An Internet Based Multimedia Infrastructure for Collaborative Engineering

By

Padmanabha N Vedam

Bachelor of Technology, Indian Institute of Technology, Madras, India

ENG

Submitted to the Department of Civil and Environmental Engineering in partial fulfillment of the requirements for the degree of

Master of Science

at the

Massachusetts Institute of Technology

May 17, 2000

Author.....

Department of Civil and Environmental Engineering

May 05, 2000

Certified By.....

Feniosky Peña-Mora

Associate Professor of Civil and Environmental Engineering

Thesis Supervisor

Accepted By.....

Daniele Veneziano

Professor and Chairman, Departmental Committee on Graduate Studies

An Internet Based Multimedia Infrastructure for Collaborative Engineering

By

Padmanabha N. Vedam

Submitted to the
Department of Civil and Environmental Engineering

May 17, 2000

In Partial Fulfillment of the Requirements for the Degree of
Master of Science

Abstract

The evolution of computer based collaborative environments has resulted in easier and more economical design efforts among geographically distributed design teams. Most of today's internet based collaborative applications allow people that are geographically dispersed to meet with each other using their computers and work together without actually having to travel. A prototype system was developed by taking two tactical planning applications and incorporating them into the collaboration model employed by CAIRO (Collaborative Agent Interaction control and synchROnization). This system was developed based on the collaboration infrastructure that was developed as a part of the Da-Vinci Society Initiative at MIT. The main focus of this research lies in the formalization of a multi-media based architecture that supplements the existing collaboration infrastructure. This architecture lays the groundwork for development of a robust collaboration system that incorporates audio/video conferencing, speech recognition and synthesis and three-dimensional virtual meeting environments in order to facilitate efficient collaboration.

Thesis Supervisor: Feniosky Pena-Mora
Title: Associate Professor of Civil and Environmental Engineering

ACKNOWLEDGMENT

May 17, 2000

First and foremost, I would like to thank Feniosky Pena-Mora for the encouragement and guidance he has provided me over the past couple of year. I am truly grateful to him for the confidence that he had in me at all times, simultaneously assisting me at my pitfalls and commending me at my successes. I have learnt a lot from him not just in the field of Collaboration, but also in several other aspects of life.

I am also extremely grateful to all of the individuals at the Department of Civil and Environmental Engineering, especially at the Intelligent Engineering Systems Laboratory who have assisted me during my learning process and made my stay here an enjoyable one. I am especially thankful to our Administrative Assistant Joan McCusker, for all of the support and help that she readily offered at all times.

Finally, I would like to express my gratitude to my family, who have supported me and provided me with guidance throughout my life. I am extremely grateful to my friends at work, Kiran Choudary, Gyanesh Dwivedi, Chang Kuang, Justin Mills, Sugata Sen, Jaime Solari, Ajit Sutar, Sanjeev Vadhavkar that have provided me a great deal of fun and assistance during my stay here. Last but not least, special thanks to my apartment-mates Amit, Rama and Vinay and all the other Pakodas@mit for all the support they have given me.

Contents

List of Figures

List of Tables

Chapter 1

1.0 Introduction

One of the key competitive factors among today's corporations is the availability of professionals and corporate expertise. Due to the growing size and geographic dispersion of corporations, it is becoming increasingly difficult to make efficient use of their human resources and expertise. This trend has also forced the restructuring of organizational structures towards a decentralized model. Existing manufacturing and design processes are being strained by all these trends combined with growing complexity of modern products. Several new paradigms such as total quality management, agile manufacturing and just-in-time systems are increasingly being adopted by corporations to adapt to modern market conditions. These paradigms address manufacturing processes. The traditional product design process involves several professionals working together in a shared work space and coordinating activities through continuous design discussions. This work process is no longer applicable to realities of modern product development. Current products are increasingly complex and involve a large number of professionals from multiple disciplines. Thus collaborative engineering has become very common in most of today's large-scale engineering projects. The success of all such collaborative projects depends on effectiveness of meetings among the professionals.

These professionals cannot all be easily located at the same work space. Having meetings among these professionals who are distributed geographically and also have busy schedules becomes an increasingly difficult task. It also involves a lot of time and money. Hence a reformulation of the meeting process is necessary for corporations to remain competitive.

1.1 CAIRO Overview

CAIRO [Hussein 1995](Collaborative Agent Interaction and synchROnization) is a component of the Da Vinci effort that provides for meetings over a computer network. CAIRO provides effective communication and coordination tools for a distributed environment. The CAIRO system focuses on many of the complex collaboration control mechanisms associated with electronic meeting systems in a distributed conferencing system. Further more CAIRO provides synchronized multimedia information transmission through the standard Internet protocol.

The main objectives of the CAIRO system are:

1) The capture of process and rationale that generated a product design through the documentation of meetings.
2) The relaxation of time and space constraints in traditional meeting settings.
3) The formalization of meeting control methodologies.
4) The exploration of intelligent agent mechanisms for meeting facilitation.

To achieve these objectives a model for the distributed meeting process has been devised. Distributed conferencing systems have their own communication requirements for effective group collaboration and cooperation. Such environments commonly known as "groupware" may be defined as computer-based-systems designed to support multiple users engaged in a common task, thereby providing an interface to a shared environment.

The most important requirements of a groupware are:

1) Multiple media channels, since group communication is usually comprised of audio and visual data.
2) Multimedia channel synchronization, due to random delays inherent in the network.
3) A conference control mechanism, in order to provide efficient group interaction.
4) Adaptability to different conference styles.
5) Ability to support groups in various stages of formation.

CAIRO [Hussein 1995] also incorporates a multiple media synchronization system as well as a conference management system/agent synchronization. **Agent Synchronization** enhances the conference efficiency by providing basic control and structuring features in addition to the mechanisms for automatic moderating of conferences by the CAIRO system. **Media Synchronization** ensures that all channels of communication are played back as they had been recorded at the transmitting end.

1.1.1 The CAIRO System Architecture

The CAIRO system is made up of several inter-linked modules and servers. Each participant taking part in a CAIRO conference generates a collaboration manager, which is comprised of media drivers and message servers. Forum servers are in charge of the control of a conference among several individuals. The forum moderators that define a specific control methodology generate the forum servers. The name server maintains a directory of all participants, forum managers and forum servers within the CAIRO system.

On the whole, the CAIRO system provides computer support for a well structured conferencing mechanism with synchronized multimedia communication among the participants. Added to this is the flexible nature of CAIRO. The CAIRO architecture is extensible and can easily be modified to adapt to non-engineering collaborative efforts.

The CAIRO conferencing tool has the potential to greatly enhance collaborative work, and could considerably reduce costs and increase productivity.

Information sharing is an integral part of Engineering Project Management. Traditionally such communication has been handled in the form of person-to-person meetings. The Da Vinci initiative aims to explore the support mechanisms for enhancing distributed engineering design change negotiation. The system envisioned would include computer supported design tools, distributed communication tools, design knowledge access tools, design artifact object models, as well as a methodology and interference engine for design change management. As a part of the Da Vinci initiative, the CAIRO [Hussein 1995] develops a methodology for computer supported coordination of distributed negotiation meetings. Conferencing is one of the most difficult and very costly aspects of large-scale projects, especially when the parties are in many different parts of the country or world. CAIRO provides a means for productive conferencing through the Internet thereby reducing the cost and difficulty involved in conducting these conferences.

CAIRO has three main objectives. First, the system aims to remove the same-place constraint, which is the characteristic of most of the face-to-face meetings. This would reduce the wastage of time and money, thereby increasing the productivity of the project as a whole. Second, in addition to the removal of physical limitations of in-person meetings, the CAIRO system also aims to remove the temporal or same-time constraint. This would add convenience to the entire negotiation process. Finally, CAIRO seeks to model meeting control structures. CAIRO system implements many forms of meeting control structures, in order to facilitate the flow of information. A process may be defined as the necessary steps that need to be taken to accomplish a successful negotiation. A well defined process which is applied to a particular part of an engineering project might be reflected in a forum with a certain agenda in which distributed clients might participate with their respective agents working to meet a common goal. The object model under which the entire CAIRO system operates can be illustrated as in Fig. 1 [Hussein, 1998].

1.1.1.2 Server

The CAIRO [Hussein 1995] server is a java based application that runs on some machine that is connected to the internet over TCP/IP Protocol. It sends messages back and forth to all clients that are connected to it thus facilitating a part of the collaboration effort. The server, referred to as the *nameserver* keeps track of all the meetings that are active as well as the names of all the users that are logged on to the server.

1.1.1.2 Client/Forum

A user who wishes to use the CAIRO system must start the CAIRO client and register with the name server. The name server is a Java message-sending program running on some machine on the Internet. The name server provides the registered client with a list of all the meetings that are currently running on the CAIRO system.

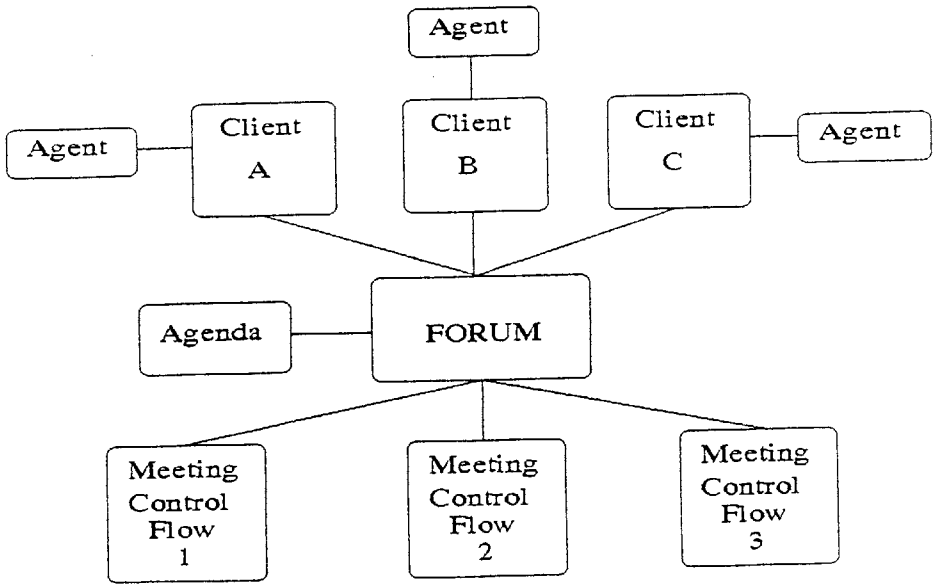

Figure 1-1. Object model Diagram for current CAIRO system [Hussein 1995].

A meeting or a Forum is another Java message sending program. Once a client decides to enter a forum, messages are then passed from the client to the particular forum. The Forum is a virtual meeting space where the collaborators can virtually collocate over the

Internet. It is very similar to a chat room, except that CAIRO [Hussein 1995] provides the clients with communication tools that are more conducive to engineering collaboration process. It also embodies the notion of a meeting control strategy, including **chaired meetings, roundtable meetings and lecture meetings**. In addition to these different meeting structures CAIRO provides the option to have **side conversation**, which is a very prevalent feature of real meetings. Associated with the CAIRO concepts of client and forums, is the notion of meeting **agenda**. An agenda presents a list of things to be done during the course of a meeting, and it should specify a notion of how long each activity should take. The Agents are a major component of the CAIRO system. An agent can be generalized as a software component that works in conjunction with people or represents people and acts in their best interest. The Agent keeps track of the allotted time for each of the agenda items and prompts the clients whenever necessary. The clients can however override the agent's suggestion.

1.1.3 Means of Collaboration

Independent of how much structure has been built into the CAIRO system to control and manage the distributed collaboration process, the process, as a whole would be futile without tools for communication. In real meetings, tools such as black boards, overhead projectors and note pads are used to collaborate. CAIRO also includes a couple of such features. They are the message board (for passing text messages) and the white board similar to a black board. Moreover, the overall design for the implementation of CAIRO tools is that of a plug and play nature. A perfect example of this kind of implementation is the scheduling tool.

An important aspect of any effective collaborative tool would be multimedia such as audio and video. A large portion of the current research has been in the area of identifying meeting protocols with respect to multimedia and implementation thereof. Implementation of the audio/video conferencing has been achieved using the Java Media Framework Application programming interface. Study was also conducted in the area of three-dimensional meeting environments and a three dimensional meeting environment has been implemented using the Java3D Application programming interface. The Java

Speech Application programming interface has also been used for the implementation of a text-speech system so that users can speak into a microphone instead of typing into the text tool. The speech engine of the IBM Viavoice has been used to facilitate the speech recognition.

1.2 Objectives of This Research

The Collaborative Agent Interaction control and synchROnization (CAIRO [Hussein 1995]) system is a distributed meeting environment that was developed at the Massachusetts Institute of Technology specifically for civil engineering project development [Benjamin 1998]. This system was created as part of the Da Vinci initiative whose goal is to "explore computer support mechanisms for enhancing distributed engineering design change negotiation" [Pena-Mora, et. al. 1996]. One of the primary goals of the CAIRO endeavor is to add structure to the collaboration process in the virtual space by defining various protocols of interaction that are commonplace in the physical space. The term "physical space" refers to the tangible world where individuals meet with one another face-to-face. On the other hand, "virtual space" represents the artificial environment that can be created and reached through computers. The functions of the virtual environment typically imitate the functions of the real world. In this respect, meeting protocols common to the physical world that control meetings are replaced by computer code that attempt to duplicate these protocols in an electronic environment.

The CAIRO initiative has taken the meeting protocols mentioned above and implemented them in its distributed meeting environment. Its collaboration scheme is based on an event-passing paradigm [Hussein 1997] to coordinate interactions among participants. This research aims at the implementation of multimedia support for distributed meeting environments as such. More specifically this research looks at Audio/Video components, Three dimensional meeting environments, Speech recognition and synthesis.

1.3 Hypothesis of This Research

By developing and using a prototype of a multimedia based collaboration system, experience can be obtained for generating a roadmap for next generation systems. Based on the advantages and disadvantages of the prototype, a new set of requirements can be developed that define needed changes. This study produces a blueprint that documents the design path for a next generation system. Future developers can then follow this blueprint in creating the next version of the system.

The architecture for the new multimedia based collaboration system will retain all the core features of the earlier CAIRO [Hussein 1995] system. The additional features that are added will be the audio/video component that will have the core collaboration features like the meeting protocols integrated and the 3D meeting environment that will have the basic functionality offered by the current 2D meeting environment. It will also have Speech to text support so that instead of typing in the chat window the user can just speak out what he wants to type and the recognition system captures his voice, recognizes it and sends text to the other users. These three basic components form the multimedia suit that supplements the collaboration offered by the core of the CAIRO system.

1.4 Benefits of This Research

The successful development of this collaboration system will greatly enhance the effectiveness of meetings. First of all, this new system will enable completely distributed meetings to occur with multimedia support facilitating users to be able to hear each other's voice and look at each other's video. The 3D meeting environment will e useful for low bandwidth users so that it can substitute video to some extent. Some minimal set of actions like nodding the head and hand raising can be depicted in the 3D virtual environment.

Advancement in Technology has aided in the availability of high end capture devices that make multimedia based collaboration possible. The increased processing speeds of today's Computers and also the availability of high band width have resulted in

making possible the use of multimedia applications that require heavy processing capacity and also high data transfer rates.

Chapter 2

2.0 Background

This Chapter outlines the basic architecture of the CAIRO system. It deals in depth the various features of the CAIRO system and describes how the different components work together.

2.1 The CAIRO System

The CAIRO [Hussein 1995] system was developed in the Civil and Environmental Engineering department at the Massachusetts Institute of Technology as a tool to help engineers collaborate remotely to plan large engineering projects [Benjamin 1998, Hussein 1997]. It has functionality similar to a regular chat daemon; however, it has extra functionality that sets it apart from the typical chat rooms one may find on the Internet.

2.1.1 The Architecture

The distributed architecture of the CAIRO system is built upon the concept of client/server technology. Figure 2-1 illustrates an instance of the CAIRO system architecture. In this representation, there are four participants (a, b, c, and d) in the

discourse. The design consists of three main elements: the Collaboration Manager, the Forum Server, and the Name Server. The following sections explain the roles of these constituents.

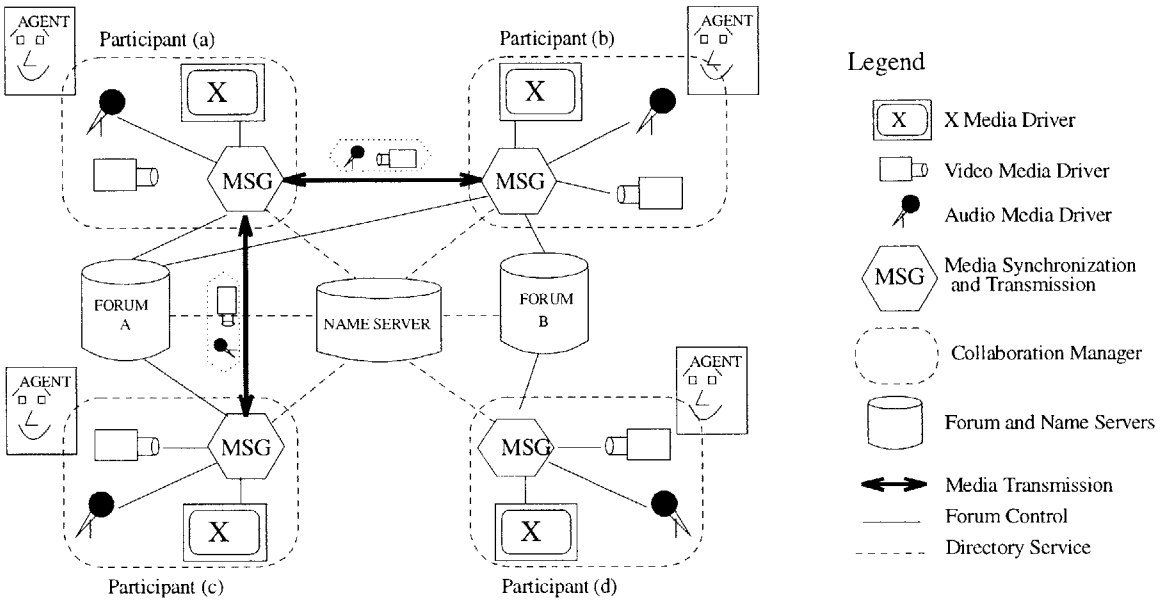

Figure 2-1. Current CAIRO Architecture [Hussein 1995].

2.1.1.1 Collaboration Manager

The collaboration manager is comprised of a number of media drivers and a message server. It also possesses a Graphical User Interface (GUI) that allows the user to easily navigate the system (see Figure 2-2). Encapsulated within the interface are numerous other tools, menus, and visual analogies that facilitate collaboration among participants. The interface and its features are discussed in Section 2.1.2.

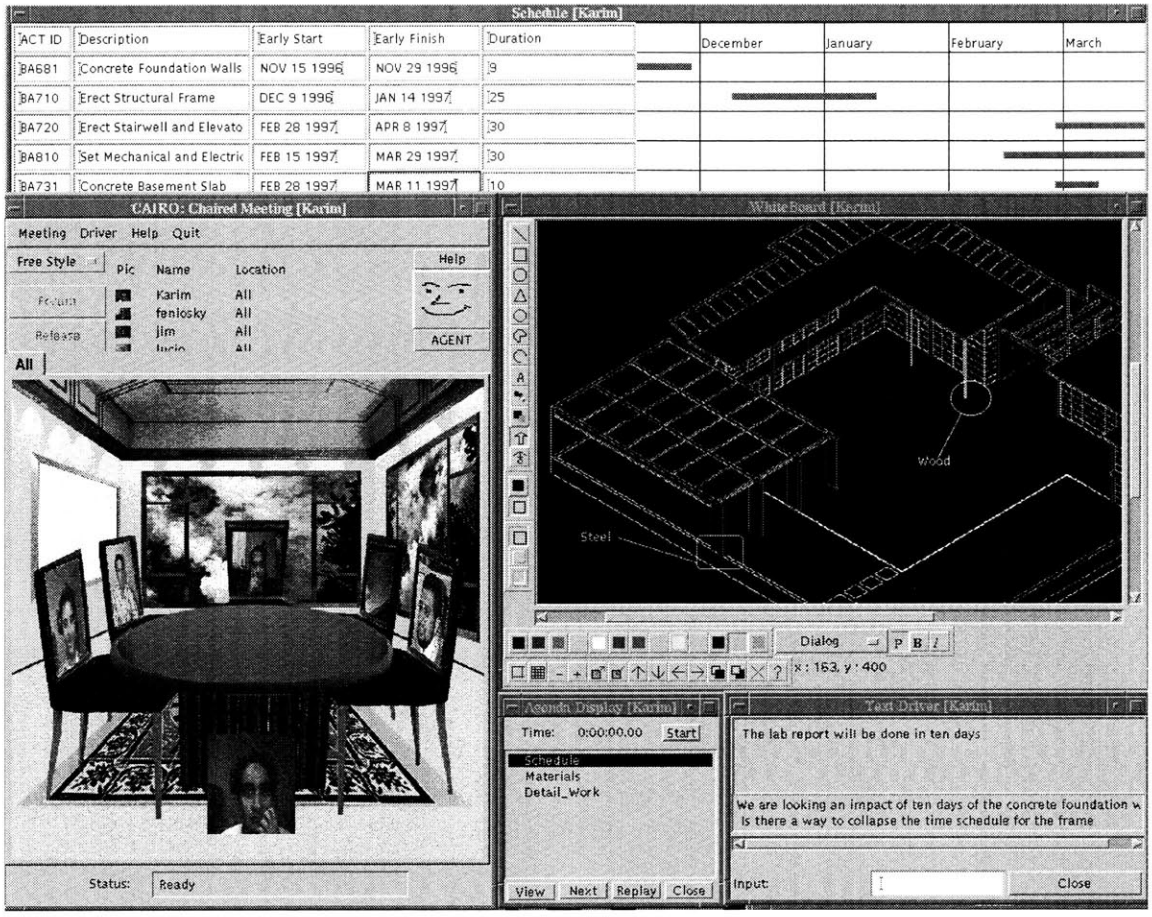

Figure 2-2. CAIRO Collaboration Manager Interface [Fu 1999].

2.1.1.1.1 Media Drivers

The term "media driver" describes a set of applications that users employ to transfer various forms of data to other users. The original version of CAIRO consists of a set of drivers that are specifically geared towards engineering project development. These drivers, which can also be seen in Figure 2-2, include a text driver, a whiteboard, a scheduler, and a design rationale and implementation tool (not pictured). The text driver permits the exchange of text messages among the clients so that they can communicate through the use of written language. The whiteboard provides a shared workspace that simulates an actual physical whiteboard that may be found in an office setting. This driver allows members to communicate information in the form of sketches or drawings. It is also capable of loading images and transmitting them to other participants. The

scheduling tool includes an interface that describes a plan in the context of time. It lets users create and modify a schedule in a group setting. Lastly, the design rationale and implementation tool enables participants to collaborate about design issues of an engineering project. The availability of these tools greatly augments the ability to convey specific engineering ideas from one person to another.

2.1.1.1.2 Message Server

This server handles all transmission of information between users and keeps track of membership in forums. The term "forum" is defined in Section 2.1.1.2. Messages that are exchanged between various participants in a forum are all in the form of TCP/IP datagrams. Each participant has his/her own handler that interprets those incoming messages and routes them to the appropriate drivers.

2.1.1.2 Forum Server

A forum is defined as "a structured group of participants involved in a collaborative effort" [Hussein 1995]. In this document, the terms "forum", "meeting", "engagement", "conference", "session", and "collaboration instance" are used interchangeably, and all refer to this same definition. The forum server forms an analogy to the actual, physical meeting space. It keeps track of all of the individuals involved in a specific meeting and their current state (see Figure 2-3). A member of a forum may be in one of three states: active (logged in and listening), speaking (has control of the floor), or non-active (not logged in). In addition, the Forum Server maintains a log of events that occur within the forum for playback purposes. Thus, a user can go to specific entries on the agenda and replay the events that took place during that agenda item. The playback of the recorded log is only seen by the individual who requested the replay and is not saved in the event log.

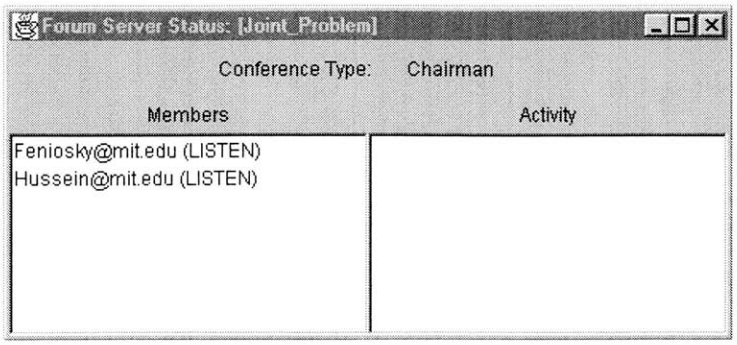

Figure 2-3. Forum Server Interface.

The forum server also has the responsibility of regulating floor control, one of the most important features in CAIRO. Three different meeting styles are presently defined: chairman, freestyle, and lecture. Each of these styles exercises a set of controls to regulate when and to whom users can speak.

2.1.1.2.1 Chairman Meeting

The chairman meeting style within CAIRO is modeled after the real world chairman meeting. In this meeting strategy, one participant is designated by some prior decision to be the chairman of the engagement. This person is then given authority to determine who gets control of the floor. The chairman himself can take control of the floor as he/she wishes. However, other members of the meeting must request permission from the chair to speak. The chair then decides whether he/she will grant permission to the requestor. These actions are performed using the menus described in Section 2.1.2.3.

2.1.1.2.2 Freestyle Meeting

The freestyle meeting style has an informal composition. Unlike the chairman style, no one is in charge of the meeting. Instead, members are allowed to take the floor and speak of their own accord. This style of meeting is common for brainstorming sessions in order to keep the conference proceeding smoothly. People can speak freely as ideas flow into their minds, and thus, can contribute ideas without the constraint of obtaining permission from another individual.

2.1.1.2.3 Lecture Meeting

The lecture meeting style is, as can be inferred by its name, analogous to a lecture or classroom setting. In this case, the lecturer assumes a role similar to the chairman in the chairman strategy. However, primary communication is directed to the lecturer and no communication takes place between participants in the lecture.

2.1.1.3 Name Server

The Name Server centrally manages a global directory for the CAIRO system. This server catalogs a variety of information that can be retrieved by any user. This information includes the name, location, and status of each participant. Likewise, it retains knowledge about the forum's name, location, and status. When a user logs into the CAIRO system, the Name Server can provide information to the user such as the other individuals who are logged into the system and what forums are available to the user. Unbeknownst to the user, information about the Internet address of the other participants and the forums are sent to the client as well. Figure 2-4 exhibits the name server interface where the list of forums and users online are displayed.

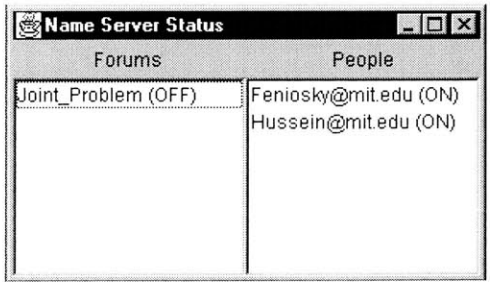

Figure 2-4. Name Server Interface.

2.1.2 The Features

CAIRO possesses many resourceful features that produce an enriching collaboration experience. These features are what distinguish the CAIRO meeting

environment from other collaboration systems, such as video teleconferencing, on the market today. The next few sections describe the features in more detail.

2.1.2.1 The Graphical User Interface

One of the intended advantages of the CAIRO meeting environment compared to other collaboration systems available in the marketplace is that CAIRO provides a number of visual analogies to actual physical meeting spaces. As an example, the image of the table visible in Figure 2-2 can take on two different shapes. The round version cues the viewer that the meeting he/she is participating in is a freestyle meeting. In contrast, a rectangular shaped table denotes a meeting with a chairman. These visual analogies have dual concepts in the physical world that may be familiar to the typical individuals. As another example, the status panel located in the upper half of the main collaboration window in Figure 2-2 contributes information about whether a participant is speaking, being addressed, or being requested. This feature is described more fully in Section 2.1.2.2.

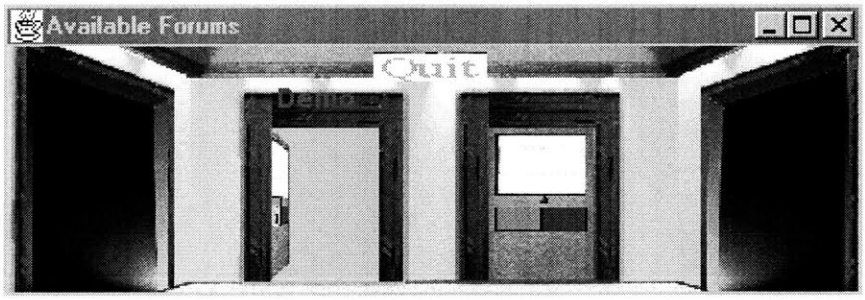

Figure 2-5. Hallway of Meetings.

Figure 2-5 demonstrates a third instance of the intended richness of the visual medium. In this window, the analogy of a hallway is used to represent the choice of meetings that are available at a particular instance in time. Each door represents a separate meeting and has a title describing the meeting over it. At the left and right sides of the image lie extensions to the hallway through which users can to reach more rooms.

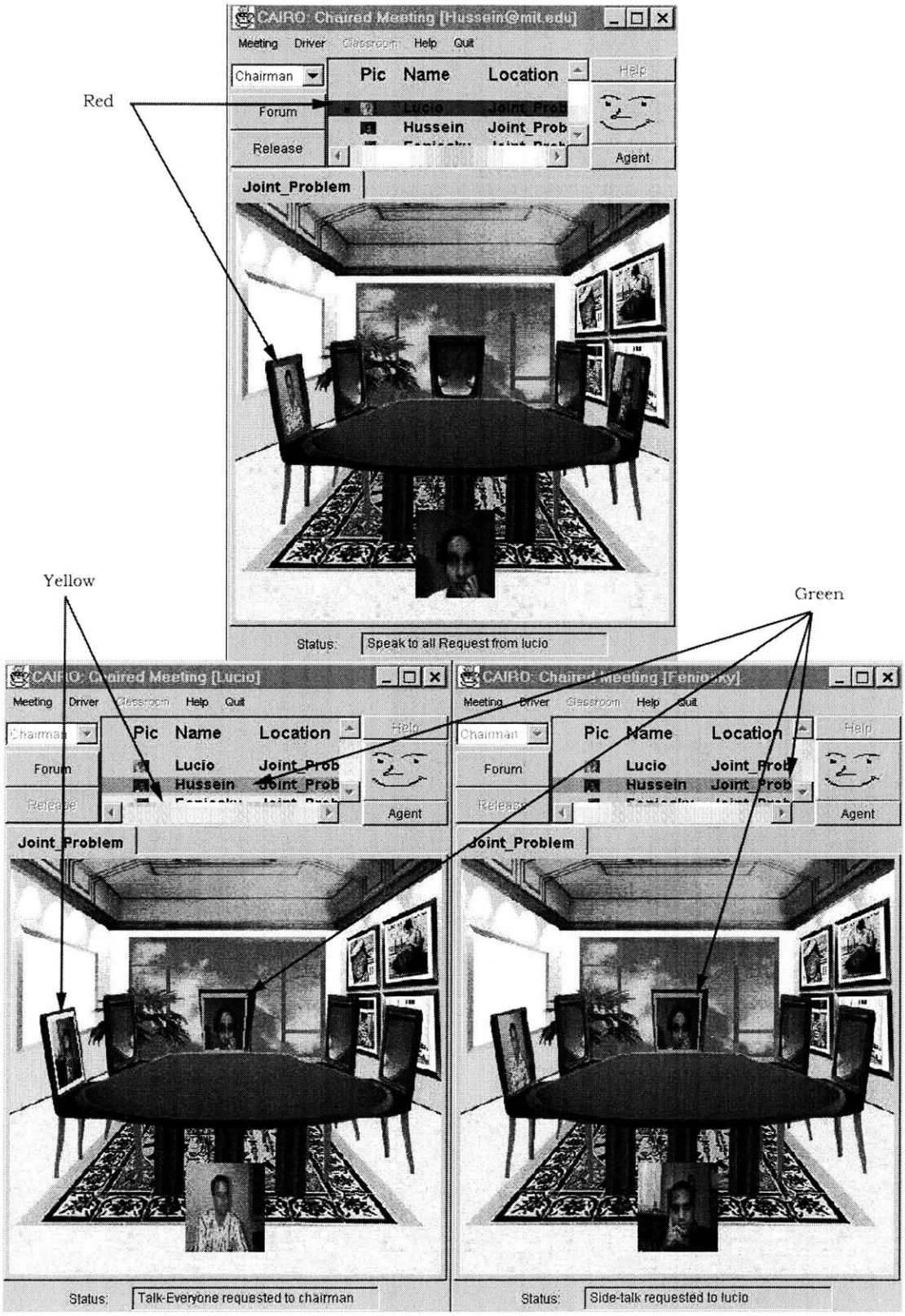

Figure 2-6. Status Panel.

Once a user finds the room that he/she would like to join, he/she simply clicks on the corresponding door. It then opens as shown in Figure 2-5. An open door signifies that a room has been entered. Therefore, this doorway represents an entry to our virtual meeting. For each of the next three figures (Figures 2-6 through 2-8), the users are identified by name as Lucio, Feniosky, and Hussein. These names are used to refer to these users in the rest of this chapter. Hussein's interface resides in the top window while Lucio and Feniosky are in the bottom left and right windows respectively. Their names also appear in the title bar of each window. The rectangular table in this snapshot intends to convey that a chairman style meeting is in progress. Both Feniosky and Lucio's windows show Hussein as the chairman. Hussein's photo also appears at the far end of the table to signify his status within the meeting. Hussein, in contrast, sees no one at the other end of the table.

2.1.2.2 The Status Panel

Figure 2-6 exhibits one example of the abundance of information that can be conveyed to the user through the status panel. At first glance, one may think that the status panel only shows the name and location of a user. However, much more information is concealed in this panel. In this screenshot, the Feniosky client, at the bottom left, has gained permission to speak to the entire group. The state of the meeting can be determined by a quick glance at the status panel. The green highlight on Hussein's name and picture in the windows of Feniosky and Lucio signal that Hussein presently has control of the floor. Similarly, on the Hussein's screen, the dots next to the names of Feniosky (cannot be seen due to size of status panel) and Lucio tell him that he is speaking to the other two participants. The red highlights seen in Hussein's interface around both Lucio's name and image signal to Hussein that Lucio has requested permission to speak. In this scenario, Hussein is the chairman, so he receives all requests to speak. Finally, the yellow highlights that appear around Feniosky's name and image in Lucio's window signal to Lucio that Feniosky is requesting a side-talk with him.

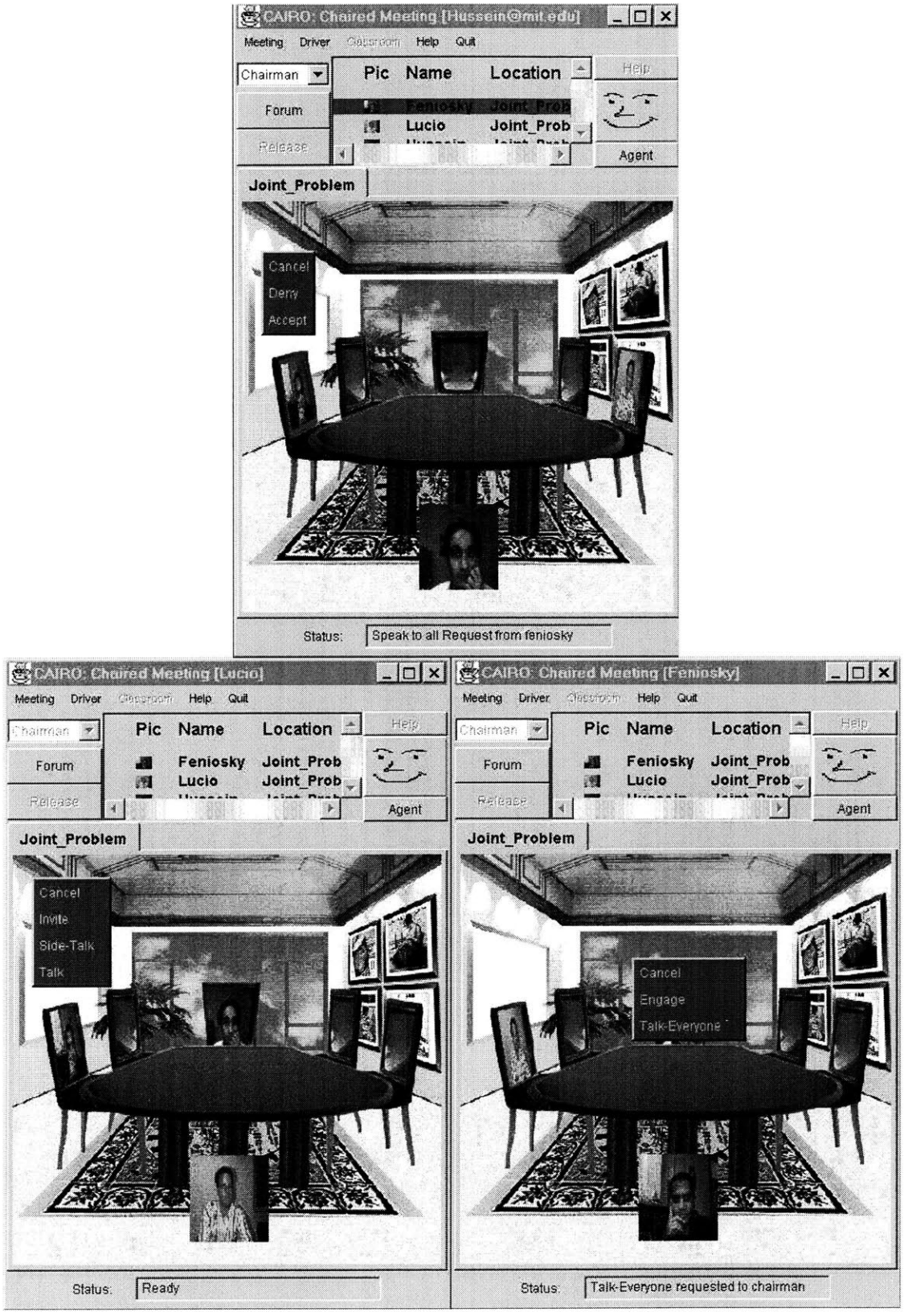

Figure 2-7. Menus

2.1.2.3 The Menus

CAIRO provides a series of pop-up menus for users to moderate the collaboration process. For example, if a user wishes to get permission to speak to the whole group, he/she can click on the table, which in turn generates a menu with three options as illustrated in Figure 2-7. In this example, Feniosky has activated the table menu on his window. The table menu gives Feniosky the opportunity to send a request to the chairman of the meeting, Hussein, by clicking on the "Talk-Everyone" option.

Lucio, on the lower left, has exercised the side-talk menu by clicking on another participant's image. The "Talk" and "Side-Talk" options both send requests to speak to the selected user. However, the "Talk" option, if accepted, only establishes a direct link between the two users within the context of the meeting. On the other hand, the "Side-Talk" menu item causes a new collaboration instance to be initiated. The Side-Talk feature is explained in more detail in Section 2.1.2.4. The "Invite" feature is only enabled when a side-talk instance is active. It allows participants within a Side-Talk to bring other collaborators into the Side-Talk. Finally, a user can respond to a request from another participant by clicking on that participant's image as Hussein is doing at the top. This click of the mouse generates yet another menu that allows him to either accept or deny the request. These menus attempt to simplify communication by reducing the protocol between participants to mouse clicks.

2.1.2.4 Side-Talk

In Figure 2-8, a Side-Talk is in progress between Feniosky and Lucio. In the main image panel, notice that their interfaces contain an extra tab labeled "feniosky 1". This tab, consisting of the side-talk initiator's name and an ordinal number, represents a Side-Talk. The number is incremented for each side-talk that is initiated, thus allowing for one person to initiate and participate in multiple Side-Talk instances simultaneously. In this case, information disseminated in this folder tab is only transferred between Feniosky and Lucio. Hussein sees none of their conversation and is completely unaware that this side conversation is taking place.

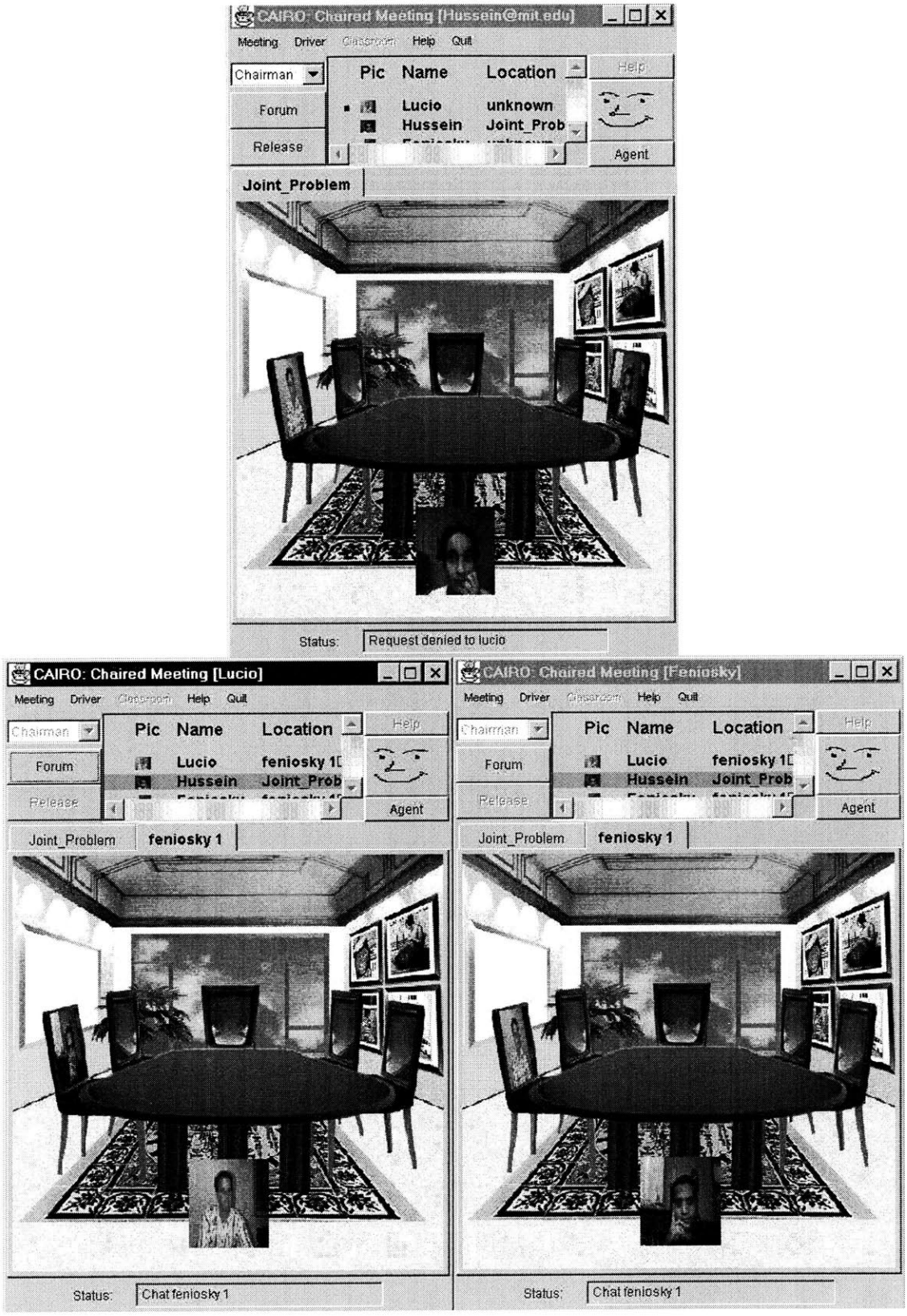

Figure 2-8. Side-Talk.

This feature allows a subgroup of a conference to speak privately during a meeting. One can conceive a scenario in which a subcommittee is formed to discuss a side issue. In this case, the side-talk function could be used for the purpose of the subcommittee. Lastly, participants of the Side-Talk can switch between tabs, and by doing so, can alternate their participation between the main meeting and the sidebar conversation.

2.1.2.5 The Agenda Tool

The agenda tool identifies the order of a meeting. As in a physical meeting, the agenda contains an outline of topics to be discussed. The agenda in CAIRO works in conjunction with the forum to control the flow of a meeting. The agenda recognizes the duration of each agenda item. CAIRO also provides an agenda wizard tool that can be run to interactively create an agenda for a meeting. This wizard has a graphical user interface that guides users through the steps in generating a schedule for a meeting. The agenda tool, presented in Figure 2-9, is simply comprised of a window that has a timer to keep track of the time spent on an agenda item and a view of the items in the agenda. The panels, from left to right, identify the name of the agenda item, the person in charge, and the style of meeting.

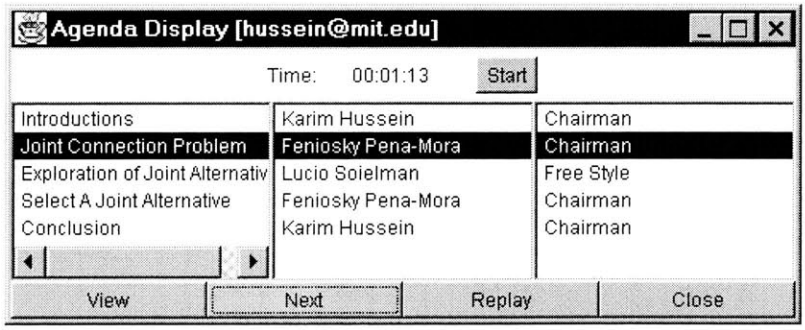

Figure 2-9. The Agenda Tool.

2.1.2.6 Collaboration Manager

The Collaboration Manager incorporates the CAIRO user interface and maintains lists of available media resources and forum servers (Figure 2-10). The Collaboration Manager

enforces conference controls associated with the forums in which the user is participating. For example, a specific forum may not allow any conversations with users outside of the forum or may not permit any private side conversations with other members of the forum.

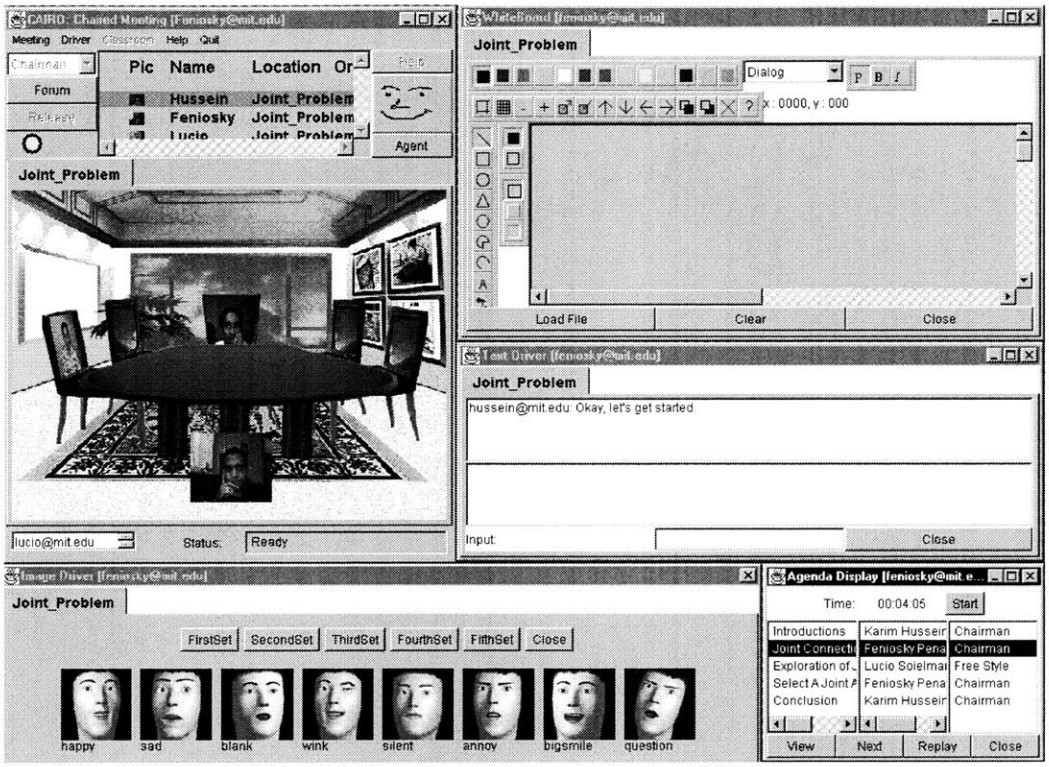

Figure 2-10. A sample session of CAIRO.

2.1.2.7 Media Drivers

Media drivers handle all I/O between the Multimedia collaboration system and the underlying media channel. Each driver is tailored specifically to the underlying media represented. Each driver is responsible for data acquisition and frame compilation for transmission and replay. This module must also provide the multimedia server with synchronization information, frame size, and delay and error tolerances. Several media drivers have been implemented that enable distributed schedule coordination, shared whiteboards for sketching and a text tool for chatting.

2.1.2.7.1 Text Driver

This is a driver that allows the exchange of short text messages among the participants. Lengthy text entries may also be pasted into the text entry input box for transmission to conference participants.

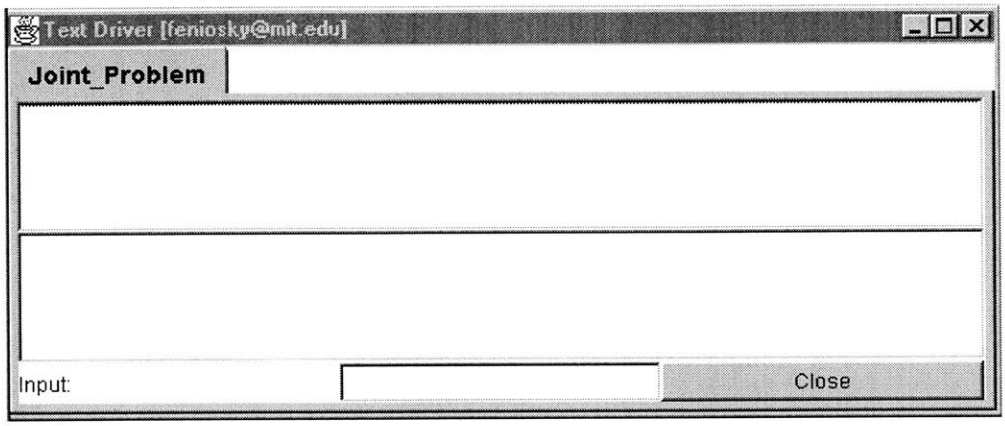

Figure 2-11. Textboard.

2.1.2.7.2 Shared White Board

This is a driver for an application that simulates a Blackboard in an office environment. It can be shared among the members of a forum to communicate visual information such as

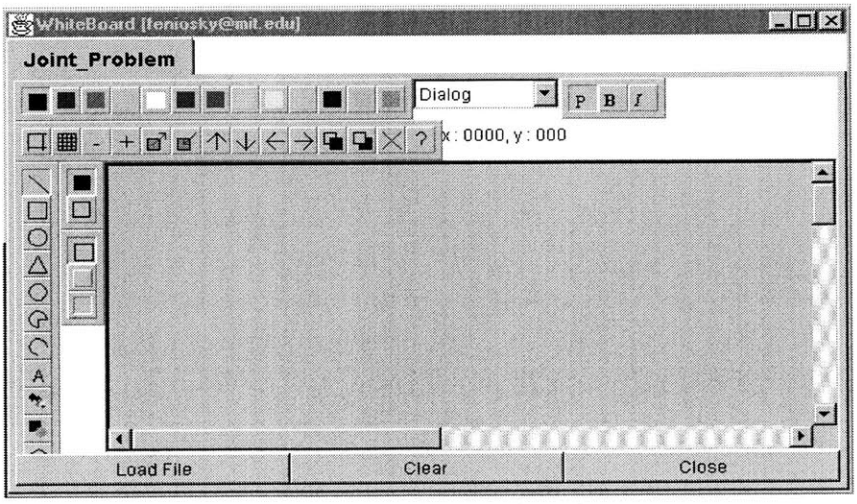

Figure 2-12. Whiteboard.

sketches of various product design ideas. It can also used to transfer bitmaps of images on the user's screen to the rest of the team.

2.1.7.2.3 Shared Schedule

This is a driver for an application that interacts with a Primavera scheduling engine for large-scale project scheduling. It can be shared among the members of a forum to allow for schedule modifications and additions.

2.1.7.2.4 Expressions

This is a driver for an application that interacts with the Collaboration Manager and the Text Driver. This has several sets of images depicting moods like sad, happy, joking, silent and angry. The faces of the users on the Collaboration Manager change based on the mood of the person.

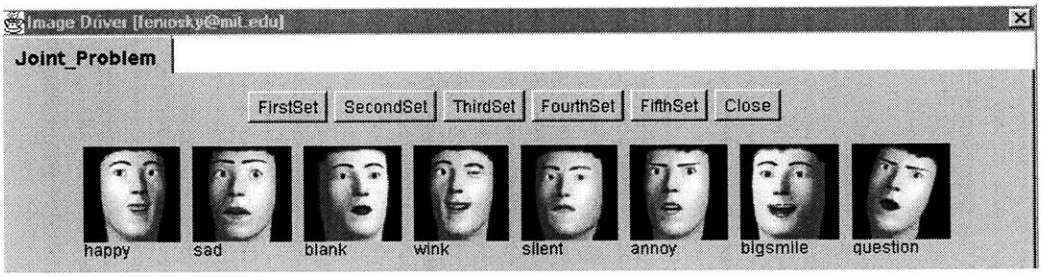

Figure 2-13. Expressions.

2.1.7.2.5 3Dimensional Meeting Environment

This is a driver for an application that represents that entire meeting environment in three dimensions. This is a parallel to the initial two-dimensional meeting environment and has a three dimensional meeting space with tables and chairs that are Virtual Reality Markup Language (VRML) files that are loaded from a centralized server. Three-dimensional human forms called Avatars are also loaded as members join the meeting. The Java 3D Application Programming interface and VRML 2.0 have been used in the implementation of this component.

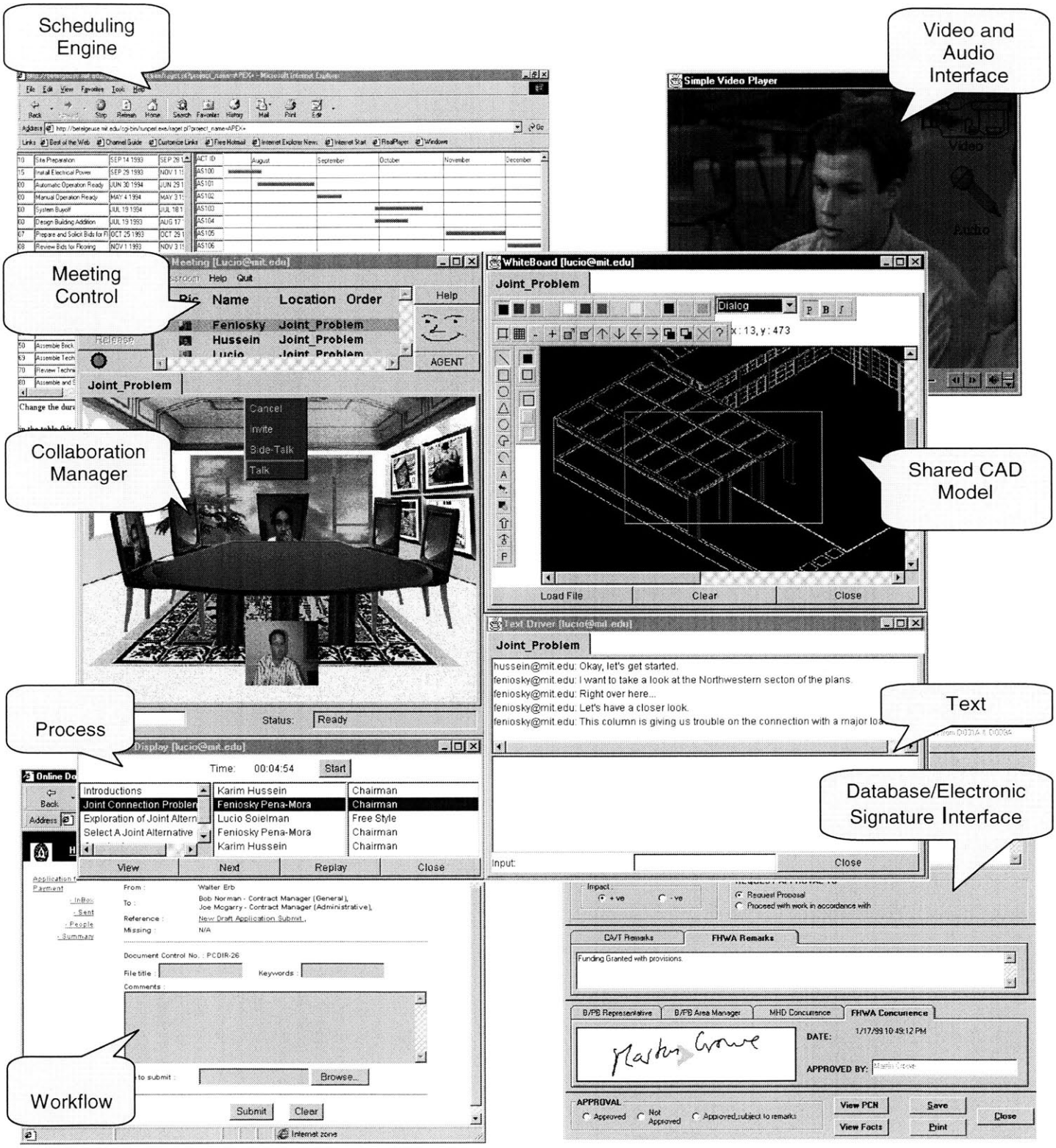

Figure 2-14. CAIRO Feature Set.

2.1.7.2.6 Audio/Video Drivers

This is the driver for the audio/video conferencing component. This driver has been implemented such that it can be instantiated by several means. The user may click on another user and send or receive audio and video. The user may also click on the table and send or receive audio and video to the rest of the meeting members. The Java Media Framework Application Programming Interface has been used in the implementation of this component.

2.1.7.2.6 Speech to Text Driver

The Speech to Text driver is a built-in feature that facilitates the user to dictate to the Text chat application instead of actually typing into it. The Java Speech API has been used along with the IBM ViaVoice speech engine.

2.1.7.3 The Agent

The CAIRO agent is one of the integral components that differentiate CAIRO from other collaboration systems. The agent is a program that monitors the actions of the users, residing with the moderator of the meeting. The agent procures information from the interactions among users and tries to make intelligent suggestions to the moderator that may increase efficiency in the meeting proceedings. Currently, the CAIRO agent possesses only limited functionality, focusing on the agenda, certain textual cues, and request activity.

During a meeting, if the time allotted for an agenda item has expired, the agent will prompt the moderator to continue to the next agenda item. In this case, the agent is a passive timer for the user. The client can choose to accept the suggestion or reject it depending on how he/she feels the meeting is proceeding. However, the agent still serves as a reminder that keeps the participants on topic and in-line with their meeting agenda.

The agent also reacts to certain textual phrases that are sent and received through the text driver. It parses the text that is being delivered for specific key words. For example, if the agent sees a word such as "explain", it may suggest to the moderator that he/she change the meeting style to a lecture control strategy. If the word "discuss" were used, then the agent would suggest that the freestyle meeting strategy be employed. Again, the user can then decide whether to accept the suggestion or not.

Figure 2-15. The CAIRO Agent.

Finally, the agent also responds to actions that are being taken by the users. If the agent notices that a lot of requests to speak are being made by the participants, then it may recommend that a freestyle meeting strategy be applied. This way, the meeting may run more smoothly if users do not have to constantly request permission to speak. Figure 2-10 shows a sample view of the CAIRO agent.

2.2 Conclusion

In this chapter the various features of the CAIRO system were studied and and outline of how all the components of CAIRO work together was laid out. The following chapter deals with the methodology and the technology used in the implementation of the Media Infrastructure of the CAIRO system.

Chapter 3

3.0 Methodology

This chapter deals with the methodology and the technology used in the implementation of the Media Infrastructure of the CAIRO system. It overviews the three main media components of the Media architecture of CAIRO, namely the Audio/Video conferencing, Speech recognition and the three-dimensional meeting environment.

3.1 An overview of the Java Media API

The Java Media APIs meet the increasing demand for multimedia in the enterprise by providing a unified, non-proprietary, platform-neutral solution. This set of APIs supports the integration of audio and video clips, animated presentations, 2D fonts, graphics, and images, as well as speech input/output, 3D models and telephony. By providing standard players and integrating these supporting technologies, the Java Media APIs enable developers to produce and distribute compelling, media-rich content.

Whether striving for a stronger competitive position, improved communications, better customer service, or reduced costs, enterprises are faced with a common challenge - aligning their information technology to achieve business objectives. The Internet, corporate intranets, and the integration of diverse media types play a significant role in

meeting this challenge and are fundamentally redefining how business works. A growing force fueling this trend is Java technology, the phenomenon that is shaking the Web and software development to its very foundation.

3.1.1 Java Media APIs: Enhancing Enterprise Applications

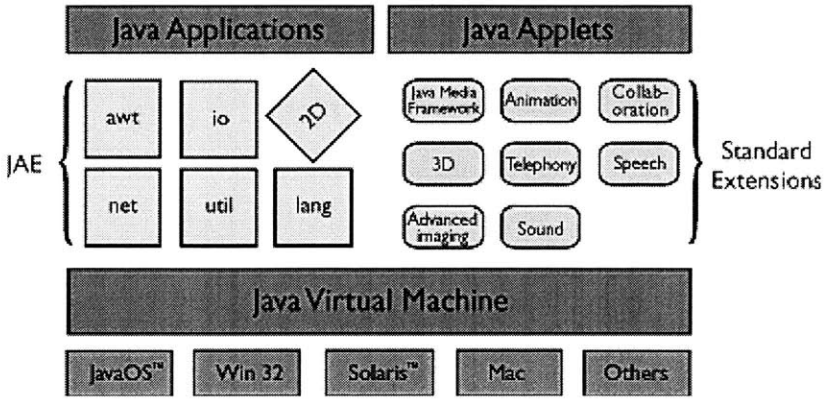

Figure 3-1. Java Media APIs [JMedia, 2000].

The Java Media APIs expand on the Java platform and deliver the benefit of "Write Once, Run Anywhere to enable the integration of media-rich content and advanced communications capabilities that will support the way business is evolving.

3.1.2 Merging Rich Media with Enhanced Communications

Java Media APIs [JMedia, 2000] are comprised of a set of class libraries that enrich the Java application environment with complete media and communications capabilities. They provide developers and users unrivaled ease and flexibility to take advantage of a wide range of diverse, interactive media on the Web and across the enterprise.

The suite of Java Media APIs provide capabilities in the following areas: audio/video playback, capture, conferencing; 2D and 3D graphics ,animation, computer telephony, speech recognition, synthesis, collaboration, advanced imaging, and other technologies. The model offered by Java and the Java Media APIs is simple: provide the freedom and ability to incorporate rich functionality - from 2D graphics to computer telephony to

realtime video - in new ways that create compelling and cost-effective business applications. These applications allow enterprises to:

- Produce more compelling content, engaging collateral, and exciting marketing material that has a big impact and effectively targets users with increasingly sophisticated tastes.
- Save time and money by coding applications once and deploying across multiple platforms for multiple purposes; for example a video or animation clip could be developed once and repurposed for manufacturing, sales training, and targeted marketing programs.
- Enhance cost-effectiveness through collaboration and conferencing that encourages more frequent "virtual" meetings while minimizing travel costs and overhead.
- Gain consistency and synergy in marketing messaging and advertising through reuse of media such as images, animation, and videos and audio clips.
- Launch new products or services. For example, telephone companies can use the Java Telephony API along with other Java Media APIs, to offer Web-based services directly to their customers.

3.2 Technical Overview of Individual APIs

This section provides a brief technical overview of all the individual Java Media APIs and the main features they offer.

3.2.1 Java Media Framework

The Java Media Framework API (JMF) is a collection of classes that enable the display and capture of multimedia data within Java applications and applets. JMF provides a set of building blocks useful by other areas of the Java Media API suite. For example, the JMF provides access to audio devices in a cross-platform, device-independent manner, which is required by both the Java Telephony and the Java Speech APIs. In addition, the playback of time-based media such as audio or video needs to be handled in the same consistent method.

Multimedia covers a wide range of requirements, which can be roughly classified as follows:

- Media Display - Playback of local and network multimedia data within an application or applet.
- Media Capture/Creation - The ability to record, save, and transfer data through local capture devices, such a microphones and cameras.
- Conferencing - Full-duplex, streaming media with real-time constraints.

To address the needs of our target customer, the Java Media Framework APIs are being released in a stages. The first API release supports media playback. The second API release will support media capture, transmission and a pluggable codec interface.

3.2.1.1 Java Media Player

The Java Media Player provides a set of high-level interfaces for reception and playback of arbitrary time-based media such as MPEG-1, QuickTime, AVI, WAV, AU, MIDI, and real-time streaming audio/video. Java Media Players can present multimedia data from a variety of sources by providing a universal resource locator (URL). Media data can come from reliable sources such a HTTP or FILE, or a streaming source such as video-on-demand (VOD) servers or Real-Time Transport Protocol (RTP).

3.2.2 Java 2D

The Java 2D API provides a powerful, flexible framework for device- and resolution-independent 2D graphics. The Java 2D API greatly enhances the existing functionality of the java.awt graphics and imaging classes. It supports arbitrary shapes, text, and images and provides a uniform mechanism for performing transformations, such as rotation and scaling, on these objects. With the Java 2D API, vendors and application developers can support a wide array of presentation devices, such as displays and printers, as well as image formats and encodings, color spaces, and rendering techniques and effects. The Java 2D API also offers comprehensive text handling and color support.

The Java 2D API delivers several innovative features for advanced graphics and imaging including:

- Antialiased rendering and other richer rendering attributes
- Bezier paths
- Transforms
- Compositing
- Alpha channel capability
- Accurate color space definition and conversion
- Rich set of display-oriented imaging operators
- Arbitrary fill styles
- Stroking parameters for lines and curves
- Extended font support and advanced text layout
- Extended imaging operations, such as convolution, lookup tables, and transformations
- Flexible image formatting
- Interfaces for supporting arbitrary graphics devices such as printers and screens
- ICC profile support

3.2.3 Java 3D

The Java 3D API is an application programming interface for developing three-dimensional graphics applications and applets. It gives developers high-level constructs for creating and manipulating 3D geometry and for building the structures used in rendering that geometry. Application developers can describe very large virtual worlds using these constructs, which allows Java 3D to render these worlds efficiently.

The Java 3D API draws its concepts from existing graphics APIs as well as from new technologies. The Java 3D API's low-level graphics constructs synthesize the best ideas found in low-level APIs such as Direct3D, OpenGL, QuickDraw3D, and XGL . Similarly, it's higher-level constructs synthesize the superior concepts found in several

scene-graph-based systems. The Java 3D API also introduces some concepts not commonly considered part of the graphics environment, such as 3D spatial sound. The Java 3D API's sound capabilities help to provide a more realistic experience for the user. The Java 3D API provides the following features:

- High-level, object-oriented, scene-graph-based programming paradigm
- High-performance rendering of 3D graphical data
- Sophisticated optimizations such as culling, pruning, and parallel scene-graph traversal
- Ability to layer on top of existing low-level 3D APIs
- Rich set of features for creating and visualizing compelling 3D worlds
- 3D geometric primitives, such as triangles, lines, points, text, and compressed geometry
- 3D spatialized sound
- Rendering attributes such as 3D transformations, lighting, shading, fog, texture mapping, aural characteristics, picking
- Sophisticated viewing model which seamlessly supports monoscopic or stereo viewing
- Advanced behavioral model which provides user-specified behaviors and execution culling
- Advanced input/sensor model which enables support for 6-degree-of-freedom tracking devices
- Support for runtime loaders to accommodate a wide variety of file formats, such as vendor specific CAD formats, interchange formats, VRML 1.0, and VRML 2.0.

3.2.4 Java Telephony

The Java Telephony API (JTAPI) is a set of modular application programming interfaces designed to merge computer and telephone capabilities. JTAPI enables developers to create platform-independent, telephony-enabled Java applications and applets that can be

deployed for a wide range of environments including customer service, Internet telephony, interactive voice response, directory services, and PBX's.

JTAPI supports both first- and third-party telephony applications and includes the following APIs:

Core: The framework that models telephone calls and features: placing a call, answering a call, and dropping a call

Call Control: Provides more advanced call-control features: hold, transfer, and conferencing telephone calls

Call Center: Provides advanced features necessary for managing large call centers: call routing, automated call distribution, and predictive dialing

Phone: Provide control of physical features of phone sets

Capabilities: Allow applications to query whether certain actions may be performed

3.2.5 Java Speech

The Java Speech API defines a software interface which allows developers to take advantage of speech technology for personal and enterprise computing. By leveraging the inherent strengths of the Java language, the Java Speech API enables developers of speech-enabled applications to incorporate more sophisticated and natural user interfaces into Java based applications and applets that can be deployed on a wide range of platforms. Two core speech technologies will be supported through the Java Speech API: speech recognition and speech synthesis.

3.2.5.1 Speech Recognition

Speech recognizers can be defined and selected by characteristics including language, the type of grammar supported, and the preferred audio quality. The core API specifications for speech recognition in the Java Speech API provide control of speech input, notifications of recognition results and other important events, vocabulary management and the definition of recognition grammars.

The basic controls for a speech recognizer are pausing and resuming of recognition. However, the most important control of a speech recognizer is in the management of recognition grammars. The two basic grammar types currently being specified as part of the Java Speech API are rule-based grammars and dictation grammars. For both grammar types, an application can load multiple grammars into a recognizer and activate and de-activate each grammar as required.

3.2.5.2 Speech Synthesis

For speech synthesis, the Java Speech API provides the control of speech output, notifications of speech output, vocabulary management, and the format of text to be spoken.

Java Synthesis Markup Language (JSML), an SGML-based markup language, is used for formatting text input to speech synthesizers. JSML allows applications to control important characteristics of the speech produced by a synthesizer. Pronunciations can be specified for words, phrases, acronyms and abbreviations to ensure understandability. Explicit control of pauses, boundaries, emphasis, pitch, speaking rate, and loudness at the word and phrase level can be used to improve naturalness and understandability.

3.3 Conclusion

As described in Chapter 2, CAIRO has a set of Media Drivers that enhance the collaboration process. The Media API has been used extensively to implement several media drivers in CAIRO like the Audio and Video conferencing, the Three Dimensional meeting space and also a Speech recognition module. Analysis has also been conducted in the area of Computer Telephony Integration and the implementation thereof as a driver in CAIRO. This chapter has dealt with describing briefly about the various media capabilities provided by the Java Media APIs. The following chapter describes several other multimedia based collaborative applications that are available in today's market and makes an in-depth comparison of the features offered by each one of them with the features offered by the CAIRO system.

Chapter 4

4.0 Literature Review

In this section, several other multimedia-based collaboration systems are examined. Each of the systems being analyzed has been described is short. Following that, an in-depth comparison of these systems with CAIRO has been presented.

4.1 Multimedia-based Collaboration Systems

Multimedia-based Collaboration Systems are systems that have media tools assisting the collaboration process. Media tools could be implemented in the form of whiteboards, audio/video, telephony or three-dimensional meeting environments.

4.1.1 Multimedia Collaboration (MMC)

MMC [MMC 1996] is a video conferencing system developed at the Technical University of Berlin, which allows several people to talk to and see one another over a computer network. The conference participants can also work together using computer applications. MMC runs on a wide variety of different platforms.

4.1.1.1 CSCW

MMC is one of the *Computer Supported Collaborative Work* (CSCW) applications. The main objective of CSCW is to provide work group applications, which require multi-user application access and control, and coordination of all users' activities.

In the CSCW research, two main approaches may be identified:

Collaborative work is based on dedicated, collaboration-aware applications (e.g. joint editors), which distribute themselves among a group of users.

Basic mechanisms are provided to distribute off-the-shelf applications for the collaborative work.

Both solutions have their advantages and disadvantages, however in the MMC project the general approach was chosen.

4.1.1.1.1 Audiovisual Communication

MMC uses high-speed networking technology and video hardware, together with sophisticated audiovisual management among the partners and dedicated multimedia communication protocols. For using MMC in an environment with only limited bandwidth (e.g. ISDN connection between the conference participants), appropriate audio and video coding standards are supported too.

4.1.1.1.2 Application Sharing

Most CSCW solutions are based on the distribution of the view and control of a single application among several users. This implies a similar (and, for some special applications, identical) presentation of the application's output to all users, and a consistent application's state by input coordination among all users. This common view and control of the application is commonly called *application sharing.*

In MMC, any application running on the local window system may be selected for application sharing. This is achieved by intercepting the communication between the

application and the window system, together with distributing the input/output of the application to other, remote window systems.

4.1.1.1.3 Group Management

Building up and administering a workgroup is very similar to generic conference management. This includes conference creation, the extension by joining users, and the reduction by users who leave the conference. Several actions in a conference may be combined with certain roles, e.g. there may be observers who may track the course of a conference, but are not allowed to take active parts. Another requirement is that only one privileged user (the chair) may exclude participants or terminate the whole conferencing session. Access to a conference has to be controllable to allow sessions for closed user groups, and there must be a mechanism provided to inform others about ongoing and projected conferences. Last but not least, there has to be some kind of directory, where needed information about all users may be found. MMC comprises all functionality to meet those requirements for conference management.

4.1.1.2 MMC Services

The actual MMC service provides the following functionality:

4.1.1.2.1 Audio

- Support of the G.711/16 kHz Mulaw/16kHz 16 bit linear Stereo
- Adjustable send volume
- Speaker or headphone output support
- Audio connection between any numbers of conference participants

4.1.1.2.2 Video

- Support of the M-JPEG video format

- Video connection between three conference participants
- Support of H.261 video format

4.1.1.2.3 Application Sharing

- Realtime application sharing support of off-the-shelf applications
- Different access modes to the shared application (token mode/free mode)
- Telemarker support

4.1.1.2.4 Conference and Audio/Video Management

- Managing several conferences by the same backbone components
- Central data base for all MMC relevant information about conference participants and participants groups
- Granting/restricting the conference access to a group of participants
- Sending/receiving any invitations with specification
- Specifying the subject and any annotations within the invitation
- Inviting participants during an ongoing conference
- Different role assignment for the participants of the conference (Chair, Potential Chair, Active, Observer)
- Transferring the chairmanship during the ongoing conference
- Excluding any participant from the conference by the actual chair
- Support of different Conferences Modes
- Support of different Audio/Video Modes

- Support of different Conference Directory Modes

4.1.2 RemoteVideo

RemoteVideo [RVideo 2000] is a networked communication solution with strong software and durable, high-class equipment. It offers powerful multimedia communication service in business-to-business applications.

4.1.2.1 RemoteVideo Features

RemoteVideo delivers real time digital video, audio and data connections for sharing information in any global corporate co-operation as well as excellent remote service and surveillance support. It can reach remote places with wired, wireless or satellite connection. RemoteVideo supports both world standards for managing communication links, H.323 for TCP/IP and H.320 for ISDN. This makes RemoteVideo^{TM} a perfect match for any infrastructure environment and with any set of communication links in the foreseeable future. RemoteVideo also helps in overcoming language barriers in global business operations support. It can even save personnel from physical threats in providing remote service to dangerous locations. RemoteVideo has been tried and accepted in demanding industrial operations in remote and harsh conditions. It is used in traffic and industrial process surveillance, as well as on board of the ships in middle of the ocean and in desert power plants. It is a tool for world class companies on-line remote service and customer support.

4.1.2.2 RemoteVideo Workstation

RemoteVideo Workstation is the ground product solution of RemoteVideo product suite, sharing all the general benefits of RemoteVideo. It is designed for a wide range of business communication service situations also in remote locations. It delivers real time digital video, audio and data connections for sharing information in remote surveillance, service and support operations, as well as in any global corporate or other group co-operation.

RemoteVideo Workstation serves in internal, partner or customer co-operation and extended, knowledge intensive key customer service like R&D and technology support. It offers handy sales and marketing support as a remote demonstration tool.

RemoteVideo Workstation supports international standards for multimedia teleconferencing, which is essential for seamless wide-area communication.

4.1.3 The VTEL Workgroup Systems

The VTEL [VTEL 1998] Workgroup Systems products are designed for geographically dispersed workgroups who need to share, create and modify information. Project teams sharing and working on collaborative data will find the WG500™ workgroup systems an excellent platform for interactive tasks and intra-team communications. These products are based on a PC hardware platform offering:

High quality video and audio images that enhance human interaction over distances

Motion handling and audio is smoother and more natural than the jerky video images associated with low-end videoconferencing products

High performance data collaboration tools allow users to display, modify and save detailed project plans, spreadsheets, presentations, flowcharts or other PC applications while conducting the video meeting

Standards compliance offers T.120, the international standard for sharing data across videoconferencing product platforms and integrates Microsoft NetMeeting, the industry leader for standards-based collaboration

Consistent user interface with VTEL's AppsView graphical conference-control tools - once users master the operation of any VTEL system, they can easily operate the entire product line without additional training

Network flexibility that will support customers' network connectivity of choice - ISDN and dedicated networks

4.1.3.2 WG500 Key Features

4.1.3.2.1 Audio/Video

- Up to 800 x 600 resolution full-screen video
- Separate local and remote video windows
- Still-image screen capture

4.1.3.2.2 Collaboration

- T.120 compliant
- T.127 Drag-and-drop file transfer
- Microsoft NetMeeting software enables application sharing and interactive whiteboard

4.1.3.2.3 Connectivity

- Videoconferencing over LAN or WAN (ISDN/Dedicated) on a call-by-call basis
- Dial-up networking to LAN, WAN or Internet, via ISDN modem connection (128i model only)
- Call blocking and call screening, where ISDN caller ID is available
- 10/100 Ethernet NIC for data

4.1.3.2.4 PC Versatility

- Full Windows 98/NT 4.0 functionality
- Ethernet NIC allows LAN, WAN and Internet access
- Open slots for additional expansion

4.2 Desktop Video-conferencing

Video-conferencing [Mong Le 1998] describes the use of compressed video technology for live, two-way, interactive communication in a variety of situations - person to person, informal discussions, formal group meeting, and large lectures. The primary use of video conferencing is to allow the timely exchange of information without traveling. Most meetings that are held face-to face can be held by video-conferencing. Video-conferencing is useful in situations of crisis management, or times when meetings are held on short notice.

Desktop video-conferencing (DVC) is an emerging technology of video-conferencing. A DVC system uses a standard personal computer (PC) enhanced with special video processing capabilities and a small video camera with a speakerphone.

A DVC system allows two or more people using their PCs from different places to communicate through an audio and video connection. Connections between DVC systems can be made over Integrated Services Digital Network (ISDN) lines through regional and long distance telephone companies. A DVC system comes in three basic types: those that operate over phone lines, such as a 56 Kbps WAN/ T1 telephone service and ISDN, those that operate over a LAN such as Ethernet, and dedicated systems that use special video cables between each connected computer. This paper provides an overview of desktop video-conferencing and the feasibility of implementing DVC running on a 56 Kbps wide area network (WAN) or T1 telephone systems, and ISDN, with a short discussion on these running on local area networks (LANs).

4.2.1 Desktop Video-Conferencing Systems Overview

Desktop video-conferencing is gaining acceptance as a viable telecommunications technology. While a video-conferencing system uses the analog voice and video technology, a DVC system employs the digital audio and video technology offered real-time applications sharing that increases productivity of groups who need to work together

but are split up geographically. Designed to run on a personal computer (PC)-based platform, DVC systems are meeting the challenge of providing varied applications in the government and the commercial sectors. DVC systems take advantage of the windows multi-tasking environment, allowing users to maintain a live face-to-face video-conference while accessing many other applications on their desktops, such as spreadsheet and word processing.

In general, a DVC system has three basic parts to the screen: the video window, collaborative workspace and "Hollywood" squares. Conferees at sites are able to manipulate images, annotate text, etc., in real time. A conferee can take "snapshots" of information and send it as an image or send information in a file, while the video portion and audio connection stay active. Using a VCR, a desktop video-conference can be recorded for future use. DVC systems, either analog or digital, can be used by organizations as an enterprise-wide solution. An analog system can provide voice, real color images, and graphics at an analog bandwidth up to 4.5 MHz. This system uses digital PC White Board Windows applications that can operate on either a LAN or WAN through switches, DSU/CSU/ Modems or codec for hands-on interaction between a point-to-point or a multi-point conference. A digital DVC system offers more advanced features as well as provides more flexibility than an analog based-system. A digital system allows voice, vivid compressed color images, graphics, spreadsheets or any data file to be transmitted between remote locations within seconds.

According to telecommunications experts, the DVC technology is expected to be commonplace in 1998. (Network World, 10/95, Tackett). Programmable multimedia processors with video-conferencing capabilities now allow high performance achieved with low cost add-in boards for PC's (Waurzyniak, 1995). Additionally, many DVC systems are developed to provide an easier access for more users through the use of LANs.

4.2.2 Standards

Most current DVC systems use H.320 as an umbrella standard that embraces several other standards detailing how video and audio information is compressed and transmitted

over wide-area digital services. In addition to H.320, there are a number of video and audio standards widely used for DVC systems. Some as discussed below.

- Video Standard: H.261, also known as the ITU Px64, which provides the specification for compressing real-time video at rates varying from 56/64 Kbps to 1.92 Mbps. H.261 covers both P x 56/64 Kbps (P = 1,2, ... 6) and M x 384 Kbps (M= 1,2,...5). The compressed video comes in two video quality modes (display formats), Common Intermediate Format (CIF) and Quarter-CIF (QCIF). The CIF provides for a 288 lines by 352 pixel picture, and the QCIF a 144 line by 176 pixel picture. Theoretically, a CIF and a QCIF can transmit video at 30 frames per second (fps) and 5 fps, respectively. But except for dedicated wire systems, as of today, no DVC system transfers video close to 30fps over ISDN. The use of an ITU P x 64 standard-based desktop video-conferencing provides (1) interoperable DVC system; (2) multiple procurement sources facilitating competitive pricing; (3) increase product life cycle; (4) economical long term investment.

- Video Framing Standard: The ITU H.221 standard is a framing protocol used for multiplexing video, audio, and control signals. It has the capability of aggregating 56/64 Kbps bandwidths to form a wider bandwidth.

- Voice Standard: The ITU P x 64 recommendations support several audio standards. G.711 describes audio transport at 64Kbps pulse code modulation (PCM) digital audio from 3.5 KHz analog, G.722 uses sub-band adaptive differential pulse code modulation (ADPCM) to provide a 48Kbps or a 56Kbps at 7KHz analog signal (almost CD quality), and G.728 provides for near-PCM quality audio at a data rate of 16 Kbps at 3KHZ for low bit rate video.

- Point-to-Point Control: Point-to-point conference control is facilitated by the use of the ITU H.230 and H.242 standards. The ITU H.230 standard provides control and signaling capability. The ITU H.242 standard provides for the set up and disconnnect; inband information exchange, fault and channel management.

- Multi-Point Conference Control: The ITU P x 64 standards associated with multi-point conference control (MCC) are H.231, and H.243. The ITU H.231 standard

describes the overall architecture of multi-point control unit (MCU) using digital channels up to 2 Mbps. The ITU H.243 standard provides the control procedures between a ITU H.231 compliant MCU and ITU H.320 codecs. Some of the MCU services include (a) meet-me conferences, where conferees get a number in advance that will connect them to the conference; (b) dial-out conferences, where the MCU calls each conferee to initiate the conference; (c) progressive conferences, where a conferee calls successive parties and adds them to the conference one at a time; (d) Reservation systems; (e) Remote Maintenance; (f)Attendant screening of conferees for added security.

- T.120 Standard: DVC uses T.120 as an umbrella standard for data sharing. This standard, which was to have been finalized last summer,allows for the sharing of information such as whiteboard, file transfer and so on between systems that conform to this standard even if the systems are not alike.

Many other video and audio standards for DVC are in the work. Some of them include the transport of video and audio over the Internet (H.323), or company network and H.324 for direct modem-to-modem dial-up link, gateways for ISO-Ethernet, and high-speed Ethernet-to-ISDN and mobile communications. Intel also developed its Indeo, a video compression technique, to compete with H.320 and others DVC standards.

4.2.3 Hardware and Software

In addition to requring a powerful PC with a high resolution monitor, a DVC system requires a pecific video-conferencing card, a color video camera, and a speakerphone. These will allow a DVC system to receive and send digitized audio, video, and text over LANs, WANs, or ISDN phone lines. The DVC software allows a user to place and answer video phone calls, control video, audio, and text sent and received. Some versions of DVC software also support collaboration which provides for a shared software window, a shared clipboard, notes and a chalkboard.

4.2.4 Local Area Network and Internet Desktop Video-Conferencing

A Local Area network (LAs) was designed to connect many PCs together. At the physical layer, a LAN usually consists of 10 Mbps Ethernet, or 4 to 16 Mbps Token Ring segments. Today, various DVC products support a variety of LAN protocols such as TCP/IP, Novel IPX/SPX, NetBIOS, and Aplletalk. To effectively use a LAN to transport video information, appropriate bandwidth is needed on that LAN. Every attempt must be made to avoid situations where the video applications have to contend with data applications for bandwidth. Video-conferencing is a bandwidth intensive application. Significant usage of desktop video-conferencing in a LAN is anticipated for both point-to-point and multipoint applications.

Since the current Ethernet network was not designed for streaming audio and video traffic, when a DVC system is implement on an existing Ethernet LAN, bandwidth may be a problem. The new ISDN-ISO 10BaseT Ethernet network technology can make DVC less stressful. This new technology allows the 10BaseT Ethernet and ISDN signals to be separated, so connecting video over the ISDN channels has no effect on the Ethernet. The fast Ethernet technology (100BaseT or 100VG-AnyLAN), which provides more bandwidth, can also make a DVC system to operate easier on a LAN. Finally, Asynchronous Transfer Mode (ATM) will eventually be the standard for transporting multimedia communication in both the local and wide areas.

Today, local distribution of the video-conferencing information over a LAN is technically feasible but is not recommended. Under heavy loading conditions such as the transport of video, the LAN would become overloaded, with the eventual result being a significant degradation in the quality of service with the possibility of total system failure. Today, the Internet can connect various LANs together. Various DVC applications that operate over the Internet primarily use User Datagram Protocol (UDP) for video and audio data transmission over the Internet.

4.3 CAIRO Multimedia Component

Having seen several multimedia tools available in the market in section 4.1 and having discussed the standards of Digital Video Conferencing systems in section 4.2, this section now compares CAIRO as a Multimedia based collaboration tool with respect to the above two sections.

CAIRO has three main multimedia components as described in Chapter 3. They are:

- 3D Virtual Environment
- Audio/Video
- Speech Recognition

4.3.1 3D Virtual Environment

- Supports different styles of meetings like the chairman, freestyle, lecture and the classroom
- Supports multiple simultaneous meetings, and also side talks
- Has different 3D human forms for users to select from

4.3.2 Audio

- Support of a wide range of formats as described in table 5-2
- Adjustable send volume
- Speaker or headphone output support
- Audio connection between any numbers of conference participants
- Audio screening for participants in private conversations

4.3.3 Video

- Support of a wide range of video formats as described in table 5-1
- Video connection between multiple conference participants
- Video Screening for participants in private conversations

4.3.4 Application Sharing

- Realtime application sharing support of off-the-shelf applications
- Different access modes to the shared application

Flexible audiovisual connection management for providing best quality audiovisual links between the conference participants

4.4 Conclusion

In this chapter, CAIRO has been compared with several other multi-media based collaborative applications. The following chapter deals with the details of how each of the components of the Media API have been used in the implementation of the various components of CAIRO's media infrastructure.

Chapter 5

5.0 Implementation

As described in Chapter 3, Java Technology was used for most of the implementation. The Java Media Packeges were used to develop the additional features such as the 3D Meeting environment, the audio and video conferencing and the speech to text. The components that were implemented using the Java Media packages are described in the following sections.

5.1 Three Dimensional Meeting environment

This section deals with the design of the three-dimensional meeting environment that has been created with the use of VRML 2.0. and also the Java3D API [Java3D 1999].

5.1.1 Definitions and Acronyms

APPEARANCE: A description of the coloration of a shape. Appearance is described by an Appearance node type. [see NODE, pg. 61]

AVATAR: In a VRML environment an AVATAR is the visual "handle" or display appearance used to represent a user [In the Hindu religion, an avatar is an incarnation of a deity; hence, an embodiment or manifestation of an idea or greater reality].

AWARENESS: The awareness driver adds affective computing functionality to CAIRO 3.0. The module will detect physiological changes in the meeting participants and then be sent to the Affective Pattern Recognizer which would detect the emotions being represented by the signals. These emotions would then be displayed to the other participants through the 3-D Avatars (Moods) and the Affective GUI Widgets(Arousal & Valence).

CAIRO: Collaborative Agent Interaction and Synchronization

EVENT: A message sent from one node to another along an animation circuit route. Every event contains a value, with a data type, and a time-stamp. [see eventIn, eventOut, and route]

EAI: External Authoring Interface, the Java API that can be used by an applet to control a VRML browser and its contents.

EVENTIN: An input to a node used when wiring a route for an animation circuit. An eventIn has a name and a data type. [see event, eventOut, node, and route]

EVENTOUT: An output from a node used when wiring a route for an animation circuit. An eventOut has a name and a data type. [see event, eventIn, field, node, and route]

FIELD: A node parameter that provides a shape dimension, color, or other form of node attribute. A field has a field name, a field data type, and a value.

GUI: Graphical User Interface.

NODE: A basic building-block used in VRML 2.0 world-building instructions. A VRML 2.0 file always has at least one node in it, and often contains hundreds or even thousands of nodes. Individual nodes build shapes, describe appearance, control animation, etc.

ROUTE: A connection between an eventOut of one node and an eventIn of another. Routes form the wires of an animation circuit. Event values flow along a route, from eventOut to eventIn. [see event, eventIn, and eventOut]

SCENE GRAPH: A family tree of coordinate systems and shapes that collectively describe a VRML world.

SENSOR: A node type that senses a change in the environment. Typical sensor nodes sense the passage of time, movement of the user's cursor, the press of the user's mouse button, the user's proximity, collision of the user with a shape, and so forth.

VRML: An acronym for *Virtual Reality Modeling Language*. VRML is a rich text language for the description of 3-D interactive worlds. [see VRML browser and world-builder]

VRML BROWSER: A stand-alone helper application or Web browser plug-in that displays VRML worlds. [see world-builder]

WORLD-BUILDER: An application that enables VRML world authoring within an interactive 3-D drawing environment.

5.1.2 Use Case

The following paragraphs provide a typical interaction between a user and the VRML interface, called a use case, describing the general functionality of the three-dimensional environment.

When a user logs into CAIRO and starts the VRML driver a visual "handle" or representation of the person is placed in the initial VRML space, which is the hallway. The user will see the hallway and the doors to the meeting room but the user will not be able to see himself or herself. Once the user decides which room he or she will enter, the avatar will automatically be lead through the hallway to the room. If the door is closed it will swing open so that the person can enter.

Once the user goes through the door frame the avatar will be guided to a default chair. Supposing the room has a chairman then there will be a rectangular table with chairs on both sides. People may already be seated in the meeting and each will be able to direct his camera to any part of the room, but if the camera is facing forward, the lens angle will be

wide enough to see all the other members in one view. The other participants will be able to see the person's avatar walk to the chosen chair.

During the meeting if the user wishes to hold a private conversation with another person in the forum, another canvas will open showing a new room, called private room, containing a desk several chairs, where the participants will be seated. Once the private session is over the canvas will be destroyed, returning the focus of attention to the original meeting room.

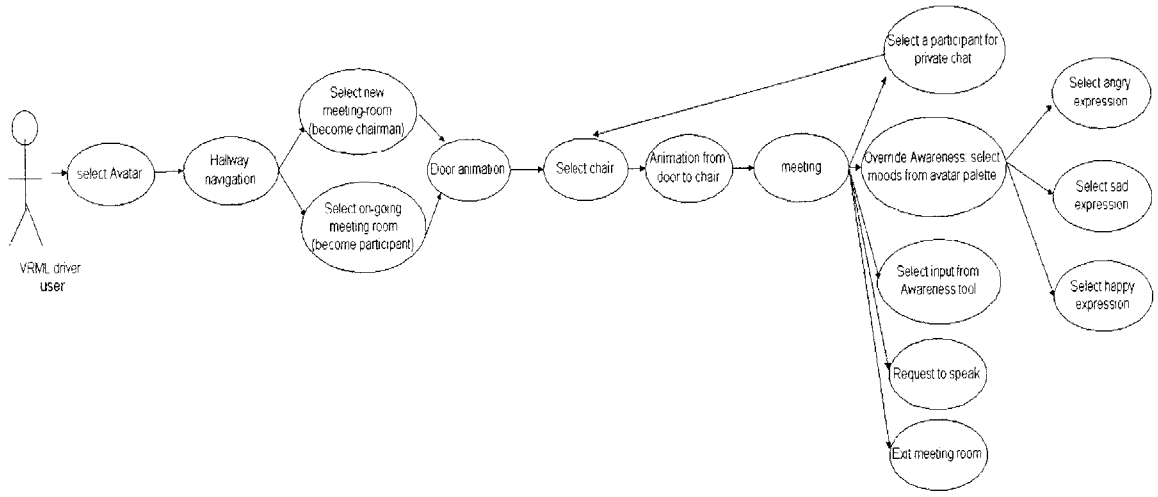

Figure 5-1. Use Case Diagram for the VRML environment.

5.1.3 VRML Execution Model

VRML consists of two fundamental parts: The scene description language and a mechanism to create behavior. Behavior might be anything that dynamically changes the properties of the scene. There could be something as simple as a box to a complex artificial intelligence avatar in a virtual world.

5.1.3.1 Events

When one node wants to pass information to another node, it creates an event. An event is something between a function call and an assignment in a programming language. The

event contains the *data* and the *timestamp*. The timestamp is an important concept in cases where networking is involved, as is the case here. It is the time at which the event was originally started. This helps the browser to keep track of the sequence of events in such an environment.

5.1.3.2 Routes

VRML does not have a function call mechanism for passing information. Instead it involves the creation of an explicit connection between two fields of nodes. There are four different types of access types that using which the behavior of the nodes can be controlled.

5.1.3.3 Sensors

Unlike Java where the programs throw events, in VRML we have to generate the events and pass them around. We do this once we have a series of nodes connected together by routes. The sensor group of nodes, sense some sort of non-VRML input and generate a VRML event which is then passed around the scene. There are three basic types of sensors namely *user input, visibility* and *time* sensors. The Time sensors and the user input sensors are very important for the product that is being developed. The Touch sensors are a kind of user input sensors which are very important to define a boundary for the environment, that restricts objects(avatars) from navigating out of the environment. For instance, using the touch sensor, we could introduce a condition by which the avatar does not leave its chair once it is seated on it, until the user logs out of the meeting.

5.1.3.4 Building an Animation

The building of the animation involves the existance if the basic scene layout. It also consists of the object to be animated and a reference point with respect to which the animation is done. There will be a set of transforms that define the animation to be executed on the object of interest. Then we need to layout the path that the object will take which is done using a set of interpolations. There are different kinds of interpolations based on the type of animation that is being done. Finally we need to provide a time sensor to drive the animation. A few routes will also need to be added at

the end to pass events around. The logging in of the users involves animation. It comprises of the animated view of the avatar entering the room and taking its seat. Similarly the logging out of the user also involves animation of the avatar leaving the room. There could also be a few small animation during the meeting process, like looking around, shrugging and waving of hands.

5.1.4 VRML Environment Configuration

The existing 2-D interface of CAIRO v2.0 was taken as the starting point for the design of the new VRML environment of v3.0. The new environment tries to mirror the configuration and functionality available in the current version of CAIRO. However, the 2-D GUI will not be removed.

The 3-D environment can be divided into three major components according to the requirements for each one of these components: the hallway or common space (i.e., navigation), the meeting rooms (i.e., environment), and the objects inside each meeting room (e.g., AVATARS and furniture).

5.1.4.1. Hallway

The starting point for the user will be a waiting room or hallway with doorways leading into the meeting rooms. As in the 2-D counterpart, the doors will have visual cues to indicate the three possible states that the room is currently in:

1) *Open door*: room available.
2) *Closed door*: session in progress.
3) *Padlocked door*: private meeting.

The purpose of this hallway is to give a 3-D analogy of a menu from which to choose to join a meeting or create a new one. Also the hallway will show through the open doors, the different types of meeting rooms available. The main issue for the hallway is navigation. Allowing the users to manually maneuver through the hallway to the room and then to the seat may unnecessarily complicate the procedure involved. So the user

will be limited to a main preprogrammed path that leads the user to the room and then the seat. Upon entering the room and choosing a seat, the user's movement will be restricted to a fixed set, comprising of looking left, right, up and down. When the user logs out, similar to the login procedure, there will be a preset path that will lead the user from his seat, out of the room and to the hallway. The purpose of these predetermined animations is to simplify and shorten the time required for navigating the space, leaving the user to concentrate on attending or conducting the meeting. In this way the new 3-D interface will support CAIRO's functionality avoiding confusion in navigation issues.

5.1.4.2 Meeting Rooms

There will be four types of rooms implemented in the VRML environment, each representing the different style of meetings that can be conducted in CAIRO. The floor plan configurations are listed in Figure 2. The four types of rooms will differ to accommodate the following types of meeting:

1) Freestyle
2) Chairman
3) Lecture
4) Private
5) Lounge

Freestyle (Roundtable): For group discussion, small group interaction that involves all participants. There is no facilitator.

Chairman (Conference style): when the discussion is guided by one participant, who gives the floor to the speakers. The focus of attention is on the chairman at the end of the table. It provides a more formal setting than the circle.

Lecture (Theater style): For more passive meetings, where a person has the floor for most of the time.

Private (2 chairs-and-a-desk-style): when two users need to have a confidential one-on-one discussion.

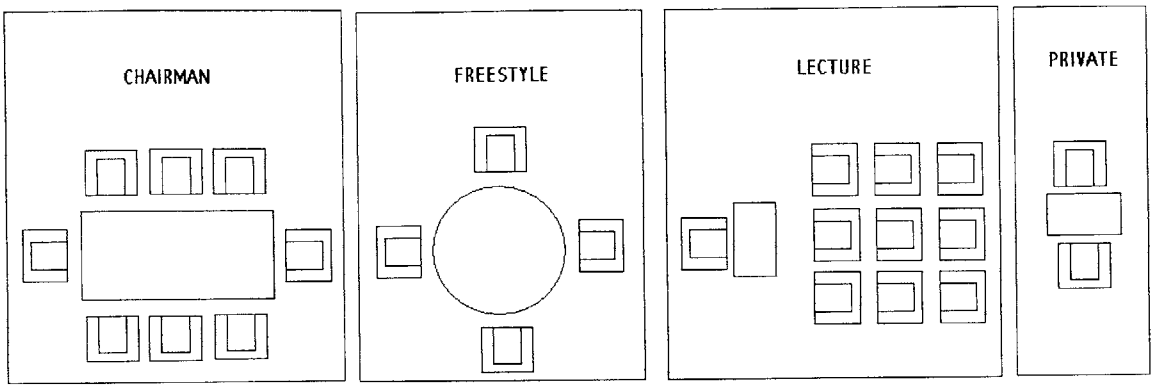

Figure 5-2. Types of Meeting Rooms [Solari et. al. 1999].

Unlike the Hallway, for the meeting rooms the navigation will be completely predetermined, the users will be guided upon entry and exit from the door to the chair that is selected, as explained in the use case. The only difference is that here the users have no options to choose from (except for chair location) therefore total restriction in the degrees of freedom was chosen. The reasons for this design decision are the same as for the Hallway.

5.1.4.3. Components

The third level of objects that compose the VRML environment are the small-scale components:

- Furniture
  - Chairs
  - Table
  - Whiteboard
  - Phone
  - Television
- Cameras
- Avatars

The components of the meeting environment can also be categorized as inanimate (e.g., chairs and tables) and animated (e.g., doors swinging and people). The furniture (e.g., chairs and tables) will reinforce the purpose of each room. Because of their static nature, these objects do not require to be animated, except for the doors. Each piece of furniture is associated with a specific functionality, where the functionality of the chairs and whiteboard will be implemented the end of the Spring working period.

There will also be one camera for each person sitting in a meeting. The cameras will be still for the duration of the meeting and follow a preprogrammed path to and from the door at the beginning and end of each session.

There will be six different configurations for people in the meeting room. They will be differentiated by sex, ethnicity, age, face and clothes. Therefore there will be two basic body types and six different faces and clothes for each person. The location of the Avatars during the meeting, like the cameras, will remain at each seat during the duration of the meeting and follow a path to and from the door at the beginning and end of each session that the other users will be able to see from the viewpoint of their respective cameras.

Each person will have a palette of facial expressions to choose from than can override the moods that are being received from the *AWARENESS* driver at any given time. These expressions (Moods) will be the same kind and number as in the Affective GUI Widgets. The behaviors for each Avatar will show the different positions each person can adopt: sitting, standing and walking.

5.1.4.4. The Architecture of the 3D Environment

A view of the general model of the three-dimensional environment is shown in Figure 5-3. Each entity is given a number of attributes that it can possess and can take on.

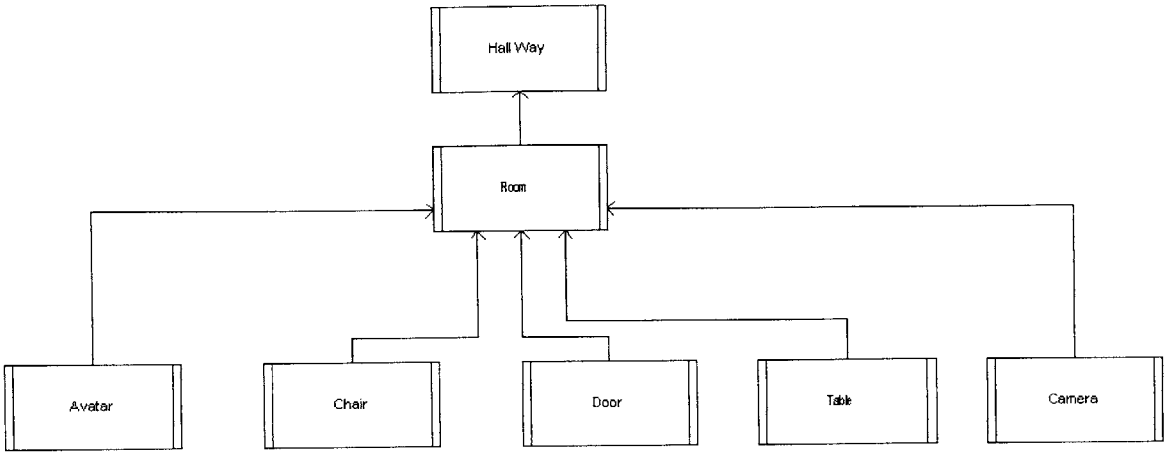

Figure 5-3. UML Diagram: The 3D Environment Architecture [Solari et. al. 1999].

5.1.5 The VRML Browser Applet

The VRML interface that we will be developing here will be a basic browser canvas. It will not have navigation buttons like translation, planning, rotation, rolling like in the case of regular VRML browsers. This is partly because we would like to restrict any kind of motion once the user logs on to the meeting and the avatar takes its seat. There are a few basic operations though that the browser applet will need to have:

- Ability to load a new VRML file
- Ability to remove objects and add new VRML objects to the existing scene
- Interact with the Multiuser Interface(Section 5.1.6)

Some of the function calls that the browser will have are *CreateVrmlFromURL, ReplaceWorld, TouchSensor, LoadURL, eventsProcessed.*

VRML browser
CreateVrmlFromURL
ReplaceWorld
TouchSensor
LoadURL
EventsProcessed

Figure 5-4. Object Model of the VRML Browser [Solari et. al. 1999].

5.1.5.1. The VRML Script Node

The Script node has a very different functionaliy from the other VRML nodes. When we wish to create our own behavior, we need to first define the fields that need to access the script nodes. This is done in the VRML file. Then we need to create the actual scripted code itself. There are other issues like defining basic Script node Fields and script execution which is beyond the scope of this document.

5.1.5.2 Interacting with the Browser

As defined in the object model diagram (Fig. 5-4) above, the Browser will have a set of functions that load and modify VRML files on the scene. The object model of the browser shown, shows the different functions that the browser will be using. The actual architecture of the browser applet will be as follows:

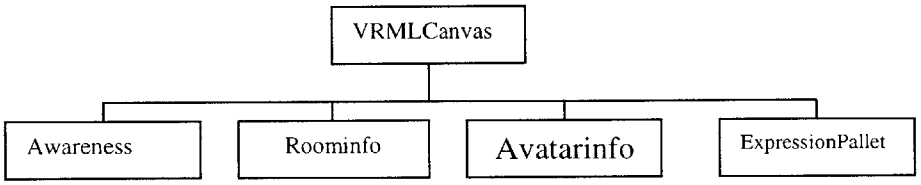

Figure 5-5. Class Diagram of the VRML Browser implementation [Solari et. al. 1999].

The **VRMLCanvas** is the main object on which the scene is loaded.

The **Avatarinfo** class takes care of all the avatars in the room. It keeps track of the number of avatars in the room and the information about them. It is responsible for the animations that take place during logging on and logging out process. It also controls the avatars during the meeting process, by taking the appropriate signals for the expressions and passing the events around the scene.

The **Roominfo** class keeps track of the room events. It changes the rooms when there is a change in meeting style.

The **Awareness** class sends appropriate signals to the Avatarinfo class based on the signal it gets from the awareness tool.

The **ExpressionPallet** is a pallet that has different expressions on it and the user can change the expression on the avatar corresponding to him in the meeting environment by clicking on the expression he wants to.

5.1.6 The Multiuser Technology

The "multiuser technology" provides us with a n interface between the VRML browser and the underlying communications network, which will be the internet in our case. Underneath the MUTech lies the network protocol stack; in the case of the internet, this consists of the UDP and the TCP running on top of the IP.Above the MU Tech lies the application/applet itself and the multiuser API lies at this layer.

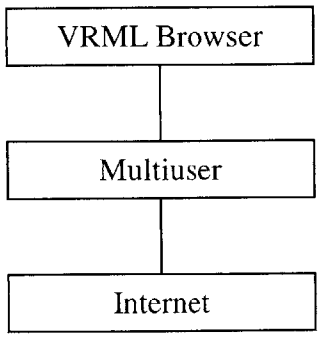

Figure 5-6. Basic Components of the system.

5.1.7 The overall Architecture

The architecture of the system as a whole would be an applet that loads the VRML file of the room when a meeting is initiated. As people logon to the meeting the applet loads avatars representing the people, these avatars sit in the seats in the room as long as the meeting is in session. When a person logs out the browser shows the corresponding avatar leaving the room. When the meeting style changes, the browser loads a different room environment. We also have a Registry host that keeps track of the entities in the environment. About the implementation issues, we could use either the multiuser API available or the already existing Networking classes in CAIRO.

But the basic architecture of the Client and the server programs is as follows:

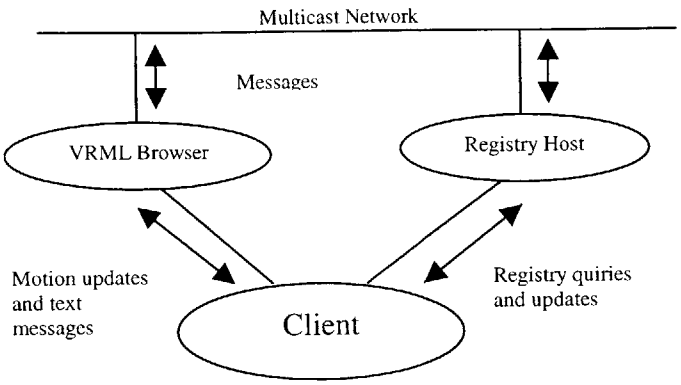

Figure 5-7. Basic System Components of the Client side [Solari et. al. 1999].

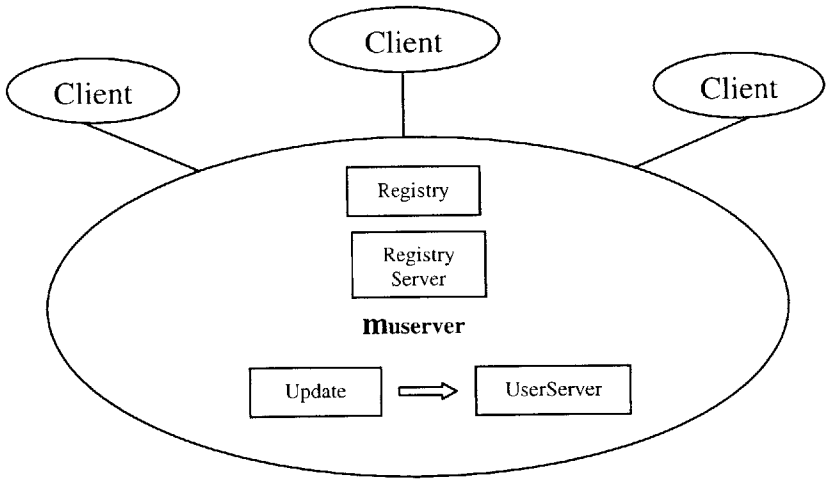

Figure 5-8. Basic System components of the server side [Solari et. al. 1999].

5.2 Audio/Video Conferencing

The Java Media Framework Package was used for the implementation of the Audio/Video conferencing component. The Java Media Framework (JMF) [JMF 1999] is an application programming interface (API) for incorporating time-based media into Java applications and applets.

5.2.1 Working with Time-Based Media

Any data that changes meaningfully with respect to time can be characterized as time-based media. Audio clips, MIDI sequences, movie clips, and animations are common forms of time-based media. Such media data can be obtained from a variety of sources, such as local or network files, cameras, microphones, and live broadcasts.

This section describes the key characteristics of time-based media and describes the use of time-based media in terms of a fundamental data processing model:

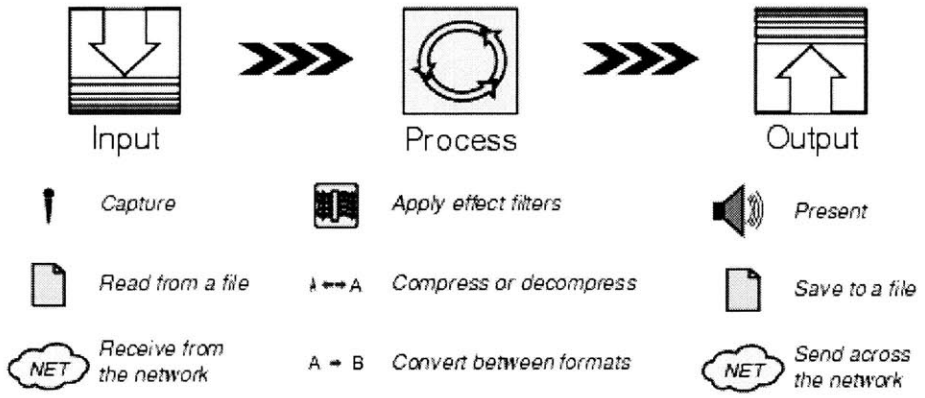

Figure 5-9: Media processing model [JMF 1999].

5.2.1.1 Streaming Media

A key characteristic of time-based media is that it requires timely delivery and processing. Once the flow of media data begins, there are strict timing deadlines that must be met, both in terms of receiving and presenting the data. For this reason, time-based media is often referred to as *streaming media*--it is delivered in a steady stream that must be received and processed within a particular timeframe to produce acceptable results.

For example, when a movie is played, if the media data cannot be delivered quickly enough, there might be odd pauses and delays in playback. On the other hand, if the data cannot be received and processed quickly enough, the movie might appear jumpy as data

is lost or frames are intentionally dropped in an attempt to maintain the proper playback rate.

5.2.1.2 Content Type

The format in which the media data is stored is referred to as its *content type*. QuickTime, MPEG, and WAV are all examples of content types. Content type is essentially synonymous with file type--content type is used because media data is often acquired from sources other than local files.

5.2.1.3 Media Streams

A *media stream* is the media data obtained from a local file, acquired over the network, or captured from a camera or microphone. Media streams often contain multiple channels of data called *tracks*. For example, a Quicktime file might contain both an audio track and a video track. Media streams that contain multiple tracks are often referred to as *multiplexed* or *complex* media streams. *Demultiplexing* is the process of extracting individual tracks from a complex media stream.

A track's *type* identifies the kind of data it contains, such as audio or video. The *format* of a track defines how the data for the track is structured.

A media stream can be identified by its location and the protocol used to access it. For example, a URL might be used to describe the location of a QuickTime file on a local or remote system. If the file is local, it can be accessed through the FILE protocol. On the other hand, if it's on a web server, the file can be accessed through the HTTP protocol. A *media locator* provides a way to identify the location of a media stream when a URL can't be used.

Media streams can be categorized according to how the data is delivered:

- Pull--data transfer is initiated and controlled from the client side. For example, Hypertext Transfer Protocol (HTTP) and FILE are pull protocols.

- Push--the server initiates data transfer and controls the flow of data. For example, Real-time Transport Protocol (RTP) is a push protocol used for streaming media. Similarly, the SGI MediaBase protocol is a push protocol used for video-on-demand (VOD).

5.2.1.4 Common Media Formats

The following tables identify some of the characteristics of common media formats. When selecting a format, it's important to take into account the characteristics of the format, the target environment, and the expectations of the intended audience. For example, delivering media content via the web, needs special attention to the bandwidth requirements.

The CPU Requirements column characterizes the processing power necessary for optimal presentation of the specified format. The Bandwidth Requirements column characterizes the transmission speeds necessary to send or receive data quickly enough for optimal presentation. Some formats are designed with particular applications and requirements in mind. High-quality, high-bandwidth formats are generally targeted toward CD-ROM or local storage applications. H.261 and H.263 are generally used for video conferencing applications and are optimized for video where there's not a lot of action. Similarly, G.723 is typically used to produce low bit-rate speech for telephony applications.

5.2.1.5 Media Presentation

Most time-based media is audio or video data that can be presented through output devices such as speakers and monitors. Such devices are the most common *destination* for media data output. Media streams can also be sent to other destinations--for example, saved to a file or transmitted across the network. An output destination for media data is sometimes referred to as a *data sink.*

Table 5-1. Common video formats.

Format	Content Type	Quality	CPU Requirements	Bandwidth Requirements
Cinepak	AVI QuickTime	Medium	Low	High
MPEG-1	MPEG	High	High	High
H.261	AVI, RTP	Low	Medium	Medium
H.263	QuickTime AVI, RTP	Medium	Medium	Low
JPEG	QuickTime AVI, RTP	High	High	High
Indeo	QuickTime AVI	Medium	Medium	Medium

Table 5-2. Common audio formats.

Format	Content Type	Quality	CPU Requirements	Bandwidth Requirements
PCM	AVI QuickTime WAV	High	Low	High
Mu-Law	AVI QuickTime WAV RTP	Low	Low	High
ADPCM(DVI,IMA4)	AVI QuickTime WAV RTP	Medium	Medium	Medium
MPEG-1	MPEG	High	High	High
MPEG Layer3	MPEG	High	High	Medium
GSM	WAV RTP	Low	Low	Low
G.723.1	WAV RTP	Medium	Medium	Low

5.2.1.6 Presentation Controls

While a media stream is being presented, VCR-style presentation controls are often provided to enable the user to control playback. For example, a control panel for a movie player might offer buttons for stopping, starting, fast-forwarding, and rewinding the movie.

5.2.1.7 Latency

In many cases, particularly when presenting a media stream that resides on the network, the presentation of the media stream cannot begin immediately. The time it takes before presentation can begin is referred to as the *start latency*. Users might experience this as a delay between the time that they click the start button and the time when playback actually starts.

Multimedia presentations often combine several types of time-based media into a synchronized presentation. For example, background music might be played during an image slide-show, or animated text might be synchronized with an audio or video clip. When the presentation of multiple media streams is synchronized, it is essential to take into account the start latency of each stream--otherwise the playback of the different streams might actually begin at different times.

5.2.1.8 Presentation Quality

The quality of the presentation of a media stream depends on several factors, including:

- The compression scheme used
- The processing capability of the playback system
- The bandwidth available (for media streams acquired over the network)

Traditionally, the higher the quality, the larger the file size and the greater the processing power and bandwidth required. Bandwidth is usually represented as the number of bits that are transmitted in a certain period of time--the *bit rate*.

To achieve high-quality video presentations, the number of frames displayed in each period of time (the *frame rate*) should be as high as possible. Usually movies at a frame rate of 30 frames-per-second are considered indistinguishable from regular TV broadcasts or video tapes.

5.2.1.9 Media Processing

In most instances, the data in a media stream is manipulated before it is presented to the user. Generally, a series of processing operations occur before presentation:

1. If the stream is multiplexed, the individual tracks are extracted.
2. If the individual tracks are compressed, they are decoded.
3. If necessary, the tracks are converted to a different format.
4. Effect filters are applied to the decoded tracks (if desired).

The tracks are then delivered to the appropriate output device. If the media stream is to be stored instead of rendered to an output device, the processing stages might differ slightly. For example, capture of audio and video from a video camera, process the data, and save it to a file:

1. The audio and video tracks would be captured.
2. Effect filters would be applied to the raw tracks (if desired).
3. The individual tracks would be encoded.
4. The compressed tracks would be multiplexed into a single media stream.
5. The multiplexed media stream would then be saved to a file.

5.2.1.10 Demultiplexers and Multiplexers

A demultiplexer extracts individual tracks of media data from a multiplexed media stream. A *mutliplexer* performs the opposite function, it takes individual tracks of media data and merges them into a single multiplexed media stream.

5.2.1.11 Codecs

A codec performs media-data compression and decompression. When a track is encoded, it is converted to a compressed format suitable for storage or transmission; when it is decoded it is converted to a non-compressed (raw) format suitable for presentation.

Each codec has certain input formats that it can handle and certain output formats that it can generate. In some situations, a series of codecs might be used to convert from one format to another.

5.2.1.12 Effect Filters

An effect filter modifies the track data in some way, often to create special effects such as blur or echo.

Effect filters are classified as either pre-processing effects or post-processing effects, depending on whether they are applied before or after the codec processes the track. Typically, effect filters are applied to uncompressed (raw) data.

5.2.1.13 Renderers

A renderer is an abstraction of a presentation device. For audio, the presentation device is typically the computer's hardware audio card that outputs sound to the speakers. For video, the presentation device is typically the computer monitor.

5.2.1.14 Compositing

Certain specialized devices support *compositing*. Compositing time-based media is the process of combining multiple tracks of data onto a single presentation medium. For example, overlaying text on a video presentation is one common form of compositing. Compositing can be done in either hardware or software. A device that performs compositing can be abstracted as a renderer that can receive multiple tracks of input data.

5.2.1.15 Media Capture

Time-based media can be captured from a live source for processing and playback. For example, audio can be captured from a microphone or a video capture card can be used to obtain video from a camera. Capturing can be thought of as the *input* phase of the standard media processing model.

A capture device might deliver multiple media streams. For example, a video camera might deliver both audio and video. These streams might be captured and manipulated separately or combined into a single, multiplexed stream that contains both an audio track and a video track.

5.2.1.16 Capture Devices

To capture time-based media specialized hardware is needed--for example, to capture audio from a live source, a microphone and an appropriate audio card are required. Similarly, capturing a TV broadcast requires a TV tuner and an appropriate video capture card. Most systems provide a query mechanism to find out what capture devices are available.

Capture devices can be characterized as either push or pull sources. For example, a still camera is a pull source--the user controls when to capture an image. A microphone is a push source--the live source continuously provides a stream of audio.

The format of a captured media stream depends on the processing performed by the capture device. Some devices do very little processing and deliver raw, uncompressed data. Other capture devices might deliver the data in a compressed format.

5.2.1.17 Capture Controls

Controls are sometimes provided to enable the user to manage the capture process. For example, a capture control panel might enable the user to specify the data rate and encoding type for the captured stream and start and stop the capture process.

5.2.2 Meeting Protocols in Audio/Video

Implementation of meeting protocols in the audio and video conferencing component involves the identification of a method of representing the meeting protocols that the other components like text chat and whiteboard use to suit the multimedia component. The meeting protocols have been implemented based on the following set of rules:

5.2.2.1 Chairman Style meeting

- The chairman has primary control to either send audio/video to and receive from any other member in the meeting. Once the media connection is made the mode of collaboration is automatically changed to a talk(Hussein) mode between the two persons sending audio video to each other.

- When any person A, other than the chairman wants to send or receive media from anyone else in the meeting(Participant B), he or she selects the appropriate option from the pop up menu that appears on clicking on the participant's image. A request is immediately sent to the chairman informing that Participant A is requesting to talk to B. The chairman has the right to make a decision as to whether it should be permitted or not. In the event of the chairman giving permission, the participants immediately get into a talk mode and a media connection is established between them.

- In the event of a participant A, entering a side talk all the audio/video channel connections that he/she had with the rest of the meeting members get paused except for the connection between the participant that he/she is entering the side talk with. The same happens to the other participant in the side talk because he/she is entering the side talk too. This implies that both the participants who enter the side talk get totally shielded from the rest of the meeting.

- Different video sizes are also used to represent the state of collaboration that one is having with the rest of the team members at a given instance. Two window sizes have been used in the current implementation to show disconnected and the

talk modes. The smaller window means that the user does not have an active collaboration channel with the other participant in the meeting. The larger window means that the user is speaking to the other participant, meaning there is a live channel of collaboration.

5.2.3 High Level Architecture

Devices such as tape decks and VCRs provide a familiar model for recording, processing, and presenting time-based media. When a movie is played using a VCR, the media stream is provided to the VCR from the videotape. The VCR reads and interprets the data on the tape and sends appropriate signals to the television and speakers. JMF uses this same basic model. A *data source* encapsulates the media stream much like a videotape and a *player* provides processing and control mechanisms similar to a VCR. Playing and capturing audio and video with JMF requires the appropriate input and output devices such as microphones, cameras, speakers, and monitors.

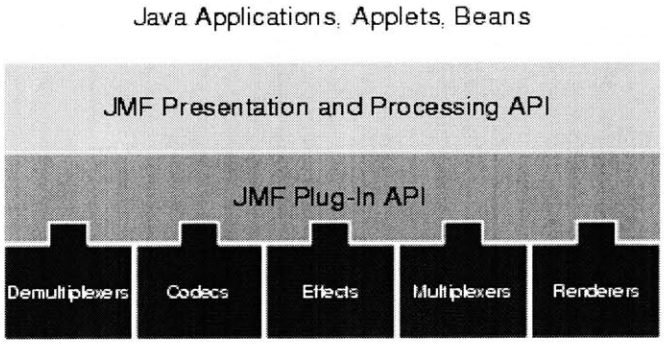

Figure 5-10. High-level JMF achitecture [JMF 1999].

Data sources and players are integral parts of JMF's high-level API for managing the capture, presentation, and processing of time-based media. JMF also provides a lower-level API that supports the seamless integration of custom processing components and extensions.

5.2.3.1 Managers

The JMF API consists mainly of interfaces that define the behavior and interaction of objects used to capture, process, and present time-based media. Implementations of these interfaces operate within the structure of the framework. By using intermediary objects called *managers*, JMF makes it easy to integrate new implementations of key interfaces that can be used seamlessly with existing classes.

JMF uses four managers:

- Manager--handles the construction of Players, Processors, DataSources, and DataSinks. This level of indirection allows new implementations to be integrated seamlessly with JMF. From the client perspective, these objects are always created the same way whether the requested object is constructed from a default implementation or a custom one.

- PackageManager--maintains a registry of packages that contain JMF classes, such as custom Players, Processors, DataSources, and DataSinks.

- CaptureDeviceManager--maintains a registry of available capture devices.

- PlugInManager--maintains a registry of available JMF plug-in processing components, such as Multiplexers, Demultiplexers, Codecs, Effects, and Renderers.

5.2.3.2 Event Model

JMF uses a structured event reporting mechanism to keep JMF-based programs informed of the current state of the media system and enable JMF-based programs to respond to media-driven error conditions, such as out-of data and resource unavailable conditions. Whenever a JMF object needs to report on the current conditions, it posts a MediaEvent. MediaEvent is subclassed to identify many particular types of events. These objects follow the established Java Beans patterns for events.

For each type of JMF object that can post MediaEvents, JMF defines a corresponding listener interface. To receive notification when a MediaEvent is posted, one needs to

implement the appropriate listener interface and register the listener class with the object that posts the event by calling its add*Listener* method.

Controller objects (such as Players and Processors) and certain Control objects such as GainControl post media events. RTPSessionManager objects also post events.

5.2.3.3 Data Model

JMF media players usually use DataSources to manage the transfer of media-content. A DataSource encapsulates both the location of media and the protocol and software used to deliver the media. Once obtained, the source cannot be reused to deliver other media. A DataSource is identified by either a JMF MediaLocator or a URL (universal resource locator). A MediaLocator is similar to a URL and can be constructed from a URL, but can be constructed even if the corresponding protocol handler is not installed on the system. (In Java, a URL can only be constructed if the corresponding protocol handler is installed on the system.) A DataSource manages a set of SourceStream objects. A standard data source uses a byte array as the unit of transfer. A *buffer data source* uses a Buffer object as its unit of transfer. JMF defines several types of DataSource objects:

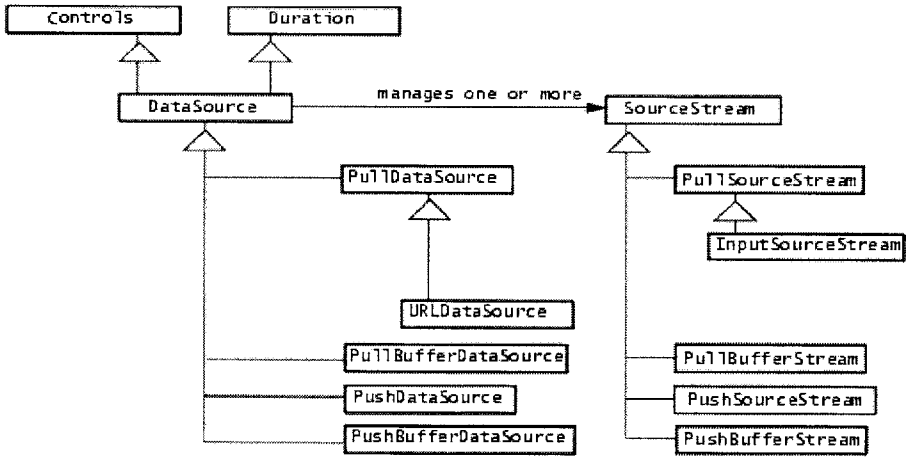

Figure 5-11. JMF data model [JMF 1999].

5.2.3.3.1 Push and Pull Data Sources

Media data can be obtained from a variety of sources, such as local or network files and live broadcasts. JMF data sources can be categorized according to how data transfer is initiated:

- *Pull Data-Source*--the client initiates the data transfer and controls the flow of data from pull data-sources. Established protocols for this type of data include Hypertext Transfer Protocol (HTTP) and FILE. JMF defines two types of pull data sources: PullDataSource and PullBufferDataSource, which uses a Buffer object as its unit of transfer.

- *Push Data-Source*--the server initiates the data transfer and controls the flow of data from a push data-source. Push data-sources include broadcast media, multicast media, and video-on-demand (VOD). For broadcast data, one protocol is the Real-time Transport Protocol (RTP), under development by the Internet Engineering Task Force (IETF). The MediaBase protocol developed by SGI is one protocol used for VOD. JMF defines two types of push data sources: PushDataSource and PushBufferDataSource, which uses a Buffer object as its unit of transfer.

The degree of control that a client program can extend to the user depends on the type of data source being presented. For example, an MPEG file can be repositioned and a client program could allow the user to replay the video clip or seek to a new position in the video. In contrast, broadcast media is under server control and cannot be repositioned. Some VOD protocols might support limited user control--for example, a client program might be able to allow the user to seek to a new position, but not fast forward or rewind.

5.2.3.3.2 Specialty Datasources

JMF defines two types of specialty data sources, cloneable data sources and merging data sources.

A cloneable data source can be used to create clones of either a pull or push DataSource. To create a cloneable DataSource, one needs to call the Manager createCloneableDataSource method and pass in the DataSource to be cloned. Once a DataSource has been passed to createCloneableDataSource, one should only interact with the cloneable DataSource and its clones; the original DataSource should no longer be used directly.

Cloneable data sources implement the SourceCloneable interface, which defines one method, createClone. By calling createClone, one can create any number of clones of the DataSource that was used to construct the cloneable DataSource. The clones can be controlled through the cloneable DataSource used to create them-- when connect, disconnect, start, or stop is called on the cloneable DataSource, the method calls are propagated to the clones.

The clones don't necessarily have the same properties as the cloneable data source used to create them or the original DataSource. For example, a cloneable data source created for a capture device might function as a master data source for its clones--in this case, unless the cloneable data source is used, the clones won't produce any data. If one hooks up both the cloneable data source and one or more clones, the clones will produce data at the same rate as the master.

A MergingDataSource can be used to combine the SourceStreams from several DataSources into a single DataSource. This enables a set of DataSources to be managed from a single point of control--when connect, disconnect, start, or stop is called on the MergingDataSource, the method calls are propagated to the merged DataSources.

To construct a MergingDataSource, one needs to call the Manager createMergingDataSource method and pass in an array that contains the data sources to be merged. To be merged, all of the DataSources must be of the same type; for example, a PullDataSource and a PushDataSource cannot be merged. The duration of the merged DataSource is the maximum of the merged DataSource objects' durations. The ContentType is application/mixed-media.

5.2.3.3.3 Data Formats

The exact media format of an object is represented by a Format object. The format itself carries no encoding-specific parameters or global timing information, it describes the format's encoding name and the type of data the format requires.

JMF extends Format to define audio- and video-specific formats.

An AudioFormat describes the attributes specific to an audio format, such as sample rate, bits per sample, and number of channels. A VideoFormat encapsulates information relevant to video data. Several formats are derived from VideoFormat to describe the attributes of common video formats, including:

- IndexedColorFormat
- RGBFormat
- YUVFormat
- JPEGFormat
- H261Format
- H263Format

To receive notification of format changes from a Controller, the ControllerListener interface has to be implemented listening for FormatChangeEvents.

5.2.3.4 Controls

JMF Control provides a mechanism for setting and querying attributes of an object. A Control often provides access to a corresponding user interface component that enables user control over an object's attributes. Many JMF objects expose Controls, including Controller objects, DataSource objects, DataSink objects, and JMF plug-ins.

Any JMF object that wants to provide access to its corresponding Control objects can implement the Controls interface. Controls defines methods for retrieving associated Control

objects. DataSource and PlugIn use the Controls interface to provide access to their Control objects.

5.2.3.4.1 Standard Controls

JMF defines the standard Control interfaces listed below:

CachingControl enables download progress to be monitored and displayed. If a Player or Processor can report its download progress, it implements this interface so that a progress bar can be displayed to the user.

GainControl enables audio volume adjustments such as setting the level and muting the output of a Player or Processor. It also supports a listener mechanism for volume changes.

DataSink or Multiplexer objects that read media from a DataSource and write it out to a destination such as a file can implement the StreamWriterControl interface. This Control enables the user to limit the size of the stream that is created.

FramePositioningControl and FrameGrabbingControl export frame-based capabilities for Players and Processors. FramePositioningControl enables precise frame positioning within a Player or Processor object's media stream. FrameGrabbingControl provides a mechanism for grabbing a still video frame from the video stream. The FrameGrabbingControl can also be supported at the Renderer level.

Objects that have a Format can implement the FormatControl interface to provide access to the Format. FormatControl also provides methods for querying and setting the format.

A TrackControl is a type of FormatControl that provides the mechanism for controlling what processing a Processor object performs on a particular track of media data. With the TrackControl methods, one can specify what format conversions are performed on individual tracks and select the Effect, Codec, or Renderer plug-ins that are used by the Processor.

Two controls, PortControl and MonitorControl enable user control over the capture process. PortControl defines methods for controlling the output of a capture device. MonitorControl enables media data to be previewed as it is captured or encoded.

BufferControl enables user-level control over the buffering done by a particular object.

JMF also defines several codec controls to enable control over hardware or software encoders and decoders:

- BitRateControl--used to export the bit rate information for an incoming stream or to control the encoding bit rate. Enables specification of the bit rate in bits per second.
- FrameProcessingControl--enables the specification of frame processing parameters that allow the codec to perform minimal processing when it is falling behind on processing the incoming data.
- FrameRateControl--enables modification of the frame rate.
- H261Control--enables control over the H.261 video codec still-image transmission mode.
- H263Control--enables control over the H.263 video-codec parameters, including support for the unrestricted vector, arithmetic coding, advanced prediction, PB Frames, and error compensation extensions.
- KeyFrameControl--enables the specification of the desired interval between key frames. (The encoder can override the specified key-frame interval if necessary.)
- MpegAudioControl--exports an MPEG audio codec's capabilities and enables the specification of selected MPEG encoding parameters.
- QualityControl--enables specification of a preference in the trade-off between quality and CPU usage in the processing performed by a codec. This quality hint can have different effects depending on the type of compression. A higher quality

setting will result in better quality of the resulting bits, for example better image quality for video.

- SilenceSuppressionControl--enables specification of silence suppression parameters for audio codecs. When silence suppression mode is on, an audio encoder does not output any data if it detects silence at its input.

5.2.3.5 Presentation

In JMF, the presentation process is modeled by the Controller interface. Controller defines the basic state and control mechanism for an object that controls, presents, or captures time-based media. It defines the phases that a media controller goes through and provides a mechanism for controlling the transitions between those phases. A number of the operations that must be performed before media data can be presented can be time consuming, so JMF allows programmatic control over when they occur. A Controller posts a variety of controller-specific MediaEvents to provide notification of changes in its status. To receive events from a Controller such as a Player, the ControllerListener interface needs to be implemented. The JMF API defines two types of Controllers: Players and Processors. A Player or Processor is constructed for a particular data source and is normally not re-used to present other media data.

5.2.3.4.1 Player

A Player processes an input stream of media data and renders it at a precise time.

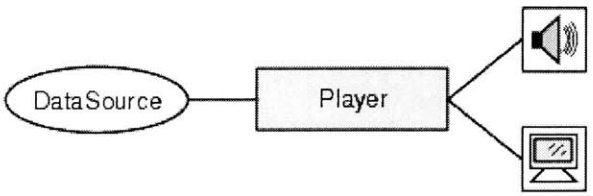

Figure 5-12. JMF player model [JMF 1999].

A DataSource is used to deliver the input media-stream to the Player. The rendering destination depends on the type of media being presented. A Player does not provide any control over the processing that it performs or how it renders the media data.

Player supports standardized user control and relaxes some of the operational restrictions imposed by Clock and Controller.

Player States

A Player can be in one of six states. The Clock interface defines the two primary states: *Stopped* and *Started*. To facilitate resource management, Controller breaks the *Stopped* state down into five standby states: *Unrealized*, *Realizing*, *Realized*, *Prefetching*, and *Prefetched*.

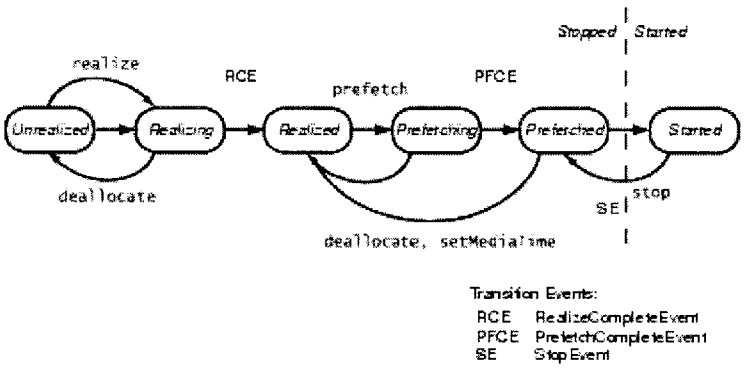

Figure 5-13. Player states [JMF 1999].

In normal operation, a Player steps through each state until it reaches the *Started* state:

- A Player in the *Unrealized* state has been instantiated, but does not yet know anything about its media. When a media Player is first created, it is *Unrealized*.
- When realize is called, a Player moves from the *Unrealized* state into the *Realizing* state. A *Realizing* Player is in the process of determining its resource requirements. During realization, a Player acquires the resources that it only needs to acquire once. These might include rendering resources other than exclusive-use resources. (Exclusive-use resources are limited resources such as particular hardware

devices that can only be used by one Player at a time; such resources are acquired during *Prefetching*.) A *Realizing* Player often downloads assets over the network.

- When a Player finishes *Realizing*, it moves into the *Realized* state. A *Realized* Player knows what resources it needs and information about the type of media it is to present. Because a *Realized* Player knows how to render its data, it can provide visual components and controls. Its connections to other objects in the system are in place, but it does not own any resources that would prevent another Player from starting.

- When prefetch is called, a Player moves from the *Realized* state into the *Prefetching* state. A *Prefetching* Player is preparing to present its media. During this phase, the Player preloads its media data, obtains exclusive-use resources, and does whatever else it needs to do to prepare itself to play. *Prefetching* might have to recur if a Player object's media presentation is repositioned, or if a change in the Player object's rate requires that additional buffers be acquired or alternate processing take place.

- When a Player finishes *Prefetching*, it moves into the *Prefetched* state. A *Prefetched* Player is ready to be started.

- Calling start puts a Player into the *Started* state. A *Started* Player object's time-base time and media time are mapped and its clock is running, though the Player might be waiting for a particular time to begin presenting its media data.

A Player posts TransitionEvents as it moves from one state to another. The ControllerListener interface provides a way for ones program to determine what state a Player is in and to respond appropriately. For example, when the program calls an asynchronous method on a Player or Processor, it needs to listen for the appropriate event to determine when the operation is complete.

Using this event reporting mechanism, a Player object's start latency can be managed by controlling when it begins *Realizing* and *Prefetching*. It also enables one to determine whether or not the Player is in an appropriate state before calling methods on the Player.

Methods Available in Each Player State

To prevent race conditions, not all methods can be called on a Player in every state. The following table identifies the restrictions imposed by JMF. If a method that is illegal in a Player object's current state is called, the Player throws an error or exception.

5.2.3.4.2 Processors

Processors can also be used to present media data. A Processor is just a specialized type of Player that provides control over what processing is performed on the input media stream. A Processor supports all of the same presentation controls as a Player.

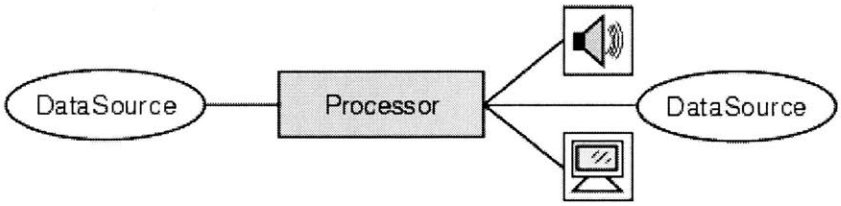

Figure 5-14. JMF processor model [JMF 1999].

In addition to rendering media data to presentation devices, a Processor can output media data through a DataSource so that it can be presented by another Player or Processor, further manipulated by another Processor, or delivered to some other destination, such as a file.

A Processor is a Player that takes a DataSource as input, performs some user-defined processing on the media data, and then outputs the processed media data. A Processor can send the output data to a presentation device or to a DataSource. If the data is sent to a DataSource, that DataSource can be used as the input to another Player or Processor, or as the input to a DataSink.

While the processing performed by a Player is predefined by the implementor, a Processor allows the application developer to define the type of processing that is applied to the media data. This enables the application of effects, mixing, and compositing in real-time.

The processing of the media data is split into several stages:

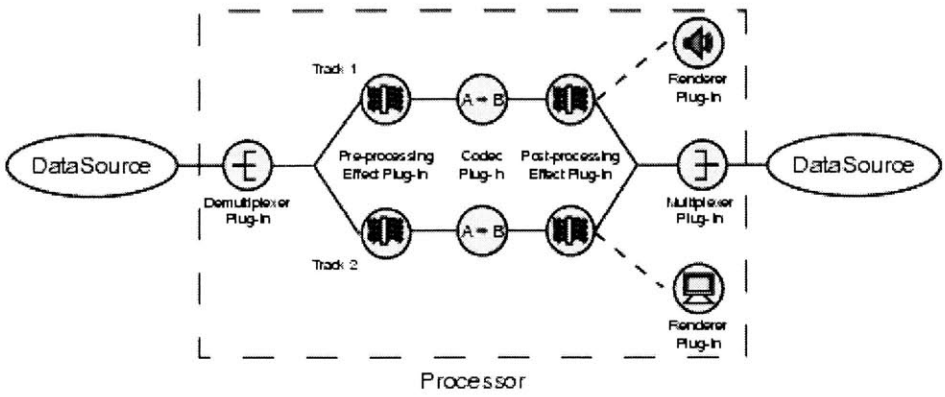

Figure 5-15. Processor stages [JMF 1999].

- Demultiplexing is the process of parsing the input stream. If the stream contains multiple tracks, they are extracted and output separately. For example, a QuickTime file might be demultiplexed into separate audio and video tracks. Demultiplexing is performed automatically whenever the input stream contains multiplexed data.

- Pre-Processing is the process of applying effect algorithms to the tracks extracted from the input stream.

- Transcoding is the process of converting each track of media data from one input format to another. When a data stream is converted from a compressed type to an uncompressed type, it is generally referred to as decoding. Conversely, converting from an uncompressed type to a compressed type is referred to as encoding.

- Post-Processing is the process of applying effect algorithms to decoded tracks.

- Multiplexing is the process of interleaving the transcoded media tracks into a single output stream. For example, separate audio and video tracks might be multiplexed into a single MPEG-1 data stream. The data type of the output stream can be specified with the Processor setOutputContentDescriptor method.

- Rendering is the process of presenting the media to the user.

The processing at each stage is performed by a separate processing component. These processing components are JMF *plug-in*s. If the Processor supports TrackControls, one can select which plug-ins to be used to process a particular track. There are five types of JMF plug-ins:

- Demultiplexer--parses media streams such as WAV, MPEG or QuickTime. If the stream is multiplexed, the separate tracks are extracted.
- Effect--performs special effects processing on a track of media data.
- Codec--performs data encoding and decoding.
- Multiplexer--combines multiple tracks of input data into a single interleaved output stream and delivers the resulting stream as an output DataSource.
- Renderer--processes the media data in a track and delivers it to a destination such as a screen or speaker.

Processor States

A Processor has two additional standby states, *Configuring* and *Configured*, which occur before the Processor enters the *Realizing* state..

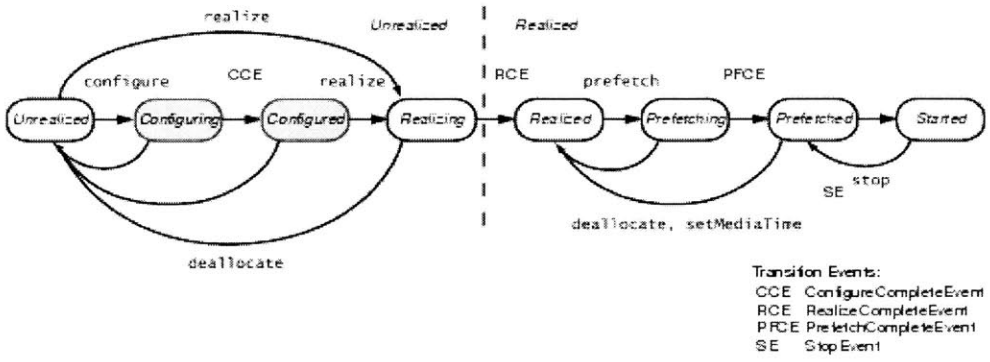

Figure 5-16. Processor states [JMF 1999].

- A Processor enters the *Configuring* state when configure is called. While the Processor is in the *Configuring* state, it connects to the DataSource, demultiplexes the input stream, and accesses information about the format of the input data.

- The Processor moves into the *Configured* state when it is connected to the DataSource and data format has been determined. When the Processor reaches the *Configured* state, a ConfigureCompleteEvent is posted.

- When Realize is called, the Processor is transitioned to the Realized state. Once the Processor is *Realized* it is fully constructed.

While a Processor is in the *Configured* state, getTrackControls can be called to get the TrackControl objects for the individual tracks in the media stream. These TrackControl objects enable one to specify the media processing operations that he or she wants the Processor to perform.

Calling realize directly on an *Unrealized* Processor automatically transitions it through the *Configuring* and *Configured* states to the *Realized* state. When this is done, the processing options cannot be configured through the TrackControls--the default Processor settings are used.

Calls to the TrackControl methods once the Processor is in the *Realized* state will typically fail, though some Processor implementations might support them.

Methods Available in Each Processor State

Since a Processor is a type of Player, the restrictions on when methods can be called on a Player also apply to Processors. Some of the Processor-specific methods also are restricted to particular states. The following table shows the restrictions that apply to a Processor. If a method that is illegal in the current state is called, the Processor throws an error or exception.

Processing Controls

The processing operations the Processor performs on a track can be controlled through the TrackControl for that track. Processor getTrackControls is called to get the TrackControl objects for all of the tracks in the media stream.

Through a TrackControl, one can explicitly select the Effect, Codec, and Renderer plug-ins needed to be used for the track. To find out what options are available, one can query the PlugInManager to find out what plug-ins are installed.

To control the transcoding that's performed on a track by a particular Codec, the Controls associated with the track can be obtained by calling the TrackControl getControls method. This method returns the codec controls available for the track, such as BitRateControl and QualityControl.

If the output data format needed is known in advance, the setFormat method can be used to to specify the Format and let the Processor choose an appropriate codec and renderer. Alternatively, the output format can be specified when the Processor is created by using a ProcessorModel. A ProcessorModel defines the input and output requirements for a Processor. When a ProcessorModel is passed to the appropriate Manager create method, the Manager does its best to create a Processor that meets the specified requirements.

Data Output

The getDataOutput method returns a Processor object's output as a DataSource. This DataSource can be used as the input to another Player or Processor or as the input to a *data sink.*

A Processor object's output DataSource can be of any type: PushDataSource, PushBufferDataSource, PullDataSource, or PullBufferDataSource.

Not all Processor objects output data--a Processor can render the processed data instead of outputting the data to a DataSource. A Processor that renders the media data is essentially a configurable Player.

Capture

A multimedia capturing device can act as a source for multimedia data delivery. For example, a microphone can capture raw audio input or a digital video capture board might deliver digital video from a camera. Such capture devices are abstracted as DataSources. For example, a device that provides timely delivery of data can be

represented as a PushDataSource. Any type of DataSource can be used as a capture DataSource: PushDataSource, PushBufferDataSource, PullDataSource, or PullBufferDataSource.

Some devices deliver multiple data streams--for example, an audio/video conferencing board might deliver both an audio and a video stream. The corresponding DataSource can contain multiple SourceStreams that map to the data streams provided by the device.

Media data Storage and Transmission

A DataSink is used to read media data from a DataSource and render the media to some destination--generally a destination other than a presentation device. A particular DataSink might write data to a file, write data across the network, or function as an RTP broadcaster.

Like Players, DataSink objects are constructed through the Manager using a DataSource. A DataSink can use a StreamWriterControl to provide additional control over how data is written to a file.

5.2.4 Working with Real-Time Media Streams

To send or receive a live media broadcast or conduct a video conference over the Internet or Intranet, one needs to be able to receive and transmit media streams in real-time. This section introduces streaming media concepts and describes the Real-time Transport Protocol JMF uses for receiving and transmitting media streams across the network.

5.2.4.1 Streaming Media

When media content is streamed to a client in real-time, the client can begin to play the stream without having to wait for the complete stream to download. In fact, the stream might not even have a predefined duration--downloading the entire stream before playing it would be impossible. The term *streaming media* is often used to refer to both this technique of delivering content over the network in real-time and the real-time media content that's delivered.

Streaming media is everywhere one looks on the web--live radio and television broadcasts and webcast concerts and events are being offered by a rapidly growing number of web portals, and it's now possible to conduct audio and video conferences over the Internet. By enabling the delivery of dynamic, interactive media content across the network, streaming media is changing the way people communicate and access information.

5.2.4.2 Protocols for Streaming Media

Transmitting media data across the net in real-time requires high network throughput. It's easier to compensate for lost data than to compensate for large delays in receiving the data. This is very different from accessing static data such as a file, where the most important thing is that all of the data arrive at its destination. Consequently, the protocols used for static data don't work well for streaming media.

The HTTP and FTP protocols are based on the Transmission Control Protocol (TCP). TCP is a transport-layer protocol designed for reliable data communications on low-bandwidth, high-error-rate networks. When a packet is lost or corrupted, it's retransmitted. The overhead of guaranteeing reliable data transfer slows the overall transmission rate.

For this reason, underlying protocols other than TCP are typically used for streaming media. One that's commonly used is the User Datagram Protocol (UDP). UDP is an unreliable protocol; it does not guarantee that each packet will reach its destination. There's also no guarantee that the packets will arrive in the order that they were sent. The receiver has to be able to compensate for lost data, duplicate packets, and packets that arrive out of order.

Like TCP, UDP is a general transport-layer protocol--a lower-level networking protocol on top of which more application-specific protocols are built. The Internet standard for transporting real-time data such as audio and video is the Real-Time Transport Protocol (RTP).

5.2.4.3 Real-Time Transport Protocol

RTP provides end-to-end network delivery services for the transmission of real-time data. RTP is network and transport-protocol independent, though it is often used over UDP.

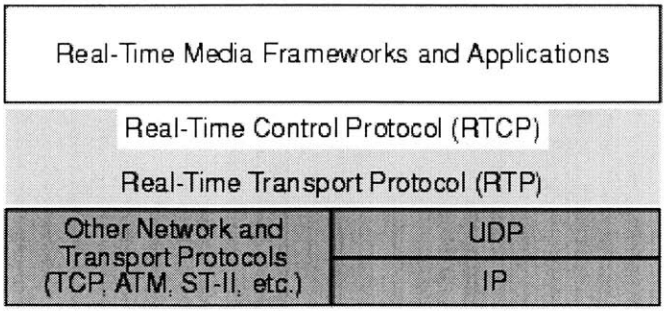

Figure 5-17. RTP architecture [JMF 1999].

RTP can be used over both unicast and multicast network services. Over a *unicast* network service, separate copies of the data are sent from the source to each destination. Over a *multicast* network service, the data is sent from the source only once and the network is responsible for transmitting the data to multiple locations. Multicasting is more efficient for many multimedia applications, such as video conferences. The standard Internet Protocol (IP) supports multicasting. RTP enables one to identify the type of data being transmitted, determine what order the packets of data should be presented in, and synchronize media streams from different sources.

RTP data packets are not guaranteed to arrive in the order that they were sent--in fact, they're not guaranteed to arrive at all. It's up to the receiver to reconstruct the sender's packet sequence and detect lost packets using the information provided in the packet header.

While RTP does not provide any mechanism to ensure timely delivery or provide other quality of service guarantees, it is augmented by a control protocol (RTCP) that enables monitoring of the quality of the data distribution. RTCP also provides control and identification mechanisms for RTP transmissions.

5.2.4.3.1 RTP Architecture

An RTP *session* is an association among a set of applications communicating with RTP. A session is identified by a network address and a pair of ports. One port is used for the media data and the other is used for control (RTCP) data.

A *participant* is a single machine, host, or user participating in the session. Participation in a session can consist of passive reception of data (receiver), active transmission of data (sender), or both.

Each media type is transmitted in a different session. For example, if both audio and video are used in a conference, one session is used to transmit the audio data and a separate session is used to transmit the video data. This enables participants to choose which media types they want to receive--for example, someone who has a low-bandwidth network connection might only want to receive the audio portion of a conference.

Data Packets

The media data for a session is transmitted as a series of packets. A series of data packets that originate from a particular source is referred to as an *RTP stream*. Each RTP data packet in a stream contains two parts, a structured header and the actual data (the packet's *payload*).

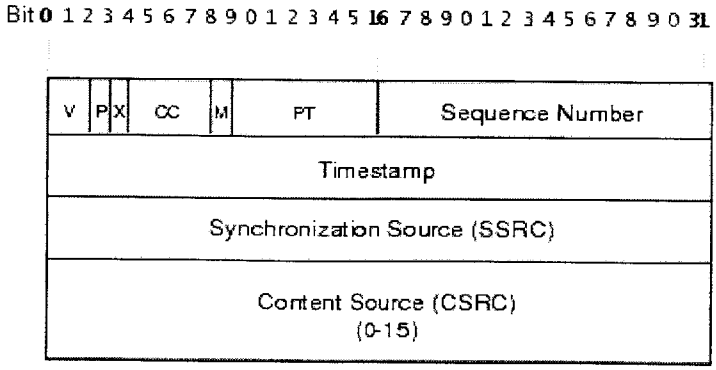

Figure 5-18. RTP data-packet header format [JMF 1999].

The header of an RTP data packet contains:

- The RTP version number (V): 2 bits. The version defined by the current specification is 2.
- Padding (P): 1 bit. If the padding bit is set, there are one or more bytes at the end of the packet that are not part of the payload. The very last byte in the packet indicates the number of bytes of padding. The padding is used by some encryption algorithms.
- Extension (X): 1 bit. If the extension bit is set, the fixed header is followed by one header extension. This extension mechanism enables implementations to add information to the RTP Header.
- CSRC Count (CC): 4 bits. The number of CSRC identifiers that follow the fixed header. If the CSRC count is zero, the synchronization source is the source of the payload.
- Marker (M): 1 bit. A marker bit defined by the particular media profile.
- Payload Type (PT): 7 bits. An index into a media profile table that describes the payload format. The payload mappings for audio and video are specified in RFC 1890.
- Sequence Number: 16 bits. A unique packet number that identifies this packet's position in the sequence of packets. The packet number is incremented by one for each packet sent.
- Timestamp: 32 bits. Reflects the sampling instant of the first byte in the payload. Several consecutive packets can have the same timestamp if they are logically generated at the same time--for example, if they are all part of the same video frame.
- SSRC: 32 bits. Identifies the synchronization source. If the CSRC count is zero, the payload source is the synchronization source. If the CSRC count is nonzero, the SSRC identifies the mixer.

- CSRC: 32 bits each. Identifies the contributing sources for the payload. The number of contributing sources is indicated by the CSRC count field; there can be up to 16 contributing sources. If there are multiple contributing sources, the payload is the mixed data from those sources.

Control Packets

In addition to the media data for a session, control data (RTCP) packets are sent periodically to all of the participants in the session. RTCP packets can contain information about the quality of service for the session participants, information about the source of the media being transmitted on the data port, and statistics pertaining to the data that has been transmitted so far.

There are several types of RTCP packets:

- Sender Report
- Receiver Report
- Source Description
- Bye
- Application-specific

RTCP packets are "stackable" and are sent as a compound packet that contains at least two packets, a report packet and a source description packet.

All participants in a session send RTCP packets. A participant that has recently sent data packets issues a *sender report.* The sender report (SR) contains the total number of packets and bytes sent as well as information that can be used to synchronize media streams from different sessions.

Session participants periodically issue *receiver reports* for all of the sources from which they are receiving data packets. A receiver report (RR) contains information about the number of packets lost, the highest sequence number received, and a timestamp that can be used to estimate the round-trip delay between a sender and the receiver.

The first packet in a compound RTCP packet has to be a report packet, even if no data has been sent or received--in which case, an empty receiver report is sent.

All compound RTCP packets must include a source description (SDES) element that contains the canonical name (CNAME) that identifies the source. Additional information might be included in the source description, such as the source's name, email address, phone number, geographic location, application name, or a message describing the current state of the source.

When a source is no longer active, it sends an RTCP BYE packet. The BYE notice can include the reason that the source is leaving the session.

RTCP APP packets provide a mechanism for applications to define and send custom information via the RTP control port.

5.2.4.3.2 RTP Applications

RTP applications are often divided into those that need to be able to receive data from the network (RTP Clients) and those that need to be able to transmit data across the network (RTP Servers). Some applications do both--for example, conferencing applications capture and transmit data at the same time that they're receiving data from the network.

Receiving Media Streams From the Network

Being able to receive RTP streams is necessary for several types of applications. For example:

- Conferencing applications need to be able to receive a media stream from an RTP session and render it on the console.
- A telephone answering machine application needs to be able to receive a media stream from an RTP session and store it in a file.
- An application that records a conversation or conference must be able to receive a media stream from an RTP session and both render it on the console and store it in a file.

Transmitting Media Streams Across the Network

RTP server applications transmit captured or stored media streams across the network.

For example, in a conferencing application, a media stream might be captured from a video camera and sent out on one or more RTP sessions. The media streams might be encoded in multiple media formats and sent out on several RTP sessions for conferencing with heterogeneous receivers. Multiparty conferencing could be implemented without IP multicast by using multiple unicast RTP sessions.

5.2.4.3.3 Understanding the JMF RTP API

JMF enables the playback and transmission of RTP streams through the APIs defined in the javax.media.rtp, javax.media.rtp.event, and javax.media.rtp.rtcp packages. JMF can be extended to support additional RTP-specific formats and dynamic payloads through the standard JMF plug-in mechanism.

Incoming RTP streams can be played locally, saved to a file, or both.

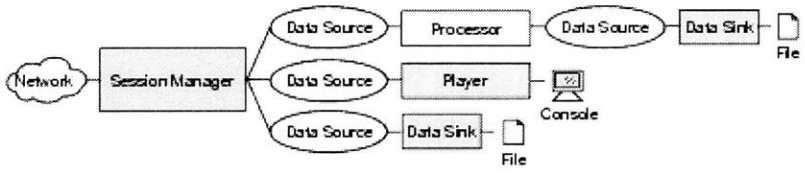

Figure 5-19. RTP reception [JMF 1999].

For example, the RTP APIs could be used to implement a telephony application that answers calls and records messages like an answering machine.

Similarly, the RTP APIs can be used to transmit captured or stored media streams across the network. Outgoing RTP streams can originate from a file or a capture device. The outgoing streams can also be played locally, saved to a file, or both.

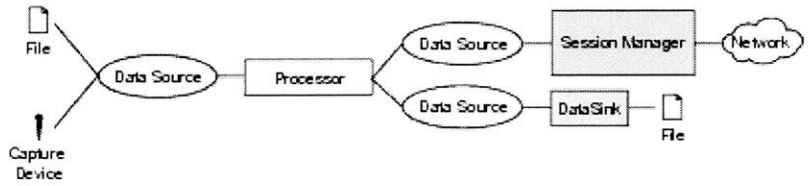

Figure 5-20. RTP transmission [JMF 1999].

CAIRO has a similar implementation of audio/video conferencing based on the RTP API.

RTP Detailed Architecture

The JMF RTP APIs are designed to work seamlessly with the capture, presentation, and processing capabilities of JMF. Players and processors are used to present and manipulate RTP media streams just like any other media content. One can transmit media streams that have been captured from a local capture device using a capture DataSource or that have been stored to a file using a DataSink. Similarly, JMF can be extended to support additional RTP formats and payloads through the standard plug-in mechanism.

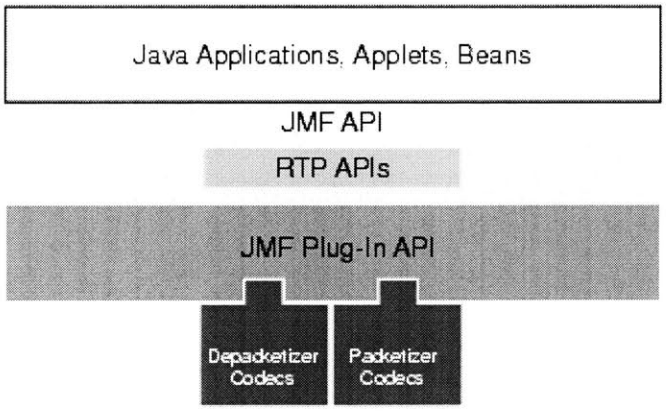

Figure 5-21. High-level JMF RTP architecture [JMF 1999].

Session Manager

In JMF, a SessionManager is used to coordinate an RTP session. The session manager keeps track of the session participants and the streams that are being transmitted.

The session manager maintains the state of the session as viewed from the local participant. In effect, a session manager is a local representation of a distributed entity, the RTP session. The session manager also handles the *RTCP* control channel, and supports RTCP for both senders and receivers.

The SessionManager interface defines methods that enable an application to initialize and start participating in a session, remove individual streams created by the application, and close the entire session.

Session Statistics

The session manager maintains statistics on all of the RTP and RTCP packets sent and received in the session. Statistics are tracked for the entire session on a per-stream basis. The session manager provides access to global reception and transmission statistics:

- GlobalReceptionStats: Maintains global reception statistics for the session.
- GlobalTransmissionStats: Maintains cumulative transmission statistics for all local senders.

  Statistics for a particular recipient or outgoing stream are available from the stream:

- ReceptionStats: Maintains source reception statistics for an individual participant.
- TransmissionStats: Maintains transmission statistics for an individual send stream.

Session Participants

The session manager keeps track of all of the participants in a session. Each participant is represented by an instance of a class that implements the Participant interface. SessionManagers create a Participant whenever an RTCP packet arrives that contains a source description (SDES) with a canonical name(CNAME) that has not been seen before in the session (or has timed-out since its last use). Participants can be passive (sending control packets only) or active (also sending one or more RTP data streams).

There is exactly one *local participant* that represents the local client/server participant. A local participant indicates that it will begin sending RTCP control messages or data and maintain state on incoming data and control messages by starting a session.

A participant can own more than one stream, each of which is identified by the synchronization source identifier (SSRC) used by the source of the stream.

Session Streams

The SessionManager maintains an RTPStream object for each stream of RTP data packets in the session. There are two types of RTP streams:

- ReceiveStream represents a stream that's being received from a remote participant.
- SendStream represents a stream of data coming from the Processor or input DataSource that is being sent over the network.

A ReceiveStream is constructed automatically whenever the session manager detects a new source of RTP data. To create a new SendStream, one needs to call the SessionManager createSendStream method.

RTP Events

Several RTP-specific events are defined in javax.media.rtp.event. These events are used to report on the state of the RTP session and streams.

To receive notification of RTP events, the appropriate RTP listener has to be implemented and registered with the session manager:

- SessionListener: Receives notification of changes in the state of the session.
- SendStreamListener: Receives notification of changes in the state of an RTP stream that's being transmitted.
- ReceiveStreamListener: Receives notification of changes in the state of an RTP stream that's being received.

- RemoteListener: Receives notification of events or RTP control messages received from a remote participant.

Session Listener

A SessionListener can be implemented to receive notification about events that pertain to the RTP session as a whole, such as the addition of new participants.

There are two types of session-wide events:

- NewParticipantEvent: Indicates that a new participant has joined the session.
- LocalCollisionEvent: Indicates that the participant's synchronization source is already in use.

Send Stream Listener

A SendStreamListener can be implemented to receive notification whenever:

- New send streams are created by the local participant.
- The transfer of data from the DataSource used to create the send stream has started or stopped.
- The send stream's format or payload changes.

  There are five types of events associated with a SendStream:

- NewSendStreamEvent: Indicates that a new send stream has been created by the local participant.
- ActiveSendStreamEvent: Indicates that the transfer of data from the DataSource used to create the send stream has started.
- InactiveSendStreamEvent: Indicates that the transfer of data from the DataSource used to create the send stream has stopped.
- LocalPayloadChangeEvent: Indicates that the stream's format or payload has changed.

- StreamClosedEvent: Indicates that the stream has been closed.

Receive Stream Listener

A ReceiveStreamListener can be implemented to receive notification whenever:

- New receive streams are created.
- The transfer of data starts or stops.
- The data transfer times out.
- A previously orphaned ReceiveStream has been associated with a Participant.
- An RTCP APP packet is received.
- The receive stream's format or payload changes.

This interface can also be used to get a handle on the stream and access the RTP DataSource so that a MediaHandler can be created.

There are seven types of events associated with a ReceiveStream:

- NewReceiveStreamEvent: Indicates that the session manager has created a new receive stream for a newly-detected source.
- ActiveReceiveStreamEvent: Indicates that the transfer of data has started.
- InactiveReceiveStreamEvent: Indicates that the transfer of data has stopped.
- TimeoutEvent: Indicates that the data transfer has timed out.
- RemotePayloadChangeEvent: Indicates that the format or payload of the receive stream has changed.
- StreamMappedEvent: Indicates that a previously orphaned receive stream has been associated with a participant.
- ApplicationEvent: Indicates that an RTCP APP packet has been received.

Remote Listener

A RemoteListener can be implemented to receive notification of events or RTP control messages received from a remote participants.

RTP Data

The streams within an RTP session are represented by RTPStream objects. There are two types of RTPStreams: ReceiveStream and SendStream. Each RTP stream has a buffer data source associated with it. For ReceiveStreams, this DataSource is always a PushBufferDataSource.

The session manager automatically constructs new receive streams as it detects additional streams arriving from remote participants. new send streams are constructed by calling createSendStream on the session manager.

Data Handlers

The JMF RTP APIs are designed to be transport-protocol independent. A custom RTP data handler can be created to enable JMF to work over a specific transport protocol. The data handler is a DataSource that can be used as the media source for a Player.

The abstract class RTPPushDataSource defines the basic elements of a JMF RTP data handler. A data handler has both an input data stream (PushSourceStream) and an output data stream (OuputDataStream). A data handler can be used for either the data channel or the control channel of an RTP session. If it is used for the data channel, the data handler implements the DataChannel interface.

An RTPSocket is an RTPPushDataSource has both a data and control channel. Each channel has an input and output stream to stream data to and from the underlying network. An RTPSocket can export RTPControls to add dynamic payload information to the session manager.

Because a custom RTPSocket can be used to construct a Player through the Manager, JMF defines the name and location for custom RTPSocket implementations:

<protocol package-prefix>.media.protocol.rtpraw.DataSource

RTP Data Formats

All RTP-specific data uses an RTP-specific format encoding as defined in the AudioFormat and VideoFormat classes. For example, gsm RTP encapsulated packets have the encoding set to AudioFormat.GSM_RTP, while jpeg-encoded video formats have the encoding set to VideoFormat.JPEG_RTP.

AudioFormat defines four standard RTP-specific encoding strings:

```
public static final String ULAW_RTP = "JAUDIO_G711_ULAW/rtp";
public static final String DVI_RTP = "dvi/rtp";
public static final String G723_RTP = "g723/rtp";
public static final String GSM_RTP = "gsm/rtp";
```

VideoFormat defines three standard RTP-specific encoding strings:

```
public static final String JPEG_RTP = "jpeg/rtp";
public static final String H261_RTP = "h261/rtp";
public static final String H263_RTP = "h263/rtp";
```

RTP Controls

The RTP API defines one RTP-specific control, RTPControl. RTPControl is typically implemented by RTP-specific DataSources. It provides a mechanism to add a mapping between a dynamic payload and a Format. RTPControl also provides methods for accessing session statistics and getting the current payload Format.

SessionManager also extends the Controls interface, enabling a session manager to export additional Controls through the getControl and getControls methods. For example, the session manager can export a BufferControl to enable one to specify the buffer length and threshold.

Reception

The presentation of an incoming RTP stream is handled by a Player. To receive and present a single stream from an RTP session, a MediaLocator can be used that describes the session to construct a Player. A media locator for an RTP session is of the form:

rtp://address:port[:ssrc]/content-type/[ttl]

The Player is constructed and connected to the first stream in the session.

If there are multiple streams in the session that need to be presented, a session manager has to be used. Notification can be received from the session manager whenever a stream is added to the session and construct a Player for each new stream. Using a session manager also enables direct monitoring and control of the session.

Transmission

A session manager can also be used to initialize and control a session so that data can be streamed across the network. The data to be streamed is acquired from a Processor.

For example, to create a send stream to transmit data from a live capture source, it would be necessary to:

1. Create, initialize, and start a SessionManager for the session.
2. Construct a Processor using the appropriate capture DataSource.
3. Set the output format of the Processor to an RTP-specific format. An appropriate RTP packetizer codec must be available for the data format to be transmitted.
4. Retrieve the output DataSource from the Processor.
5. Call createSendStream on the session manager and pass in the DataSource.

The transmission is controlled through the SendStream start and stop methods.

When it is first started, the SessionManager behaves as a receiver (sends out RTCP receiver reports). As soon as a SendStream is created, it begins to send out RTCP sender reports and

behaves as a sender host as long as one or more send streams exist. If all SendStreams are closed (not just stopped), the session manager reverts to being a passive receiver.

5.2.5 Implementation Details

The basic architecture of the Media component in CAIRO is done in two steps:

- Streaming the media from a client to a broadcasting server
- Streaming the media from the broadcasting server to the clients that request the particular media stream that the broadcasting server is receiving

5.3 Speech to Text

The Java Speech API [JSAPI 1998] defines a standard, easy-to-use, cross-platform software interface to state-of-the-art speech technology. Two core speech technologies are supported through the Java Speech API: *speech recognition* and *speech synthesis*. Speech recognition provides computers with the ability to listen to spoken language and to determine what has been said. In other words, it processes audio input containing speech by converting it to text. Speech synthesis provides the reverse process of producing synthetic speech from text generated by an application, an applet or a user. It is often referred to as *text-to-speech* technology.

Enterprises and individuals can benefit from a wide range of applications of speech technology using the Java Speech API. For instance, interactive voice response systems are an attractive alternative to touch-tone interfaces over the telephone; dictation systems can be considerably faster than typed input for many users; speech technology improves accessibility to computers for many people with physical limitations.

Speech interfaces give Java application developers the opportunity to implement distinct and engaging personalities for their applications and to differentiate their products. Java application developers will have access to state- of-the-art speech technology from leading speech companies. With a standard API for speech, users can choose the speech products which best meet their needs and their budget.

The Java Speech API was developed through an open development process. With the active involvement of leading speech technology companies, with input from application developers and with months of public review and comment, the specification has achieved a high degree of technical excellence. As a specification for a rapidly evolving technology, Sun will support and enhance the Java Speech API to maintain its leading capabilities.

The Java Speech API is an extension to the Java platform. Extensions are packages of classes written in the Java programming language (and any associated native code) that application developers can use to extend the functionality of the core part of the Java platform.

5.3.1 Speech Synthesis

A speech synthesizer converts written text into spoken language. Speech synthesis is also referred to as *text-to-speech* (TTS) conversion.

The major steps in producing speech from text are as follows:

- *Structure analysis*: process the input text to determine where paragraphs, sentences and other structures start and end. For most languages, punctuation and formatting data are used in this stage.
- *Text pre-processing*: analyze the input text for special constructs of the language. In English, special treatment is required for abbreviations, acronyms, dates, times, numbers, currency amounts, email addresses and many other forms. Other languages need special processing for these forms and most languages have other specialized requirements.

The result of these first two steps is a spoken form of the written text. The following are examples of the difference between written and spoken text.

St. Mathews hospital is on Main St.

-> "Saint Mathews hospital is on Main street"

Add $20 to account 55374.

-> "Add twenty dollars to account five five, three seven four."

Leave at 5:30 on 5/15/99.
-> "Leave at five thirty on May fifteenth nineteen ninety nine."

The remaining steps convert the spoken text to speech.

- *Text-to-phoneme conversion*: convert each word to *phonemes*. A phoneme is a basic unit of sound in a language. US English has around 45 phonemes including the consonant and vowel sounds. For example, "times" is spoken as four phonemes "t ay m s". Different languages have different sets of sounds (different phonemes). For example, Japanese has fewer phonemes including sounds not found in English, such as "ts" in "tsunami".

- *Prosody analysis*: process the sentence structure, words and phonemes to determine appropriate *prosody* for the sentence. Prosody includes many of the features of speech other than the sounds of the words being spoken. This includes the pitch (or melody), the timing (or rhythm), the pausing, the speaking rate, the emphasis on words and many other features. Correct prosody is important for making speech sound right and for correctly conveying the meaning of a sentence.

- *Waveform production*: finally, the phonemes and prosody information are used to produce the audio waveform for each sentence. There are many ways in which the speech can be produced from the phoneme and prosody information. Most current systems do it in one of two ways: *concatenation* of chunks of recorded human speech, or *formant synthesis* using signal processing techniques based on knowledge of how phonemes sound and how prosody affects those phonemes. The details of waveform generation are not typically important to application developers.

5.3.1.1 Speech Synthesis Limitations

Speech synthesizers can make errors in any of the processing steps described above. Human ears are well-tuned to detecting these errors, so careful work by developers can minimize errors and improve the speech output quality.

The Java Speech API and the *Java Speech Markup Language* (JSML) provide many ways for an application developer to improve the output quality of a speech synthesizer. The Java Synthesis Markup Language defines how to markup text input to a speech synthesizer with information that enables the synthesizer to enhance the speech output quality. In brief, some of its features which enhance quality include:

- Ability to mark the start and end of paragraphs and sentences.
- Ability to specify pronunciations for any word, acronym, abbreviation or other special text representation.
- Explicit control of pauses, boundaries, emphasis, pitch, speaking rate and loudness to improve the output prosody.

  These features allow a developer or user to override the behavior of a speech synthesizer to correct most of the potential errors described above. The following is a description of some of the sources of errors and how to minimize problems.

- *Structure analysis*: punctuation and formatting do not consistently indicate where paragraphs, sentences and other structures start and end. For example, the final period in "U.S.A." might be misinterpreted as the end of a sentence. Try: Explicitly marking paragraphs and sentences in JSML reduces the number of structural analysis errors.
- *Text pre-processing*: it is not possible for a synthesizer to know all the abbreviations and acronyms of a language. It is not always possible for a synthesizer to determine how to process dates and times, for example, is "8/5" the "eighth of May" of the "fifth of August"? Should "1998" be read as "nineteen ninety eight" (as a year), as "one thousand and ninety eight" (a regular number) or

as "one nine nine eight" (part of a telephone number). Special constructs such as email addresses are particularly difficult to interpret, for example, should a synthesizer say "tedwards@cat.com" as "Ted Wards", as "T. Edwards", as "Cat dot com" or as "C. A. T. dot com"?
Try: The SAYAS element of JSML supports substitutions of text for abbreviations, acronyms and other idiosyncratic textual forms.

- *Text-to-phoneme conversion*: most synthesizers can pronounce tens of thousands or even hundreds of thousands of words correctly. However, there are always new words which it must guess for (especially proper names for people, companies, products, etc.), and words for which the pronunciation is ambiguous (for example, "object" as "OBject" or "obJECT", or "row" as a line or as a fight).
Try: The SAYAS element of JSML is used to provide phonetic pronunciations for unusual and ambiguous words.

- *Prosody analysis*: to correctly phrase a sentence, to produce the correct melody for a sentence, and to correctly emphasize words ideally requires an understanding of the meaning of languages that computers do not possess. Instead, speech synthesizers must try to guess what a human might produce and at times, the guess is artificial and unnatural.
Try: The EMP, BREAK and PROS elements of JSML can be used to indicate preferred emphasis, pausing, and prosodic rendering respectively for text.

- *Waveform production*: without lips, mouths, lungs and the other apparatus of human speech, a speech synthesizer will often produce speech which sounds artificial, mechanical or otherwise different from human speech. In some circumstances a robotic sound is desirable, but for most applications speech that sounds as close to human as possible is easier to understand and easier to listen to for long periods of time.

5.3.1.2 Speech Synthesis Assessment

The major feature of a speech synthesizer that affects its understandability, its acceptance by users and its usefulness to application developers is its output quality. Knowing how to evaluate speech synthesis quality and knowing the factors that influence the output quality are important in the deployment of speech synthesis.

Humans are conditioned by a lifetime of listening and speaking. The human ear (and brain) are very sensitive to small changes in speech quality. A listener can detect changes that might indicate a user's emotional state, an accent, a speech problem or many other factors. The quality of current speech synthesis remains below that of human speech, so listeners must make more effort than normal to understand synthesized speech and must ignore errors. For new users, listening to a speech synthesizer for extended periods can be tiring and unsatisfactory.

The two key factors a developer must consider when assessing the quality of a speech synthesizer are its *understandability* and its *naturalness*. Understandability is an indication of how reliably a listener will understand the words and sentences spoken by the synthesizer. Naturalness is an indication of the extent to which the synthesizer sounds like a human - a characteristic that is desirable for most, but not all, applications.

Understandability is affected by the ability of a speech synthesizer to perform all the processing steps described above because any error by the synthesizer has the potential to mislead a listener. Naturalness is affected more by the later stages of processing, particularly the processing of prosody and the generation of the speech waveform.

Though it might seem counter-intuitive, it is possible to have an artificial- sounding voice that is highly understandable. Similarly, it is possible to have a voice that sounds natural but is not always easy to understand (though this is less common).

5.3.2 Speech Recognition

Speech recognition is the process of converting spoken language to written text or some similar form. The basic characteristics of a speech recognizer supporting the Java Speech API are:

- It is mono-lingual: it supports a single specified language.
- It processes a single input audio stream.
- It can optionally adapt to the voice of its users.
- Its grammars can be dynamically updated.
- It has a small, defined set of application-controllable properties.

The major steps of a typical speech recognizer are:

- *Grammar design*: recognition grammars define the words that may be spoken by a user and the patterns in which they may be spoken. A grammar must be created and activated for a recognizer to know what it should listen for in incoming audio. Grammars are described below in more detail.
- *Signal processing*: analyze the spectrum (frequency) characteristics of the incoming audio.
- *Phoneme recognition*: compare the spectrum patterns to the patterns of the phonemes of the language being recognized.
- *Word recognition*: compare the sequence of likely phonemes against the words and patterns of words specified by the active grammars.
- *Result generation*: provide the application with information about the words the recognizer has detected in the incoming audio. The result information is always provided once recognition of a single utterance (often a sentence) is complete, but may also be provided during the recognition process. The result always indicates

the recognizer's best guess of what a user said, but may also indicate alternative guesses.

Most of the processes of a speech recognizer are automatic and are not controlled by the application developer. For instance, microphone placement, background noise, sound card quality, system training, CPU power and speaker accent all affect recognition performance but are beyond an application's control.

The primary way in which an application controls the activity of a recognizer is through control of its *grammars*.

A grammar is an object in the Java Speech API which indicates what words a user is expected to say and in what patterns those words may occur. Grammars are important to speech recognizers because they constrain the recognition process. These constraints makes recognition faster and more accurate because the recognizer does not have to check for bizarre sentences, for example, "pink is recognizer speech my".

The Java Speech API supports two basic grammar types: *rule grammars* and *dictation grammars*. These grammar types differ in the way in which applications set up the grammars, the types of sentences they allow, the way in which results are provided, the amount of computational resources required, and the way in which they are effectively used in application design. The grammar types are describe in more detail below.

Other speech recognizer controls available to a Java application include pausing and resuming the recognition process, direction of result events and other events relating to the recognition processes, and control of the recognizer's vocabulary.

5.3.2.1 Rule Grammars

In a rule-based speech recognition system, an application provides the recognizer with rules that define what the user is expected to say. These rules constrain the recognition process. Careful design of the rules, combined with careful user interface design, will produce rules that allow users reasonable freedom of expression while still limiting the

range of things that may be said so that the recognition process is as fast and accurate as possible.

Any speech recognizer that supports the Java Speech API must support rule grammars.

The following is an example of a simple rule grammar. It is represented in the Java Speech Grammar Format (JSGF) which is defined in detail in the Java Speech Grammar Format Specification.

```
#JSGF V1.0;
// Define the grammar name
grammar SimpleCommands;
// Define the rules
public <Command> = [<Polite>] <Action> <Object> (and <Object>)*;
<Action> = open | close | delete;
<Object> = the window | the file;
<Polite> = please;
```

Rule names are surrounded by angle brackets. Words that may be spoken are written as plain text. This grammar defines one *public* rule, <Command>, that may be spoken by users. This rule is a combination of three sub-rules, <Action>, <Object> and <Polite>. The square brackets around the reference to <Polite> mean that it is optional. The parentheses around "and <Object>" group the word and the rule reference together. The asterisk following the group indicates that it may occur zero or more times.

The grammar allows a user to say commands such as "Open the window" and "Please close the window and the file".

The Java Speech Grammar Format Specification defines the full behavior of rule grammars and discusses how complex grammars can be constructed by combining smaller grammars. With JSGF application developers can reuse grammars, can provide Javadoc-style documentation and can use the other facilities that enable deployment of advanced speech systems.

5.3.2.2 Dictation Grammars

Dictation grammars impose fewer restrictions on what can be said, making them closer to providing the ideal of free-form speech input. The cost of this greater freedom is that they require more substantial computing resources, require higher quality audio input and tend to make more errors.

A dictation grammar is typically larger and more complex than rule-based grammars. Dictation grammars are typically developed by statistical training on large collections of written text. Fortunately, developers don't need to know any of this because a speech recognizer that supports a dictation grammar through the Java Speech API has a built-in dictation grammar. An application that needs to use that dictation grammar simply requests a reference to it and enables it when the user might say something matching the dictation grammar.

Dictation grammars may be optimized for particular kinds of text. Often a dictation recognizer may be available with dictation grammars for general purpose text, for legal text, or for various types of medical reporting. In these different domains, different words are used, and the patterns of words also differ.

A dictation recognizer in the Java Speech API supports a single dictation grammar for a specific domain. The application and/or user selects an appropriate dictation grammar when the dictation recognizer is selected and created.

5.3.2.3 Limitations of Speech Recognition

The two primary limitations of current speech recognition technology are that it does not yet transcribe free-form speech input, and that it makes mistakes. The previous sections discussed how speech recognizers are constrained by grammars. This section considers the issue of recognition errors.

Speech recognizers make mistakes. So do people. But recognizers usually make more. Understanding why recognizers make mistakes, the factors that lead to these mistakes,

and how to train users of speech recognition to minimize errors are all important to speech application developers.

The reliability of a speech recognizer is most often defined by its *recognition accuracy*. Accuracy is usually given as a percentage and is most often the percentage of correctly recognized words. Because the percentage can be measured differently and depends greatly upon the task and the testing conditions it is not always possible to compare recognizers simply by their percentage recognition accuracy. A developer must also consider the seriousness of recognition errors: misrecognition of a bank account number or the command "delete all files" may have serious consequences.

The following is a list of major factors that influence recognition accuracy.

- Recognition accuracy is usually higher in a quiet environment.
- Higher-quality microphones and audio hardware can improve accuracy.
- Users that speak clearly (but naturally) usually achieve better accuracy.
- Users with accents or atypical voices may get lower accuracy.
- Applications with simpler grammars typically get better accuracy.
- Applications with less *confusable* grammars typically get better accuracy. Similar sounding words are harder to distinguish.

While these factors can all be significant, their impact can vary between recognizers because each speech recognizer optimizes its performance by trading off various criteria. For example, some recognizers are designed to work reliably in high-noise environments (e.g. factories and mines) but are restricted to very simple grammars. Dictation systems have complex grammars but require good microphones, quieter environments, clearer speech from users and more powerful computers. Some recognizers adapt their process to the voice of a particular user to improve accuracy, but may require training by the user. Thus, users and application developers often benefit by selecting an appropriate recognizer for a specific task and environment.

Only some of these factors can be controlled programmatically. The primary application-controlled factor that influences recognition accuracy is grammar complexity. Recognizer performance can degrade as grammars become more complex, and can degrade as more grammars are active simultaneously. However, making a user interface more natural and usable sometimes requires the use of more complex and flexible grammars. Thus, application developers often need to consider a trade-off between increased usability with more complex grammars and the decreased recognition accuracy this might cause. Most recognition errors fall into the following categories:

- *Rejection*: the user speaks but the recognizer cannot understand what was said. The outcome is that the recognizer does not produce a successful recognition result. In the Java Speech API, applications receive an event that indicates the rejection of a result.

- *Misrecognition*: the recognizer returns a result with words that are different from what the user spoke. This is the most common type of recognition error.

- *Misfire*: the user does not speak but the recognizer returns a result.

5.3.3 Implementation Details

The Java Speech API has been used along with JSML as described in the previous section. The speech engine that has been used is the IBM ViaVoice. The Java Speech API provided by Sun Microsystems Inc. being only a specification has only the infrastructure of the API. Most of the classes provided by the API are interfaces. IBM ViaVoice also provides a speech API based on the Sun Microsystems' standards. Available as ***IBM's "Speech for Java",*** it is built on the infrastructure of the JSAPI.

- *Description:* Implementation based on IBM's ViaVoice product, which supports continuous dictation, command and control and speech synthesis. It supports all the European language versions of ViaVoice -- US & UK English, French, German, Italian and Spanish -- plus Japanese.

- *Requirements:* JDK 1.1.7 or later or JDK 1.2 on Windows 95 with 32MB, or Windows NT with 48MB. Both platforms also require an installation ViaVoice 98.

The current implementation has support for American English and facilitates the user to dictate to the Text chat tool instead of typing into it. The functionality can be extended to other components and also the whole collaboration environment by adding appropriate parsing files.

Chapter 6

6.0 Conclusion

This research consisted of three main goals. The initial goal was to develop a three dimensional meeting environment to replace the earlier two dimensional environment through which most of the collaboration process was being effected. The three dimensional meeting environment has been implemented, but not all features pertaining to the earlier two dimensional meeting environment could be ported to the 3D environment. Therefore both the 2D and the 3D environment have been retained. The 3D environment still has its functionality of showing gestures like nodding the head and hand raising. This three dimensional meeting environment will provide as a substitute to Video for low bandwidth users, thereby providing the users gesture information. The current 2D meeting interface loads images of the users as the users logon to the meeting. Applying an image mapping algorithm, the faces of the avatars can be mapped with the corresponding images to make it more realistic.

The audio and video components add most of the multimedia support for CAIRO. Being java based, this is one multimedia application that works cross platform with respect to Sun machines, Windows machines and SGI's. Though Java based programs can be expected to be truly platform independent, there is a limitation in this case because

of diverse hardware features between different platforms with respect to capture devices. Client based multimedia programs developed using the Java Media Framework are truly platform independent as long as they are receiving media streams. The notion of pure java breaks down when it comes to capture of media data. Future research in this field can be focused at writing faster and more efficient codecs so as to enhance the features provided by the Java Media Framework.

The Speech Recognition Component is the third part of this research effort. Implementation with respect to this component is very limited, but there is scope for a lot of future research in this area. So far implementation has been done limited to the user being able to dictate to the text chat tool instead of actually typing in it. Further research can be conducted at different speech recognition and synthesis engines that are compatible with the Java speech API. One area that can then be pursued would be to make the whole CAIRO application voice activated. Another area that is worth while researching would be at translation engines thereby facilitate multi-lingual collaboration. This component clubbed with appropriate speech recognition engines would ideally result in a situation where a client at one end speaks out what he wants to say at his end. CAIRO does the recognition, and just sends the text to the other user's end. Thus information transfer is sped because only textual messages are exchanged instead of bandwidth-hogging audio data. At the end of the other user CAIRO will identify the language of the user, use a translator to convert the textual information to the appropriate language and then synthesize it to voice using an appropriate speech-synthesizing engine.

In conclusion this research has provided insight into the implementation of a multimedia infrastructure for CAIRO. Further research needs to be done in the area of how each of these areas can be streamlined and made more efficient. Work also needs to be done to extend the abilities of each of the multimedia components identified in order to further enhance the collaboration effort.

Bibliography

[Abdel-Wahab, et. al. 1997] Abdel-Wahab, H., Kvande, B., Kim, O. and Favreau, J. "An Internet Collaborative Environment for Sharing Java Applications." *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, October 29-31, 1997, pp. 112-117. Tunis, Tunisia.

[Ali-Ahmad 1997] Ali-Ahmad, W. "Adaptable Software Agent for Retrieval and Presentation of Information in Collaborative Engineering Environments." M.S. Thesis, September, 1997, Massachusetts Institute of Technology.

[Bajaj, et. al. 1995] Bajaj, Chandrajit, Zhang, Peinan, Chaturvedi, Alok. "Brokered Collaborative Infrastructure for CSCW." *Proceedings of the Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1995, pp. 207-213. Los Alamitos, CA, USA.

[Benjamin 1998] Benjamin, Karim A. "Defining Negotiation Process Methodologies for Distributed Meeting Environments." MEng Thesis, May 1998, Massachusetts Institute of Technology.

[Chang, et. al. 1997] Chang, C., Quek, F., Cai, L., Kim, S. "A Research on Collaboration Net." *Interactive Distributed Multimedia Systems and Telecommunication Services*, September 1997, pp. 228-233. Darmstadt, Germany.

[Fu 1999] Fu, Luke Tan-Hsin. "Adaptation of CAIRO Meeting Environment Toward Military Collaboration Efforts." B.S and M.Engg. Thesis, May 1999, Massachusetts Institute of Technology.

[Hussein 1995] Hussein, Karim. "Communication Facilitators for a Distributed Collaborative Engineering Environment." B.S. Thesis, May 1995, Massachusetts Institute of Technology.

[Java3D, 1999] "Java3D API Specification." URL: http://java.sun.com/products/java-media/3D/forDevelopers/j3dguide/j3dTOC.doc.html. Sun Microsystems, Inc., Version 1.1.2, June 1999.

[JMedia, 2000] "Java Media API's." URL: http://java.sun.com/products/java-media/. Sun Microsystems, Inc., May 2000.

[JMF, 1999] "Java Media Framework API Guide." URL: http://java.sun.com/products/java-media/jmf/2.0/jmf20-fcs-guide/JMFTOC.html. Sun Microsystems, Inc., JMF2.0 FCS, November 1999.

[JSAPI, 1998] "Java Speech API Programmer's Guide." URL: http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-guide/index.html. Sun Microsystems, Inc., Version 1.0, October, 1998.

[LAB 1999] "Electronic Collaboration: A practical guide to educators" URL: http://www.lab.brown.edu/public/pubs/collab/elec-collab.pdf The LAB at Brown University, 1999.

[Lauff & Gellersen 1997] Lauff, M., Gellersen, H. "Multimedia Client Implementation on Personal Digital Assistants." *Interactive Distributed Multimedia Systems and Telecommunication Services*, September 1997, pp. 283-295. Darmstadt, Germany.

[Maybury 1997] Maybury, Mark T. "Distributed, Collaborative, Knowledge Based Air Campaign Planning." URL:http://www.mitre.org/support/papers/dc_kkb. MITRE, November 1997.

[MMC 1996] "Multimedia Collaboration" URL: http://www.prz.tu-berlin.de/docs/html/MMC Technical University Berlin, 1996.

[Mong Le 1998] "Desktop Video-conferencing: A Basic Tool for the 21st Century's Communications Technology" URL: http://www.vacets.org/vtic97/tmle.htm. Computing Engineering, Inc., June 1998.

[Novatel Wireless, Inc. 1999] "Corporate Profile." URL: http: //www.novatelwireless.com/html/profile.htm. Novatel Wireless, Inc., May 13, 1999.

[Pena-Mora, et. al. 1996] Pena-Mora, F., Ali-Ahmad, W., Chen, H., Hussein, K., Kennedy, J., Soibelman, L., and Vadhavkar, S. "The Da Vinci Agent Society Initiative Progress Report as Fall 1996." *IESL Report No. 96-06*, Intelligent Engineering Systems Laboratory, Engineering System Group, Henry L. Pierce Laboratory, Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, November, 1996.

[Roure 1996] Roure, D. D., Hall W., Davis H., Dale J. "Agents for Distributed Multimedia Information Management." University of Southampton, UK, 1996.

[RVideo 2000] "Remtec Video Products – Remtec Video Workstation." URL: http://www.rst.fi. Remtec Systems, Finland.

[Solari et. al. 1999] Solari, J., Vedam, P. "CDI Software Design Document." Massachusetts Institute of Technology, March, 1998.

[Sun Microsystems, Inc. 1999] "Java Development Kit." URL: http://www.javasoft.com/products/jdk/1.1/index.html. Sun Microsystems, Inc., May 13, 1999.

[VTEL 1998] "VTEL WG500 Work Group Conferencing System" URL: http://www.vtel.com/newsinfo/resource/products/wg500g.pdf. VTEL Corp., September 1998.

[Woo 1994] Woo, T. K., Rees M. J. "A Synchronous Collaboration Tool for World-Wide Web." URL:http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/CSCW/rees/SynColTol.htm. The University of Queensland, Queensland, 1994.