After creating a *Port*, the multimedia application can communicate with another multimedia application that also has created its own *Port*. A *Port* alone cannot join a multi-point communication because the *Port* cannot know all the information about the created *Ports*, *Connectors* and *Channels*. Thus, a *Port* must join a multi-point communication by passing a connection request to a *ConnectionManager* that has all the information about the created *Ports*, *Connectors* and *Channels*. *Connector* is used when an inter-domain communication is needed. The role of *Connector* will be described in more detail in Section 4.1.3. *Channel* keeps all the references of *Ports* that are joining the *Channel*.

There are two kinds of *Ports*. One is one-to-many *Port*, the other is many-to-many *Port*. When a multimedia application creates a *Port*, it should pass an argument that specifies whether the *Port* should be one-to-many *Port* or many-to-many *Port*.

4.1.2 Multi-point Communication

Multi-point communication (i.e., one-to-many, many-to-many communication) is essential when developing multimedia applications. Thus, the Communication Service in MAESTRO provides multi-point communication. One-to-one communication can be easily achieved by using either one-to-many communication or many-to-many communication.



(a) One-to-many communication (b) Many-to-many communication

Fig. 5: One-to-many and many-to-many communications in MAESTRO

Before describing the multi-point communication in MAESTRO, we define what we mean by one-to-many and many-to-many communication. Fig. 5 (a) illustrates an example of one-to-many communication where component A can send the media to all the other components. The components B, C and D can send the media back to component A only. Fig. 5 (b) illustrates an example of many-to-many communication where any component can send the media to all other components.

In MAESTRO, *Ports* and *Channels* are used for multi-point communication. If a multimedia application component wants to take part in one-to-many or many-to-many communication, it must create a one-to-many or many-to-many *Port* using the *CommunicationFactory*. The multimedia application component must also create a *Channel* using the created *Port*. Then, other *Ports* that have been created by other participating multimedia application components can join the created *Channel* by connecting to the *Port* that has created the *Channel*.

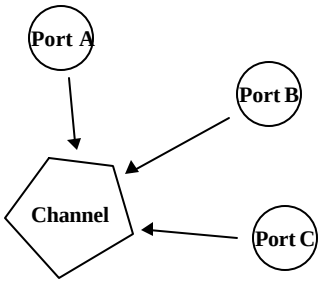


Fig. 6: Ports and Channels in Multi-point communication

Fig. 6 illustrates how a multi-point communication is realized conceptually. In the figure, *Port A* created by a multimedia application component creates a *Channel*. *Port A* automatically joins the *Channel* by creating the *Channel*. Then, when *Port B* and *Port C* connect to *Port A*, they automatically get joined to the *Channel*. Thus, there are two ways for a *Port* can join a *Channel*:

1. by creating a *Channel* itself, or
2. by connecting to a *Port* which has already joined a *Channel*.

When a *Port* joins a *Channel* we can make the connection of *Ports* in many ways. For example, a hierarchical tree or a virtual ring of *Ports* can be constructed to make the connection of *Ports*. In MAESTRO, we selected to construct a hierarchical tree when a *Port* joins a *Channel*. Fig. 7 illustrates how a hierarchical tree of *Ports* is constructed in MAESTRO. First, *Port A* creates a *Channel*, which makes it possible for other *Ports* to connect to *Port A*. So, *Port B* can connect to *Port A*. Similarly, *Port C* and *Port D* can connect to *Port A*. Now, *Port E* can connect to *Port D* because there is a *Channel* that *Port D* has joined. *Port F* can join likewise. Through this hierarchical tree of *Ports*, multi-point communication between multimedia application components using the respective *Ports* is possible.

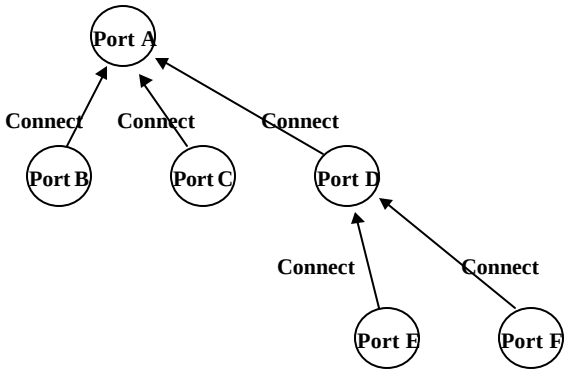


Fig. 7: Hierarchical Tree of Ports in Multi-point communication

The construction process of hierarchical tree is the same in both one-to-many communication and many-to-many communication. The difference lies in the way media flows in the tree. We now describe the difference using Fig. 7.

We describe the case of one-to-many communication first. If the root *Port* sends a media object, all the other *Ports* receive it. For example, if a multimedia application component sends a media object to *Port A*, *Port A* sends the media object to its children *Ports B*, *C* and *D*. *Port D* then sends the media object to its children *Ports E* and *F*. If any *Port* except the root *Port* sends a media object, only the root *Port* receives that media object. For example, if a multimedia application component sends a media object to *Port F*, *Port F* sends the media object to its parent, *Port D*, but *Port D* just passes the media object only to its parent, *Port A*.

In the case of many-to-many communication, any *Port* can send a media object to all the other *Ports*. For example, if a multimedia application component sends a media object to *Port A*, the media object flows same as in one-to-many communication. If a multimedia application component sends a media object to *Port F*, then *Port F* sends the media object to its parent *Port D*. And *Port D* sends the media object to its child, *Port E* and parent, *Port A*. Also, *Port A* then subsequently send he media object to its children *Ports B* and *C*.

4.1.3 Intra-domain and Inter-domain Communication

We define a domain as an area in which distributed multimedia services are provided to multimedia applications. In MAESTRO, we have used the concept of domain for the ease of administration as well as for the purpose of scalability. There can be no doubt that multimedia applications should be able to communicate each other in a same domain. But it should be also possible for multimedia applications in different domains to communicate each other.

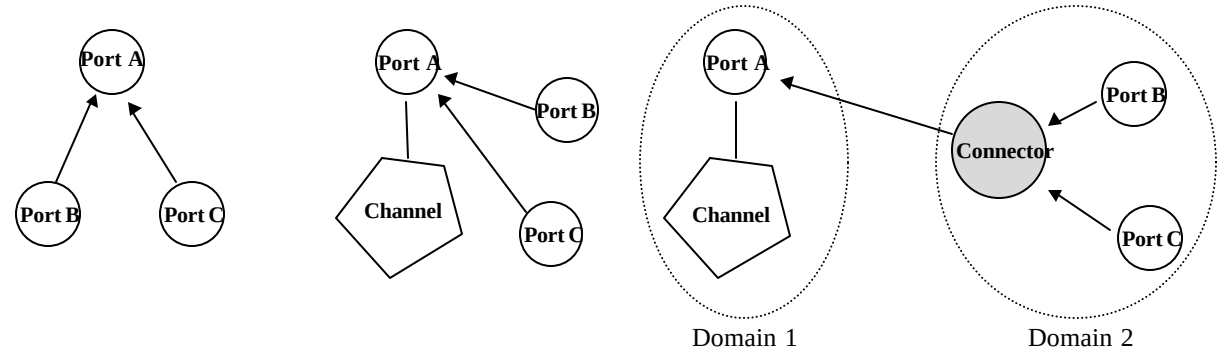
The general concept of intra-domain communication and inter-domain communication is illustrated in Fig. 8. Multimedia application components A and B, which use the distributed multimedia services in the same domain, are participating in an intra-domain communication. Multimedia applications components A and C, which use the distributed multimedia services in two different domains are participating in an inter-domain communication.

Now, we will illustrate the difference of intra-domain and inter-domain communications in MAESTRO. Suppose a hierarchical tree has been constructed as in Fig. 9 (a). If all the *Ports (A, B and C)* are located in a local domain, then the connection of the *Ports* is the same as what we described in Section 4.1.2 using Fig. 6. This is shown in Fig. 9 (b). However, if *Port A* is located in a local domain, and *Ports B* and *C* are located in a remote domain, a *Connector* in the remote domain is needed to connect the *Ports* as shown in Fig. 9 (c).

Although we can construct a hierarchical tree without a *Connector,* the reason why we use the *Connector* in an inter-domain communication is for transmitting the media object more efficiently across domains. That is, if a media object should be transmitted across different domains, a single copy of the media object is transmitted to the *Connector* only rather than transmitting multiple copies to all the destination *Ports* in a remote domain. The *Connector* would then transmit the media object locally to appropriate *Ports*.

4.1.4 Connection Establishment

Now, we describe the detailed process of connection establishment between *Ports*. First, we describe an intra-domain connection where two *Ports* exist in a single domain. Then, we describe an inter-domain connection where two *Ports* exist in different domains. An intra-domain connection is illustrated in Fig. 10 (a).



(a) Example connection (b) Intra-domain connection (c) Inter-domain connection

Fig. 9: Intra-domain and Inter-domain connections

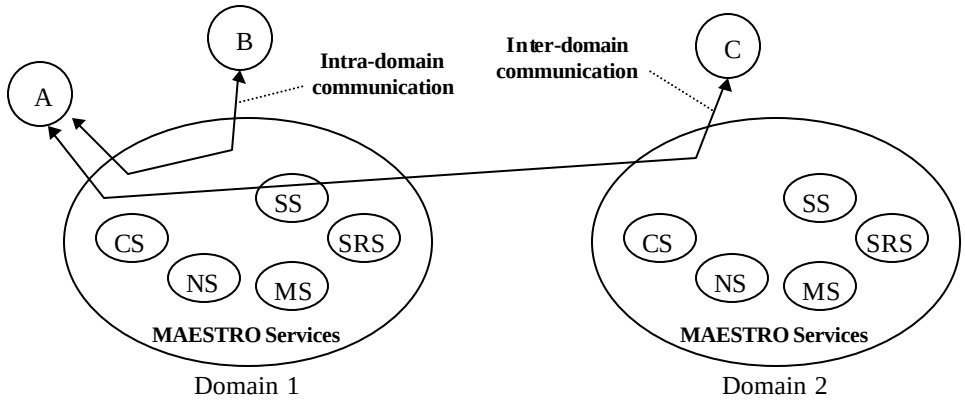
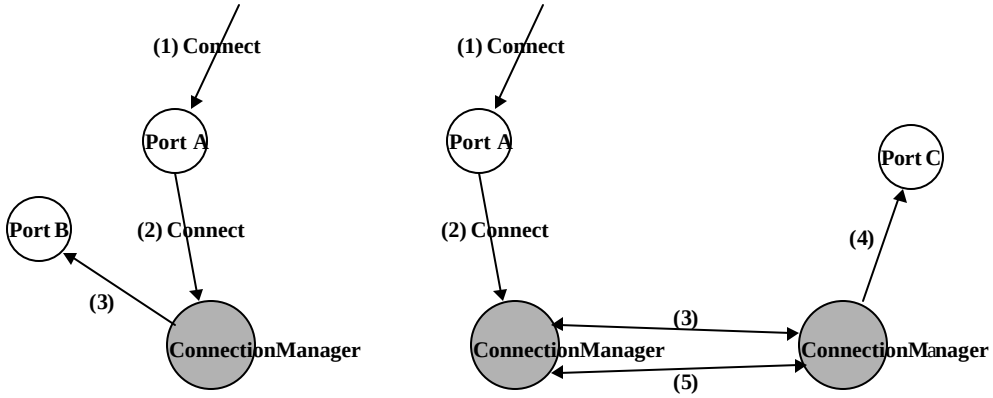


Fig. 8: Intra-domain and Inter-domain communication in MAESTRO



(a) Intra-domain connection (b) Inter-domain connection

Fig. 10: The process of connection establishment

(1) A multimedia application component issues a connection request. When issuing a connection request, the multimedia application component passes a domain name and port number which identify a unique destination *Port*. Note that a unique port number is given to a *Port* when it is created.
(2) *Port A* passes the connection request to *ConnectionManager*.
(3) *ConnectionManager* finds the destination *Port* in a local domain and makes a tree relationship between the two *Ports*.

An inter-domain connection is illustrated in Fig. 10 (b).

(1) A multimedia application component issues a connection request.
(2) *Port A* passes the connection request to *ConnectionManager*.
(3) After discovering that the destination *Port* exists in a remote domain, *ConnectionManager* communicates with the appropriate remote *ConnectionManager* to carry out the connection request.
(4) Remote *ConnectionManager* finds the destination *Port*.
(5) The local *ConnectionManager* and remote *ConnectionManager* make a tree relationship between the two *Ports* by passing the needed information to each other.

We have presented the details of the most important multimedia service, communication service. In the remainder of the section, we describe the rest of distributed multimedia services that comprise MAESTRO services.

4.2 Session Service

The second multimedia service provided in MAESTRO is the Session Service that keeps track of currently active sessions in MAESTRO. A session is a live activity in which one or more users can interact using multimedia applications. In MAESTRO, multiple sessions can exist simultaneously, involving users from different domains.

Essential functional requirements of the Session Service are as follows. The Session Service in MAESTRO has been designed with these functional requirements.

- A user should be able to create, join, leave or destroy a session.
- A user should be able to discover the users currently participating in a session
- There should be a chairman (i.e., a super user) in a session such that he should have all the rights in a session.
- If needed, the chairman should be able to control the right to speech.
- A user should be able to join and leave a session that is created not only in a local domain but also in a remote domain.
- If needed, the chairman should be able to forcefully terminate sessions or eliminate participants from a session.

4.2.1 Objects in Session Service

The objects that comprise the Session Service are Session Service Object (SSO), Session, User, Application and ApplicationDB objects. The role of SSO is to create, destroy and search a Session object that has the information of a session. A User object has the information of a user and an Application object has the information of an application.

Fig. 11 illustrates how the objects including SSO, Session, User and Application are managed in the Session Service. SSO manages Sessions and Session manages Users and User manages Applications. Thus, one can discover which sessions have been created by SSO and which users have joined a session. One can also discover which users are running which applications.

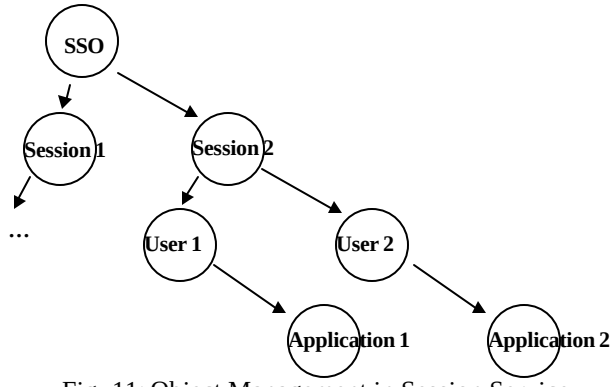


Fig. 11: Object Management in Session Service

Finally, ApplicationDB has the information about what kind of multimedia applications can be used with the Session Service. If any multimedia applications should be integrated with the Session Service, an authorized user should register the name and the location of those multimedia applications first before users can use the applications.

4.3 Naming Service

The objects that comprise distributed multimedia services are distributed in a network. In such an environment, multimedia applications should be able to find the objects that serve the wanted services. The role of the Naming Service is to allow a name to be bound to an object and to allow the object to be found subsequently by resolving that name within the Naming Service. Thus, if an object wants to provide a service, it should register itself with the Naming Service first, giving it a name that can be used by multimedia applications to find the object.

The Naming Service provides the following functions:

- Binding a name to an object reference.
- Resolving a name to find an object reference.
- Unbinding a name to remove a binding.
- Listing the names.

There is a Naming Service specification that has been defined by OMG (Object Management Group). The Naming Service specification is one of the many service specifications of the CORBAservices [16]. The CORBAservices is a standard document from OMG, which extend the core CORBA specification with a set of optional services that are useful in many applications. In our current version of MAESTRO, we are using the naming service specification defined by OMG.

4.4 Storage/Retrieval Service

In the multimedia API layer, various kinds of media objects have been modeled. The Storage/Retrieval Service allows these objects to be stored into or retrieved from a common repository. The Storage/Retrieval Service provides the following basic functions:

- Storing a media object.
- Retrieving a media object.
- Searching a media object.

When storing a media object, the multimedia application should pass a name with the media object. This name can be used when a multimedia application wants to retrieve the media object that the name identifies. Also, multimedia applications can discover what kinds of media objects are stored currently by using the searching function.

4.5 Management Service

In this section, we describe the Management Service that is required to monitor and control MAESTRO services and applications. The Management Service makes MAESTRO services more efficient and reliable so that MAESTRO applications can perform their tasks without problems.

The functional requirements of the Management Service are as follows:

- Various information about MAESTRO services and applications should be provided.
- Relationship information between MAESTRO services and applications should be provided.
- Events and faults which may occur in MAESTRO services and applications should be reported.
- Performance level of MAESTRO services should be measured.
- Users' list and access control list should be managed.

The Management Service in MAESTRO has been designed based on the functional requirements mentioned above. We now describe the details of the Management Service.

4.5.1 Management Service Architecture

The objects that are defined and used in the Management Service Object (MSO) are configuration management service object (cMSO), fault management service object (fMSO), security management service object (sMSO) and performance management service object (pMSO). Each object is in charge of each management service category. In addition, a management interface object (MIO), which is instrumented in each service object, is used to make service objects manageable. Fig. 12 shows the architecture of the Management Service.

MIO is an object that is instrumented in every managed service object (SO) so that SOs can be managed by MSO [18]. MIO is defined as a set of management operations and management information [19] needed to manage SOs. Specific MIOs can be developed by extending the generic MIO using the inheritance feature of the object-oriented technique.

MSOs deal with management requests from management applications (MA). A request may involve one, two or three MSOs. In some cases, a request may require services

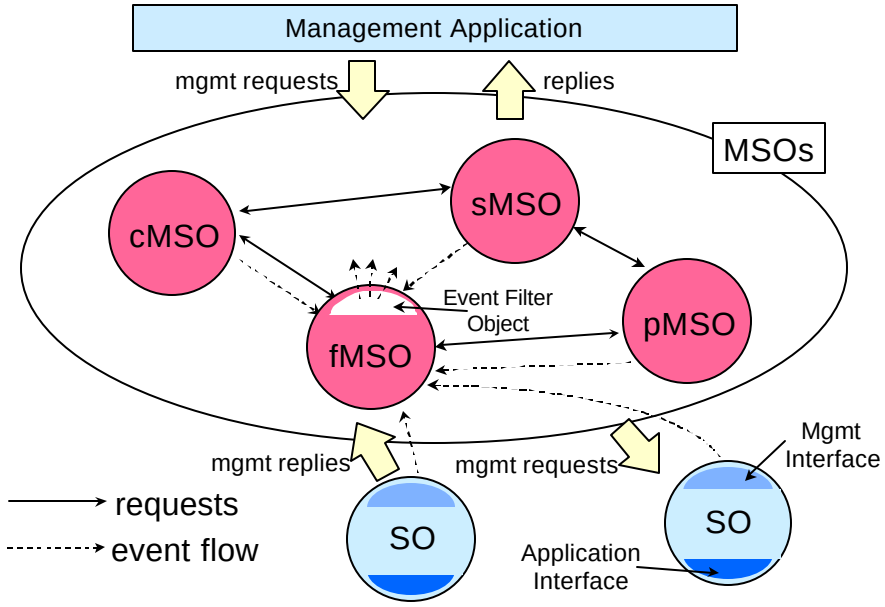


Fig. 12: Architecture of Management Service

provided to the MIO of SO indirectly. Such request is transferred through MSOs. After processing the requests, replies are sent to MA. The replies may be management information of an SO, configuration information between SOs, or some events occurred.

In addition, MSOs interact each other. For example, fMSO may need configuration information of managed SOs to detect faults. In this case, fMSO uses services provided by cMSO. Also, cMSO may request sMSO to provide an access control list (ACL) for a SO when it is requested to get some MIB information of a SO by MA.

fMSO contains an object for forwarding events called "event channel" and objects for filtering events, called "event filter". All events flow into the event channel and then the event channel forwards the events to objects that are supposed to receive them. Upon receiving the events from fMSO, MA will perform appropriate actions such as changing the color of the corresponding object or displaying the message associated with the event.

In addition, since we are interested in managing multimedia services that are distributed in a possibly large internetwork environment, we may require more than one set of MSOs. MSOs may need to communicate with each other to support distributed management of the distributed multimedia services. Below, we describe major component services that comprise the Management Service in MAESTRO.

4.5.2 Configuration Management Service

The configuration management service is concerned with configuration initialization, maintenance and shutdown of service objects (SOs) and applications within a MAESTRO domain. While the SOs are in operation, the configuration management service is responsible for monitoring the configuration and making changes in response to user requests. The detailed functions of configuration management are as follows:

- Finding and maintaining a list of SOs to be managed.
- Providing information of SOs.
- Monitoring SO's status and notifying problems.
- Initializing and terminating the operation of an SO.

4.5.3 Fault Management Service

The fault management service is responsible for providing a reliable service environment by handling faults gracefully when they do occur in the SOs of a management domain. The fault management service consists of two areas: the underlying event transmission mechanism and fault analysis. Event transmission mechanism is designed by adopting event service specification [17] of OMG CORBA. The detailed functions of fault management are as follows:

- Providing underlying event transmission mechanism.
- Filtering and logging reported events.
- Detecting faults by analyzing reported events and other information.
- Providing some fault diagnostic capabilities.

4.5.4 Performance Management Service

The performance management service is responsible for providing an efficient multimedia service environment. To achieve this, the behavior of SOs must be monitored closely and appropriate performance data must be collected and analyzed. The detailed functions of performance

management are as follows:

- Measuring performance data, e.g., response time, throughput, average number of requests, operation errors and bytes.
- Providing thresholds which can be assigned.
- Notifying cases in which a performance level grows worse, e.g., when a performance metric exceeds its threshold.
- Providing a history of performance data.

4.5.5 Security Management Service

The security management is responsible for providing a secure environment for the operation and management of the SOs. The service is composed of two functional areas, namely, authentication and authorization. The detailed functions of security management are as follows:

- Adding a user to authorized users' list.
- Deleting a user from authorized users' list.
- Allowing a user to log in a management system by passing a valid password.
- Allowing a user to log out.
- Providing ACL (access control list) against an MAESTRO service.
- Allowing ACL to be changed.

5.0 MAESTRO IMPLEMENTATION

Based on the MAESTRO design architecture, multimedia API and distributed multimedia services presented in the previous sections, we have carried out a prototype implementation of the MAESTRO system. This section presents the details of our implementation.

First, MAESTRO multimedia API has been implemented using Java [29] and C++ programming languages. This allows the development of Java-based applications as well as more traditional X Window/Motif GUI-based applications. Java-based applications can run on any Java-enabled Web browsers and thus have the advantage of being able to run on many Unix and PC platforms.

Second, MAESTRO distributed multimedia services (in the form of distributed cooperative servers) have been implemented using CORBA and C++ on the Sun Solaris platform. The use of CORBA allows the multimedia services to be defined using CORBA IDL and the body of services to be implemented in a number of programming languages but we chose C++ as the programming language. Thus, MAESTRO distributed multimedia services have been implemented as CORBA objects interacting with each other within a domain or between domains to support the operation of distributed multimedia applications.

There are currently several freely available CORBA implementations as well as a number of commercially available implementations. We chose to use IONA's CORBA implementations (OrbixWeb 2.0.1 [20] and Orbix 2.2 [21]) that are commercial but known to be the most stable among the ones currently available. IONA OrbixWeb 2.0.1 allows Java-based applications to access MAESTRO multimedia services implemented as CORBA objects and Orbix 2.2 allows C/C++ based applications to access the services.

Using the MAESTRO multimedia API and distributed multimedia services, we developed a number of multimedia applications such as video/audio conferencing tool, video/audio multicasting tool, whiteboard, electronic notebook, chatting tool and session management tool. Most of these applications have been developed using Java and thus they can run on any platform that supports Java Virtual Environment [30]. For example, the multimedia applications that we developed can be run on the Web because Web browsers such as Netscape and Microsoft Explorer supports Java Virtual Environment. The multimedia applications that we have developed will be explained in more detail in the following section.

The hardware we have used are Sun Sparc 20 (running on Solaris 2.4) and Sun Ultra 1 (running Solaris 2.5). We have used SunVideo Card [22] and Solaris XIL Library [23] for video/audio compression and decompression used in video/audio conferencing and multicasting tools.

6.0 MULTIMEDIA APPLICATIONS ON MAESTRO

In this section, we first briefly introduce the "Distant Cooperative Research Experiments" project that is funded by the Korean Information Infrastructure (KII) initiative [1]. The use of multimedia applications is essential in carrying out this project and thus a number of multimedia applications have been developed and operating for this project using MAESTRO. The developed multimedia applications include video/audio conferencing tool, video/audio multicasting tool, whiteboard, electronic notebook, chatting tool and session tool, whose details are then explained.

6.1 Distant Cooperative Research Project

The Korean Information Infrastructure (KII) initiative is funded by the Ministry of Information and Communication, South Korea, for setting up high-speed, broadband networks throughout the peninsula of Korea, connecting various government organizations, research institutions and universities. KII has been funding various research projects for developing systems and applications that will utilize the information superhighway being set up. One of the KII initiative projects is "Distant Cooperative Research Experiments" where researchers in the fields of science and engineering perform various research experiments using facilities possibly located in remote sites and sharing the results. The motivation behind this is to encourage cooperative research work among co-workers without

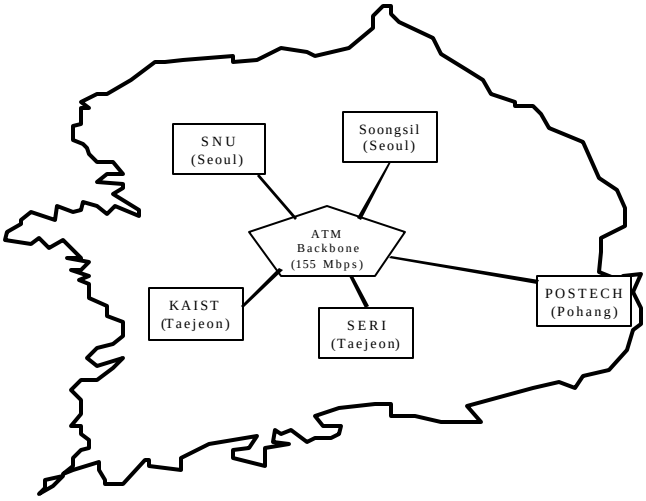


Fig. 13: Underlying 155Mbps ATM Backbone Network connecting Distant Researchers

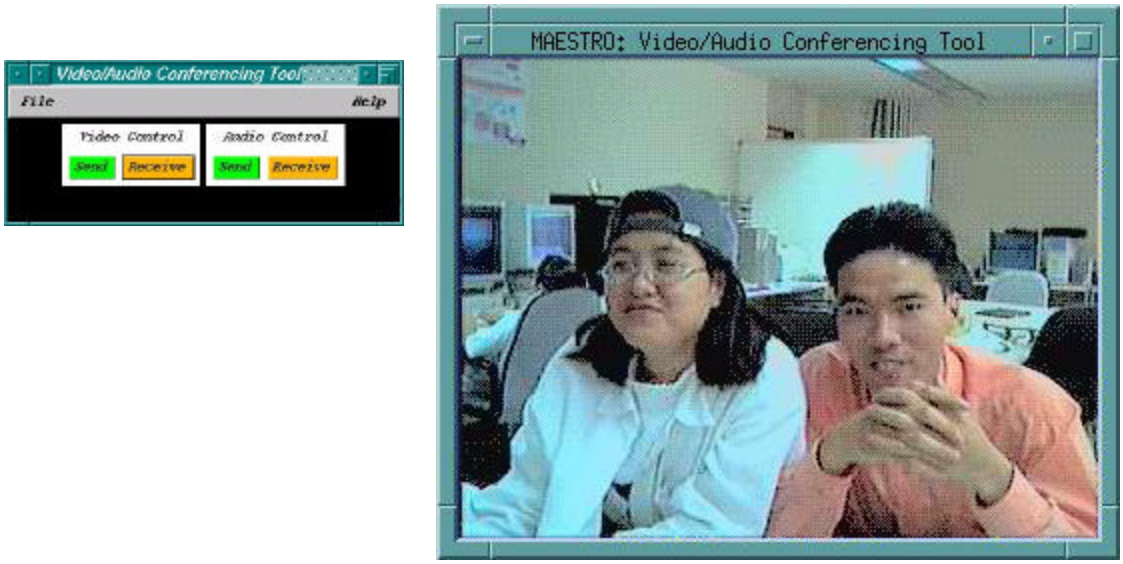


Fig. 14: Video/Audio Conferencing Tool

travelling to a site where expensive research facilities are located and at the same time reduce the time for traveling and the cost. Such kind of cooperative research is possible through the use of multimedia applications such as video/audio conferencing, whiteboard, electronic notebook, remote experimental apparatus monitoring, and information archival and retrieval.

MAESTRO has been chosen to be a multimedia system for developing applications required to support the project and to support the operation and management of those applications on the ATM-based high-speed, broadband network. Fig. 13 illustrates the underlying 155-Mbps ATM backbone network connecting four universities and one research institute spread apart by hundreds of kilometers in South Korea. Two universities (SNU and Soongsil) are located in Seoul, one university (KAIST) and one research institute (SERI) are located in Taejeon which is about 150 kilometers from Seoul, and our university (POSTECH) is located in Pohang which is about 200 kilometers from Taejeon and 350 kilometers from Seoul. Researchers at these institutions interact each other with a number of multimedia applications running on the ATM backbone network. The remainder of the section describes the multimedia applications we have developed using MAESTRO for this project.

6.2 Video/Audio Conferencing Tool

Using a video/audio conferencing tool, researchers in this project can interact each other in real-time without travelling to remote sites. Fig. 14 shows the video/audio

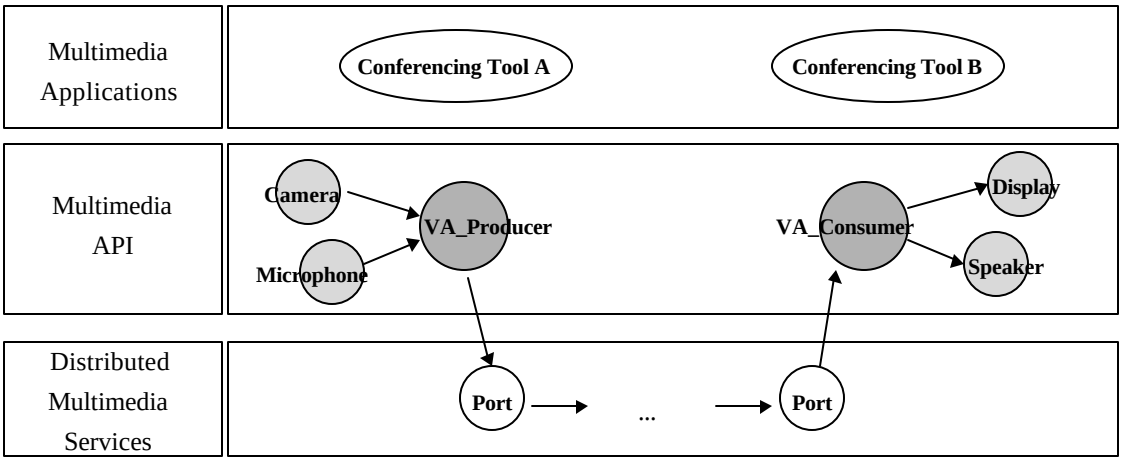


Fig. 15: Objects in Video/Audio Conferencing Tool

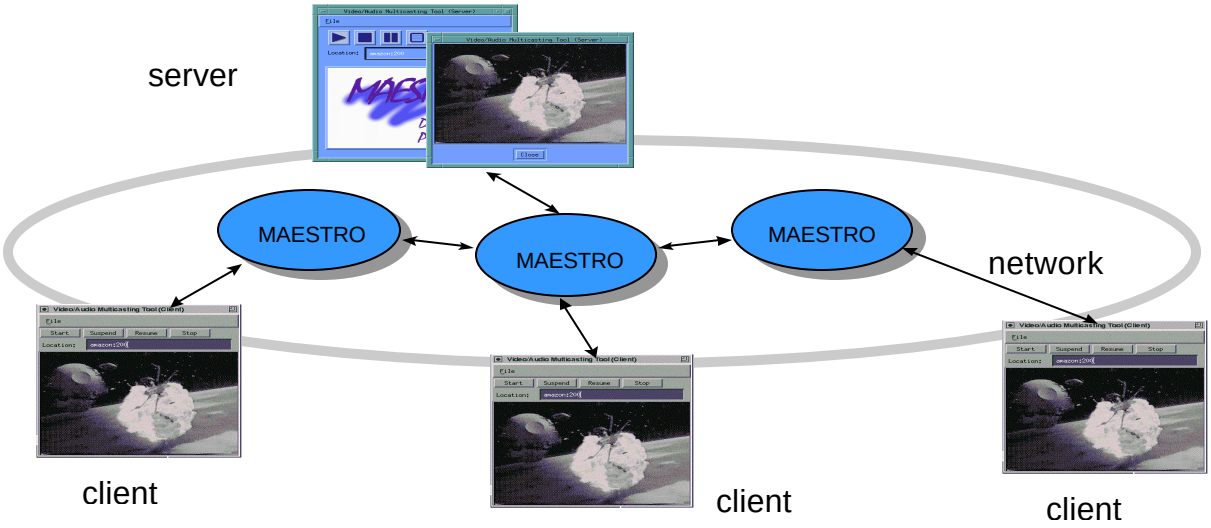


Fig. 16: Video/Audio Multicasting Tool

conferencing tool developed using and operating on MAESTRO. The tool has been implemented using C++ and X Window/Motif library for user interface and Sun Video/Audio Card for capture of video and audio from the source. Users can control the transmission and reception of video and audio data separately using the buttons shown in the left figure. This is a desirable capability so that when no voice is to be heard, one can shut it off and send/receive the video data only. Further, when audio conferencing is only required, then video signals can be turned off, thus effectively using the tool as an audio conferencing tool.

The video/audio conferencing tool has been implemented using the objects in multimedia API and one-to-many communication service. Fig. 15 shows how objects in the layers of multimedia API and distributed multimedia services have been used to implement the video/audio conferencing tool. A video/audio conferencing session is achieved by the use of *VA_Producer* and *VA_Consumer* objects in the multimedia API layer (described in Section 3.2) and *Port* objects in the multimedia services layer (described in Section 4.1).

Conferencing tool A uses a *VA_Producer* object which is an instance of the *VA_Producer* class in the multimedia API layer. *VA_Producer* has inherited features of the *Video_Producer* and *Audio_Producer* classes. *VA_Producer* generates video and audio data by capturing video data using a camera and audio data using a microphone. The captured data are represented as *Video* and *Audio* media objects and are transmitted to the receiver *Port* objects. Conferencing tool B, in turn, uses a *VA_Consumer* object which is an instance of the *VA_Consumer* class. *VA_Consumer* has inherited features of the *Video_Consumer* and *Audio_Consumer* classes. *VA_Consumer* object uses a display (such as monitor) and speaker to output the received media objects. Transmission of video and audio data in the reverse direction (i.e., tool B to tool A) is achieved using the same objects only in the reverse direction.

6.3 Video/Audio Multicasting Tool

The video/audio multicasting tool can be used for broadcasting or showing a video tape or equivalent video archive to a participating group of researchers. The tool consists of server and client applications. The server application controls the initiation and termination of multicasting and the client application is used by each of the participating users to receive the multicasting. Like the video/audio conferencing tool, this tool has also been implemented using C++ and X Window/Motif and Sun Video/Audio Card. Fig. 16 illustrates the video/audio multicasting tool.

6.4 Whiteboard & Chatting Tool

Other multimedia applications that have been developed for the "Distant Cooperative Research Project" are whiteboard and chatting tool. Researchers may use the whiteboard application to aid the discussion with others in real-time by drawing figures and adding texts. This simulates an interactive group discussion using a black or white board in a meeting room. Figures and texts entered by a user are displayed instantaneously to the whiteboards of other participants. Fig. 17 illustrates the drawing control and display windows of the whiteboard tool we have developed. The tool has been developed using the Java language so that any Java-enabled Web browser on any platform can be used to execute the tool.

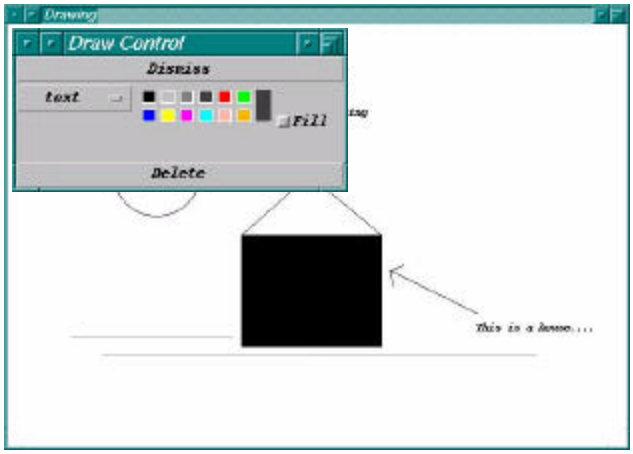


Fig. 17: Whiteboard drawing control and display

The Chatting tool shown in Fig. 18 can be used by a number of researchers when they need to have interactive multi-user discussions or when they wish to simply talk by exchanging texts. This tool may be used in conjunction with video/audio conferencing and whiteboard tools or separately, especially when video/audio conferencing is not available due to problems with camera and/or speakers or due to lack of bandwidth available to support video/audio conferencing. This tool has also been developed using the Java language so that any Java-enabled Web browser on any platform can be used to execute the tool.

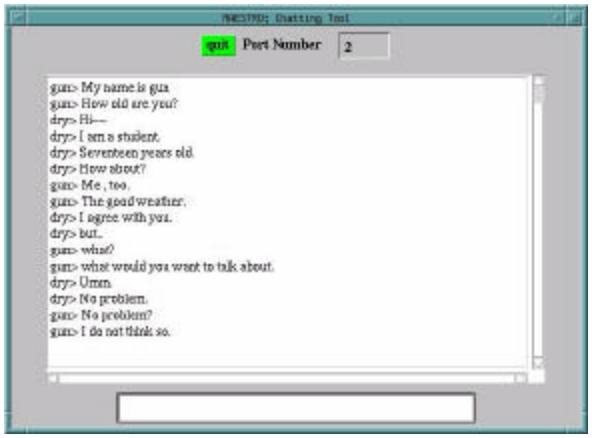


Fig. 18: Chatting Tool

6.5 Electronic Notebook & Session Management Tool

Electronic notebook can be thought as a normal paper notebook that is made with the help of computer. Thus, we do write on it using a pen by hand but by keyboard and mouse through a graphic user interface. The electronic notebook tool that we have developed can be used by a group of researchers for taking notes during a distant experiment. It can be also used to retrieve the old notes taken during a previous experiment. The tool is capable of inserting graphical images as well as texts, as shown in Fig. 19. This tool also has been implemented using Java and can be executed on any Java-enabled Web browser.

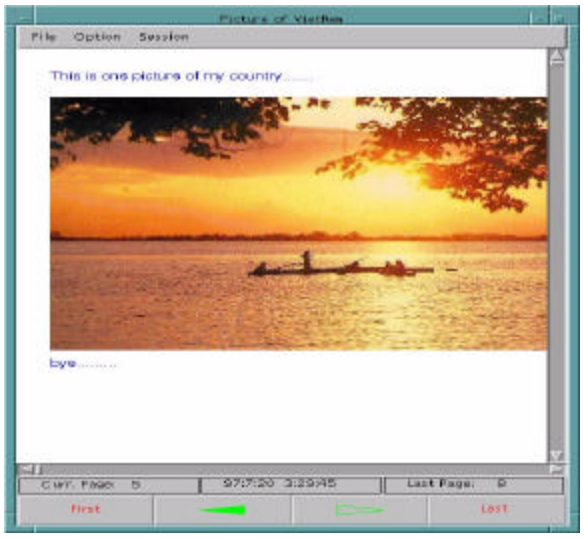


Fig. 19: Electronic Notebook

Fig. 20 illustrates the user interface of the session management tool that is used to manage and control the sessions in MAESTRO as well as the information on the participants of a session. Using this tool, a user can discover what sessions have been opened in a domain, which users are currently participating in a session, and which applications are being used by which users. It is also

possible to do a collaborative work using the session management tool because all the multimedia applications introduced above are integrated with it. That is, using this tool, one can run a video/audio conferencing tool automatically connecting to other video/audio conferencing tool and run a whiteboard automatically sharing the information of whiteboard with other user's whiteboard, and so on. This tools also has been implemented using Java and can be executed on any Java-enabled Web browser.

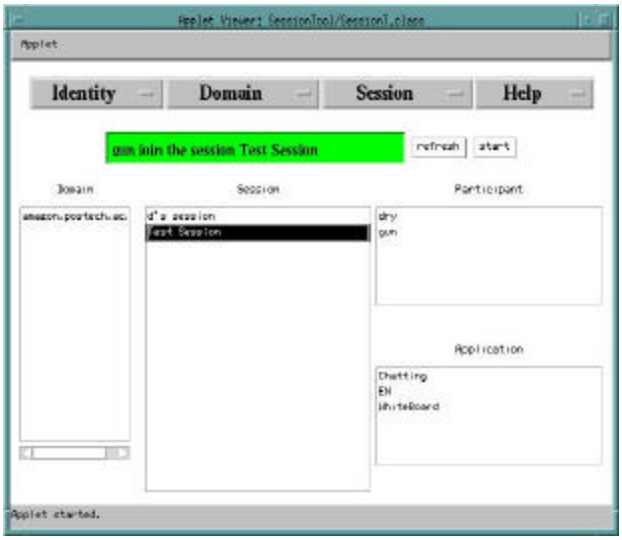


Fig. 20: Session Management Tool

7.0 CONCLUDING REMARKS

This paper discusses the motivation behind our work on developing a distributed multimedia system called MAESTRO, which can be used to support a variety of multimedia applications on multimedia information superhighways. We presented its architecture, multimedia API and distributed multimedia services. Using this system, distributed multimedia applications can be easily developed and operated efficiently and reliably.

As a proof of concept, we have developed a number of multimedia applications that are required for supporting cooperative research experiments among distant users on the Korean national information superhighway. Using such multimedia applications, researchers can perform experiments using research facilities located in remote institutions, collect and analyze data, and interact with remote colleagues in real-time. This not only can save a lot of money by sharing expensive experimental facilities but also can save a lot of time by not traveling to remote sites. We believe that this type of collaboration will be exercised more commonly in the future and multimedia systems such as MAESTRO can play a key role in making it possible.

We are currently developing other multimedia applications (such as telemedicine, telepublishing and telecommerce) using MAESTRO. We plan to evaluate our system with an increase in the number of users (in the order of thousands or more), to truly evaluate the scalability of MAESTRO and to fine-tune the system for even better performance. We also plan to include international sites running multimedia applications on MAESTRO so that we can extend the underlying network infrastructure from the national information superhighway to an international information superhighway.

Another work in progress deals with Quality of Service (QoS) in MAESTRO. Most of national and international networks being used today in various organizations are not high-speed, broadband networks and thus do not satisfy all the requirements of multimedia applications. What is needed is a way to adapt flexibly to the quality of underlying networks so that these applications can run on most of the networks that exist today.

8.0 ACKNOWLEDGEMENTS

The work presented in this paper has been funded in part by 1995 Special Fund for University Research Institute, Korea Research Foundation and by 1997 Research Fund for the Cooperative Research & Experimental System Project, Systems Engineering Research Institute (SERI). The authors would like to thank the MAESTRO project team members, Young-Hwan Lee, Yong-Woo Shin, Chang-Won Kwak, Binhminh Do and Sang-Eun Park who spent many sleepless nights in developing multimedia applications on MAESTRO.

REFERENCES

[1] Korea Information Infrastructure Project, *http://www.nca.or.kr/NCA97e/index97.html*.

[2] The ACTS ATM Internetwork, *http://www.arl.mil /HPCMP/DREN/testbed/aai.html*.

[3] Singapore Information Technology (IT2000), *http://www.ncb.gov.sg/*.

[4] CANARIE Information Highway, *http://www.canarie.ca:80 /eng/main.html*.

[5] National Information Infrastructure, *http://nii.nist.gov /nii/niiinfo.html*.

[6] Global Information Infrastructure, *http://www.gii.org/*.

[7] Barry K. Aldred, "IBM Lakes Architecture: Introduction and Programmers' Guide", IBM UK Laboratories Ltd., *http://www.hursley.ibm.com /~p2p/*, 1993.

[8] Interactive Multimedia Association, "IMA Recommended Practice Draft", *http://www.ima.org /forums/imf/mss/*, September 1994.

[9] P. Dubois, "Detailed Specification of the BETEUS application platform", BETEUS Consortium, *http://www.tik.ee.ethz.ch/ ~beteus/*, November 1994.

[10] *Mux Tutorial*, Electronics and Telecommunications Research Institute (ETRI), 1995.

[11] M. Arango, et al, "The Touring Machine System", *Communications of the ACM*, 36(1), January 1993, pp. 68-77.

[12] T. H. Yun, J. Y. Kong and J. W. Hong, "Object-oriented Modeling of Distributed Multimedia Services", *Proc. of IEEE International Conference on Communications, Montreal, Canada*, June 1997, pp. 777-781.

[13] T. H. Yun, J. Y. Kong and J. W. Hong, "A CORBA-based Distributed Multimedia System", *Proc. of 1997 Pacific Workshop on Distributed Multimedia Systems, Vancouver, Canada*, July 1997, pp. 1-8.

[14] T. H. Yun, J. Y. Kong and J. W. Hong, MAESTRO: A CORBA-based Distributed Multimedia System, *Technical Report, PIRL-TR-97-2, POSTECH,* March 1997.

[15] OMG, *The Common Object Request Broker: Architecture and Specification Revision 2.0*, OMG, July 1995.

[16] OMG, *"CORBA services: Common Object Services Specification"*, OMG Document Number 95-3-31, March 1995.

[17] OMG, CORBAservices: EventService Specification, *http://www.omg.org/corba/sectrans.htm*, March 1995.

[18] J. W. Hong, G. Gee, and M. A. Bauer, "Towards automating instrumentation of systems and applications for management", *Proc. of the IEEE Globecom, Singapore*, November 1995, pp. 107-111.

[19] Ji-Young Kong and J. Won-Ki Hong, "A CORBA-based Management Framework for Distributed Multimedia Services and Applications", *Technical Report PIRL-TR-97-1, POSTECH*, March 1997.

[20] IONA, *OrbixWeb 2.0.1*, IONA Technologies Ltd., Release 2.0.1.

[21] IONA, *Orbix 2*, IONA Technologies Ltd., Release 2.0.

[22] *Sun Video User's Guide*, Sun Microsystems, August 1994.

[23] *Solaris XIL 1.1 Imaging Library Programmer's Guide*, Sun Microsystems, November 1993.

[24] H. Lutfiyya G. S. Perrow, J. W. Hong and M. A. Bauer, "The Abstraction and Modeling of Management Agents", *Proc. of the Fourth International Symposium on Integrated Network Management, Santa Barbara CA*, May 1995, pp. 466-478.

[25] J. Won-Ki Hong. "Distributed Systems Management Technology", *KISS Review* 14(1), January 1996, pp. 51-61.

[26] OMG, "Control and Management of A/V Streams Request For Proposal", August 1996.

[27] S. J.Gibbs and D. C. Tsichritzis, *Multimedia Programming, Objects, Environments and Frameworks*, Addison-Wesley, 1995.

[28] Microsoft, "DCOM: A Business Overview", *http://www.microsoft.com/ntserver/info/dcom.htm*, August 1997.

[29] E. R. Harold, *Java Network Programming*, O'Reilly & Associates, 1997.

[30] *The Java Virtual Machine Specification*, Sun Microsystems, *http://java.sun.com/docs/books/ vmspec/index.html*, 1997.

BIOGRAPHY

James Won-Ki Hong is an assistant professor in the Dept. of Computer Science and Engineering, POSTECH, Pohang, Korea. He has been with POSTECH since May 1995. Prior to joining POSTECH, he was a research professor in the Dept. of Computer Science, University of Western Ontario, London, Canada, where he worked on the CORDS project and MANDAS project. Dr. Hong received the BSc and MSc degrees from the University of Western Ontario in 1983 and 1985, respectively, and the PhD degree from the University of Waterloo, Waterloo, Canada in 1991. His research interests include network and systems management, distributed computing and multimedia systems. He is a member of IEEE, ACM, KNOM and KISS.

Tae-Hyoung Yun received the BS degree in Computer Science and Engineering from POSTECH, Korea in 1996. He is currently a Masters degree candidate in the Dept. of Computer Science and Engineering, POSTECH. His research interests include distributed computing and multimedia systems. He is a member of IEEE and KISS.

Ji-Young Kong received the BS degree in Computer Science and Engineering from POSTECH, Korea in 1996. She is currently a Masters degree candidate in the Dept. of Computer Science and Engineering, POSTECH. Her research interests include network and systems management and distributed computing. She is a member of KISS.

Young-Mi Shin received the BS degree in Computer Science and Engineering from POSTECH, Korea in 1997. She is currently a Masters degree candidate in the Dept. of Computer Science and Engineering, POSTECH. Her research interests include distributed multimedia systems and network and systems management. She is a member of KISS.