INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6” x 9” black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600
NOTE TO USERS

The original manuscript received by UMI contains pages with slanted print. Pages were microfilmed as received.

This reproduction is the best copy available

UMI
CORBA-BASED MIDDLEWARE FRAMEWORK FOR DISTRIBUTED MULTIMEDIA SYSTEMS

by

Pinkesh Jethalal Shah

Copyright ©Pinkesh Jethalal Shah 1998

A Dissertation Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
DOCTOR OF PHILOSOPHY
In the Graduate College
THE UNIVERSITY OF ARIZONA
1998
As members of the Final Examination Committee, we certify that we have read the dissertation prepared by Pinkesh J. Shah entitled **CORBA-based Middleware Framework for Distributed Multimedia Systems** and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

---

Dr. Ralph Martinez
Date: 3/2/98

Dr. Bernard P. Zeigler
Date: 3/2/98

Dr. Jay Cock
Date: 2/26/98

Dr. Gregory R. Andrews
Date: 3/3/98

Dr. John H. Hartman
Date: 3/4/98

---

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copy of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Dissertation Director Dr. Ralph Martinez
Date: 3/2/98
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under the rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

SIGNED: [Signature]
I enjoyed my research experience at the University of Arizona. The Electrical and Computer Engineering Department and the Computer Science Department has an excellent faculty and educational facilities.

I pursued my doctoral work under the guidance, encouragement, and support of my advisor, Dr. Ralph Martinez. I am really grateful to him.

I would like to thank Dr. Bernard P. Zeigler, Dr. Gregory R. Andrews, Dr. John H. Hartman, Dr. Jay Cook, Dr. Indra Widjaja, and Dr. Max Liu for their time and effort in guiding me. Thanks to my family for all their support and encouragement during the course of my study.

I admire all the researchers for their contribution to the advancement of science and technology.
DEDICATION

To my wife Pina, my father Jayendra, my mother Veena, and my sister Purvi.
# TABLE OF CONTENTS

## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>

## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Problem Statement & Motivation ........................................ 14
1.2 Objective ............................................................... 17
1.3 Approach ................................................................. 19
1.4 Related Work in Multimedia
    1.4.1 Multimedia Architectures ..................................... 20
    1.4.2 Multimedia Related Protocols ................................. 42

## 2 Multimedia in Distributed Collaborative Systems

2.1 Example System Applications ........................................... 64
    2.1.1 Remote Consultation & Diagnosis (RCD) ...................... 64
    2.1.2 Java enabled Interactive Environment (JIVE) .............. 67
2.2 Application System Requirements ...................................... 69
2.3 Isochronous Media ..................................................... 70
    2.3.1 Audio .................................................................. 71
    2.3.2 Video .................................................................. 73
2.4 Non-isochronous media ................................................... 81
    2.4.1 Image .................................................................. 81
    2.4.2 Text ................................................................... 81
    2.4.3 Annotation ......................................................... 82
2.5 Quality of Service (QoS) .................................................. 82
    2.5.1 Qualitative .......................................................... 82
    2.5.2 Quantitative ........................................................ 83
    2.5.3 QoS in IPv6 .......................................................... 83
    2.5.4 QoS in ATM .......................................................... 111

## 3 Review of CORBA Middleware Protocols

3.1 Object Management Architecture (OMA) ................................. 113
    3.1.1 Object Services ...................................................... 114
    3.1.2 Common Facilities ................................................ 116
    3.1.3 Domain Interfaces ................................................... 117
    3.1.4 Application Interfaces ............................................. 117
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Distributed multimedia environment</td>
<td>15</td>
</tr>
<tr>
<td>1.2</td>
<td>Distributed multimedia middleware framework layers</td>
<td>18</td>
</tr>
<tr>
<td>1.3</td>
<td>TINA distributed processing environment</td>
<td>22</td>
</tr>
<tr>
<td>1.4</td>
<td>H.323 terminal architecture</td>
<td>27</td>
</tr>
<tr>
<td>1.5</td>
<td>H.323 interoperability architecture</td>
<td>28</td>
</tr>
<tr>
<td>1.6</td>
<td>DAVIC system representation</td>
<td>31</td>
</tr>
<tr>
<td>1.7</td>
<td>DAVIC core functional groups</td>
<td>32</td>
</tr>
<tr>
<td>1.8</td>
<td>DAVE component architecture</td>
<td>36</td>
</tr>
<tr>
<td>1.9</td>
<td>Multimedia System Services overview</td>
<td>42</td>
</tr>
<tr>
<td>1.10</td>
<td>ATM and B-ISDN reference model</td>
<td>44</td>
</tr>
<tr>
<td>1.11</td>
<td>ATM User to Network Interface cell</td>
<td>48</td>
</tr>
<tr>
<td>1.12</td>
<td>ATM Network to Network Interface cell</td>
<td>49</td>
</tr>
<tr>
<td>1.13</td>
<td>ATM switch with VPI's and VCI's</td>
<td>50</td>
</tr>
<tr>
<td>1.14</td>
<td>IPv4 packet</td>
<td>53</td>
</tr>
<tr>
<td>1.15</td>
<td>IPv6 packet header</td>
<td>55</td>
</tr>
<tr>
<td>1.16</td>
<td>TCP packet</td>
<td>58</td>
</tr>
<tr>
<td>1.17</td>
<td>UDP packet</td>
<td>59</td>
</tr>
<tr>
<td>1.18</td>
<td>MBONE topology</td>
<td>61</td>
</tr>
<tr>
<td>2.1</td>
<td>RCD session between physicians</td>
<td>65</td>
</tr>
<tr>
<td>2.2</td>
<td>Bandwidth requirement for digital audio</td>
<td>72</td>
</tr>
<tr>
<td>2.3</td>
<td>Block diagram for various types of JPEG coding</td>
<td>76</td>
</tr>
<tr>
<td>2.4</td>
<td>2-D matrix of DCT coefficients</td>
<td>78</td>
</tr>
<tr>
<td>2.5</td>
<td>JPEG zigzag ordering</td>
<td>78</td>
</tr>
<tr>
<td>2.6</td>
<td>MPEG frame sequence</td>
<td>79</td>
</tr>
<tr>
<td>2.7</td>
<td>Source initiated resource reservation in a multicast tree</td>
<td>98</td>
</tr>
<tr>
<td>2.8</td>
<td>Receiver initiated resource reservation in a multicast tree</td>
<td>100</td>
</tr>
<tr>
<td>2.9</td>
<td>State information at a router</td>
<td>107</td>
</tr>
<tr>
<td>3.1</td>
<td>OMA reference model</td>
<td>114</td>
</tr>
<tr>
<td>3.2</td>
<td>CORBA Components</td>
<td>118</td>
</tr>
<tr>
<td>3.3</td>
<td>A path of static request in CORBA</td>
<td>123</td>
</tr>
<tr>
<td>4.1</td>
<td>Software design steps</td>
<td>130</td>
</tr>
<tr>
<td>4.2</td>
<td>DMMF MediaDevice factory architecture</td>
<td>134</td>
</tr>
<tr>
<td>4.3</td>
<td>Media object lifecycle</td>
<td>134</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>4.4</td>
<td>Relationship between stream and flow</td>
<td>135</td>
</tr>
<tr>
<td>4.5</td>
<td>A typical stream</td>
<td>137</td>
</tr>
<tr>
<td>4.6</td>
<td>MediaDevice interface inheritance</td>
<td>138</td>
</tr>
<tr>
<td>4.7</td>
<td>StreamEndPoint and VDev interface inheritance</td>
<td>139</td>
</tr>
<tr>
<td>4.8</td>
<td>Stream interface inheritance</td>
<td>141</td>
</tr>
<tr>
<td>4.9</td>
<td>Message flows during bind_devs operation</td>
<td>144</td>
</tr>
<tr>
<td>4.10</td>
<td>Connect operation steps</td>
<td>144</td>
</tr>
<tr>
<td>4.11</td>
<td>Audio packet format</td>
<td>147</td>
</tr>
<tr>
<td>4.12</td>
<td>Video packet format</td>
<td>148</td>
</tr>
<tr>
<td>5.1</td>
<td>Process, threads, and objects at a node</td>
<td>159</td>
</tr>
<tr>
<td>5.2</td>
<td>MicDeviceImpl object diagram</td>
<td>161</td>
</tr>
<tr>
<td>5.3</td>
<td>SpkDeviceImpl object diagram</td>
<td>163</td>
</tr>
<tr>
<td>5.4</td>
<td>CamDeviceImpl object diagram</td>
<td>164</td>
</tr>
<tr>
<td>5.5</td>
<td>DispDeviceImpl object diagram</td>
<td>165</td>
</tr>
<tr>
<td>5.6</td>
<td>MicVDevImpl object diagram</td>
<td>166</td>
</tr>
<tr>
<td>5.7</td>
<td>SpkVDevImpl object diagram</td>
<td>167</td>
</tr>
<tr>
<td>5.8</td>
<td>CamVDevImpl object diagram</td>
<td>168</td>
</tr>
<tr>
<td>5.9</td>
<td>DispVDevImpl object diagram</td>
<td>169</td>
</tr>
<tr>
<td>5.10</td>
<td>MicStreamEndPointImpl object diagram</td>
<td>170</td>
</tr>
<tr>
<td>5.11</td>
<td>Microphone credit_listen thread object diagram</td>
<td>170</td>
</tr>
<tr>
<td>5.12</td>
<td>Source credit listen thread algorithm</td>
<td>171</td>
</tr>
<tr>
<td>5.13</td>
<td>Source datasend thread (Talkmic) object diagram</td>
<td>172</td>
</tr>
<tr>
<td>5.14</td>
<td>Source datasend thread algorithm</td>
<td>174</td>
</tr>
<tr>
<td>5.15</td>
<td>SpkStreamEndPointImpl object diagram</td>
<td>175</td>
</tr>
<tr>
<td>5.16</td>
<td>Speaker data receive thread object diagram</td>
<td>176</td>
</tr>
<tr>
<td>5.17</td>
<td>Algorithm for sink receive thread</td>
<td>177</td>
</tr>
<tr>
<td>5.18</td>
<td>Object diagram for speaker (sink) play thread</td>
<td>178</td>
</tr>
<tr>
<td>5.19</td>
<td>Algorithm for media play thread in sink</td>
<td>179</td>
</tr>
<tr>
<td>5.20</td>
<td>CamStreamEndPointImpl object diagram</td>
<td>180</td>
</tr>
<tr>
<td>5.21</td>
<td>Camera credit_listen thread object diagram</td>
<td>181</td>
</tr>
<tr>
<td>5.22</td>
<td>Design of Sendcam object that sends video</td>
<td>182</td>
</tr>
<tr>
<td>5.23</td>
<td>Design of DispStreamEndPointImpl object</td>
<td>183</td>
</tr>
<tr>
<td>5.24</td>
<td>Design of Listendisp object that receives video</td>
<td>184</td>
</tr>
<tr>
<td>5.25</td>
<td>Object diagram of DispFrame class</td>
<td>185</td>
</tr>
<tr>
<td>5.26</td>
<td>Object diagram of MyFrame class</td>
<td>186</td>
</tr>
<tr>
<td>5.27</td>
<td>Design of MCanvas class</td>
<td>187</td>
</tr>
<tr>
<td>5.28</td>
<td>Object diagram of audio stream control</td>
<td>189</td>
</tr>
<tr>
<td>5.29</td>
<td>Object diagram of video stream control</td>
<td>190</td>
</tr>
</tbody>
</table>
5.30 LAN environment .............................................. 192
5.31 Prototype application client messages in DMMF ............. 192
5.32 Prototype application graphical user interface ............... 194
5.33 Video display in DMMF ....................................... 195

6.1 Protocol stack for performance evaluation ..................... 197
6.2 WAN environment .............................................. 198
6.3 Algorithm to measure media delay ............................. 198
LIST OF TABLES

2.1 Bandwidth requirements for digital video .............................................. 75
5.1 Mapping of IDL basic types to JAVA types ............................................. 155
5.2 Mapping of IDL constructed types to JAVA types ..................................... 155

6.1 Time to get ref. to fac. on gpacs1 machine bound 1st .......................... 201
6.2 Time to get ref. to fac. on gpacs1 machine bound 2nd .......................... 202
6.3 Time to get ref. to fac. on pcpacs2 machine bound 1st .......................... 202
6.4 Time to get ref. to fac. on pcpacs2 machine bound 2nd .......................... 203
6.5 Time to create object on gpacs1 machine bound 1st ............................ 204
6.6 Time to create object on gpacs1 machine bound 2nd ............................ 204
6.7 Time to create object on pcpacs2 machine bound 1st ............................ 205
6.8 Time to create object on pcpacs2 machine bound 2nd ............................ 205
6.9 Time to bind 2 devices ............................................................................ 205
6.10 Time for destroy operation to return ..................................................... 205
6.11 Time for start operation to return ........................................................ 206
6.12 Time for stop operation to return ......................................................... 206
6.13 Delay from media record to play in LAN .............................................. 206
6.14 Performance results for WAN .............................................................. 208
ABSTRACT

In recent years, computer and communication technology has advanced rapidly. Advancement in computing has created powerful multimedia capable computers, and advancement in communications has made it possible to connect these computing devices using diverse networks across geographically distributed areas. Currently, most of the multimedia capability of computers is used in a stand-alone mode. There are emerging applications such as distance learning, telemedicine, group collaboration, video conferencing, etc. that require use of distributed multimedia capability. Currently, a very limited number of tools exists that facilitate creation of distributed multimedia applications. This dissertation presents architecture and implementation of a middleware framework for distributed multimedia systems. The framework extends CORBA request-reply functionality by providing streaming services for audio and video data. The CORBA distributed object environment is used for control and interaction with media objects. Media devices negotiate the use of transport protocols and media formats at bind time. The framework uses JAVA to accommodate the heterogeneity of various computing platforms. Prototype applications that use the framework are described. Performance results of the framework in a LAN and a WAN environment are shown.
CHAPTER 1

Introduction

1.1 Problem Statement & Motivation

In recent years, there has been a tremendous growth in the computer and communications field. Speed of the computer Central Processing Unit (CPU) is doubling every 18 months. There has been a dramatic decrease in the price of other computer components such as RAM and hard disk drives. This has made personal computers (PC) affordable to the common public. Most of the current PC's are multimedia capable, meaning they come with sound card and speakers. Some also have a video camera and a video compression card. These PC's are used in the stand alone mode. The multimedia capability of the personal computers is rarely used in the networked environment.

With the advancement of communication technology, it is possible to connect computers or other computing devices locally and globally [99]. Computer networking technology allows for connecting computers together [82]. Computer networks also vary in the type of protocol they use and the type of service they provide. Some are
high bandwidth and some are low bandwidth. Internet has been growing exponen­
tially and the World Wide Web is immensely popular. There is a need to include
multimedia content that is easy to use on the web.

A computing device can vary from handheld wireless computers to palmtop com­
puters to personal computers to supercomputers. These various devices have different
hardware architecture and they run different types of software. Media devices in these
computers also vary in terms of their capability, such as media format, processing
capability, etc.

We are dealing with the distributed and heterogeneous environment of computing
devices and media devices within the computers. Fig. 1.1 shows such a heterogeneous
distributed multimedia environment. There are growing number of applications, such as group collaboration, distance learning, and telemedicine that can use the networked multimedia capability [21, 91, 90, 97, 98]. These applications require sending a media, such as audio or video, from one computing device to other. To develop these applications, a developer has to be concerned with networking capabilities, computing platforms heterogeneity, and media device capability. It is very cumbersome to deal with all these issues for every application. There is a need for a middleware framework, that let developers create distributed multimedia applications easily and enables reuse of previously developed software components.

Object-oriented analysis and design methodology treats a software system as a collection of objects. This methodology enables reuse of previously developed software components. Each object is autonomous. An object has well-defined interface and precise behavior. Group of objects collaborate and cooperate to make the whole system work.

The work done in [5] accesses media data via a parameter in an operation, so the media data are subject to ORB marshalling and unmarshalling. Data have to travel using the protocol used by ORB, so other protocols cannot be used for data transfer. Each media packet incurs a round-trip communication delay rather than one-way delay from a source to a sink. DAVE project [7] does not take advantage of standard distributed object framework such as CORBA. Plugging of a camera device to a speaker is allowed, so no negotiation is done of flow types or media formats.
Project described in [4] extends CORBA Interface Definition Language (IDL), hence the existing ORBs cannot support the framework. Media data are passed as events through an ORB, so this approach has more overhead than bypassing the ORB.

1.2 Objective

The objective of this dissertation was to develop distributed multimedia middleware framework (DMMF), implement it, and measure the performance of such a framework. The middleware framework had to have the following characteristics:

- Provide platform independence.
- Support heterogeneity of the media devices.
- Provide abstraction of network communication.
- Support multimedia streams and Quality of Service (QoS). QoS describes an application's requirement for its data. QoS parameters are required to accommodate real-time, non-real-time, delay sensitive, or loss sensitive applications.
- Leverage the benefits of object-oriented analysis and design methodology to develop a modular framework. Components of the framework should be inter-changeable.
Figure 1.2: Distributed multimedia middleware framework layers

- Scalability and extensibility. New components can be added in the framework and the existing one can be modified as needed without affecting the applications.

- Provide dynamic binding of media objects.

- Allow access to multimedia information from anywhere and at anytime.

To measure and analyze the performance of the multimedia middleware framework, it was necessary to develop a prototype application and gather performance data in local area network (LAN) and wide area network (WAN) environment.
1.3 Approach

To accomplish the stated objectives, Common Object Request Broker Architecture (CORBA) middleware protocols from Object Management Group (OMG) were used to support distributed object based environment [9, 12]. CORBA provides various services in the middleware and uses request-reply semantics. When an operation is invoked on an object, that object performs the operation and returns the result. Some multimedia data need to be sent continuously in time, so CORBA was used for control and management of media objects in the distributed environment. Multimedia data are transported outside of CORBA using network transport protocols as shown in Fig. 1.2. Distributed multimedia architectures were investigated and DMMF was created based on OMG specification "Control and management of audio and video streams" [8]. Computer and communication technology is converging, and there is an increasing trend in the telecommunications industry to use distributed objects for network control and management [82]. The developed multimedia framework will be able to communicate with future telecommunication objects for network services for multimedia applications. Object-oriented methodology was used to create modular and extensible framework. To provide interoperability among heterogeneous computing platforms the JAVA programming environment was used. JAVA to Native Interface (JNI) was used where appropriate JAVA functionalities were not
available. Once the DMMF was developed, it was used on different computing platforms. Streaming protocol was developed to transport multimedia data from one computing device to other in a networked environment.

1.4 Related Work in Multimedia

1.4.1 Multimedia Architectures

In the last eight years, several approaches have been developed for distributed multimedia architecture. This section summarizes this work.

1.4.1.1 TINA-C

Telecommunications Information Networking Architecture Consortium (TINA-C) [40, 30] is formed by leading network operators, telecommunications equipment manufacturers and computer manufacturers to define a common software architecture for multimedia and information services. Traditional telecommunication is based on voice communication, while the Internet is based on data communications. Media, such as audio and video, can be digitized and sent over a data communications network. The Internet technology is based on intelligent user terminals, while in the traditional telecommunications, the network is made intelligent with dumb end-user terminals. In the current Internet, the services are developed using the power and software capability of the user terminals; so, it is very flexible to develop new services. The underlying model of the Internet is best-effort delivery. Traditional
telecommunications offer high security, reliability, and quality-of-service. The goal of the TINA-C is to create a software architecture that integrates the Internet and traditional telecommunication networks so that versatile multimedia services can be provided over both networks.

TINA is applied to all parts of telecommunications and information systems, such as end-user terminals (PC and other network devices), transport servers (switches, routers, etc.), service provider servers (Video on Demand, news, etc.), and management servers (billing, network management, security, etc.). TINA software architecture is based on four principles.

- Object-oriented analysis: To break the complex system into different models.

- Distribution: To distribute service software components so that network load can be managed, increase network survivability, and meet specific customer demand.

- Decoupling: To make software components independent, so that each can be changed without affecting the others.

- Separation: To separate policy and mechanism in providing multimedia services.

TINA architecture is divided into 3 sub-architectures:

- TINA Computing Architecture
Fig. 1.3 shows the distributed processing environment (DPE) that supports TINA and hides the heterogeneity of the various equipments in the network. Using DPE, distribution (location, access, etc.) transparency is assured. TINA computational objects can interact with each other on an object level without worrying about the underlying platform specific communication details. DPE services and facilities can be distributed using Kernel Transport Network (kTN). This DPE is based on CORBA so that DMMF should be able to access multimedia services provided by TINA.

- TINA Service Architecture

The Service Architecture uses the notion of a session to capture the information used by all processes in the TINA architecture, which are involved in providing
a particular service for a certain duration. For example, if an application requested a particular QoS connection from TINA, the described QoS parameters will become part of the session. Some of the particular sessions are:

- Access session: This is used for establishment of a connection from the user to a system using specified terms and conditions.
- Service session: This session provides the service itself and manages the overall performance of the session.
- Communication session: This session provides an abstraction of the actual transport network connections.

Objects that provide services are categorized into generic objects, which provide common service functionalities and service-specific objects.

• TINA Network Architecture

This architecture is used to set-up connections and manage telecommunication networks in a technologically independent way. It consists of three layers. The communication layer provides interfaces that the service components can use to manage end-to-end communications. The connectivity layer provides services to the communication layer. It provides technologically-independent interfaces that the communication layer can use to internetwork different technologies to provide end-to-end connection. The third layer is the Network layer, which abstracts a specific networking technology.
TINA defines multimedia services from the perspective of the network service providers. DMMF needs to use the services provided by the communication network to stream multimedia data. TINA is still being defined and standardized. When these services become available, they can be integrated into DMMF easily because they both use the CORBA-based distributed object computing model and because of the modular design of DMMF.

1.4.1.2 H.323

H.323 [84] is a recommendation developed by the International Telecommunication Union (ITU) for "Visual Telephone Systems and Equipment for Local Area Networks which provide a Non-Guaranteed Quality of Service." It is intended to provide multimedia communication over a packet-switched network that does not provide any service guarantee. H.323 specifies components, procedures, and protocols that are needed to provide audio, video and data communication in a packet switched network. It is required that the network provides reliable and unreliable transport mechanisms. H.323 terminals can communicate through hubs, routers, bridges, and dial-up connections via any network topology. This recommendation provides various levels of multimedia communications. These levels include different types of media combinations, such as voice only, voice and data, voice and video, or voice, video, and data. Various multimedia applications such as multimedia conferencing can be developed.
that use this standard to communicate. H.323 allows interoperability between the following terminal types using common recommendations, procedures, and messages.

- H.320 terminals on narrowband integrated services digital network (N-ISDN).
- H.321 terminals on broadband ISDN.
- H.322 terminals on IsoEthernet network.
- H.324 terminals operating on plain old telephone network.
- H.310 terminals on asynchronous transfer mode (ATM) networks.

H.323 defines the multimedia terminal specification for end-user equipment, a gatekeeper specification for conference admission, a multipoint controller and multipoint processor specification for multipoint communication, and a gateway specification for interoperability with other types of network. The H.323 specifies system and component descriptions, call model descriptions, and call signaling procedures. It refers to many other ITU-T (Telecommunications standard sector) recommendations such as H.225.0 for audio/video packetization, media stream synchronization, control stream packetization, and control message formats; H.245 for messages and procedures for mode requests, control, indications, terminal capability exchange, and opening/closing logical channels for different media; G.711, G.722, G.728, G.723.1, and G.729 for audio coding; H.261 QCIF, H.261 CIF, and H.263 for video coding; and T.120 for data communications. Streaming protocol in H.323 is based on Real-Time
Protocol (RTP) and Real-Time Control Protocol (RTCP). Each of the component is presented below in detail.

- **H.323 Terminal**

  Fig. 1.4 shows the block diagram of the H.323 terminal. It provides real-time bidirectional audio, video, and data communications. Call signaling, control messages, multiplexing, audio codecs, video codecs, and data protocols are defined for the standard. Specific applications or media equipment is not specified, but the codec standards are. Using capability exchange, appropriate audio or video coding standard is determined. If the two terminals do not have common codec support, an intermediate may be required to transcode media streams in the gateway. Receive path delay is optional for audio and video streams; it can be used for jitter control.

- **H.323 Multipoint Control Unit (MCU)**

  Several different conferencing modes are defined in the H.323 recommendation. This is one of the major difference between other ITU-T terminal types and H.323 terminal. Point-to-point, multipoint, and broadcast conference modes are defined. Point-to-point is one-to-one, multipoint is many-to-many, and broadcast is one-to-many conference mode. MCU is divided into two units. The Multipoint processor (MP) performs media processing, such as video mixing,
audio mixing, or video switching. The Multipoint controller (MC) is used for conference control, such as establishing a communication mode and media channels. MC is required for all multipoint conference types. MC can be located in terminal, gateway, or gatekeeper.

- Gateways for H.323

Gateways are used to provide interoperability between H.323 and other types of terminals. How a gateway provides interoperability is shown in Fig. 1.5.
Gateway does transcoding if necessary; it does translation of call signaling, control channel messages, and multiplexing to provide interoperability. H.323 terminals are not restricted to transmission rates that are multiples of 64 kbps. A gateway can have any number and type of interfaces. It may be able to support several concurrent calls between local area networks (LAN) and circuit switched network (CSN). A gateway may incorporate MCU within itself to provide multipoint conferences with several terminals on and off the LAN.

Figure 1.5: H.323 interoperability architecture

- H.323 Gatekeeper
A gatekeeper provides admission control service for the multimedia H.323 traffic on the LAN. This admission control mechanism allows an administrator to control the amount of H.323 traffic in the network. A permission from the gatekeeper is required to place or accept a call. A request and permission includes the amount of bandwidth that will be utilized by a particular connection so that the gatekeeper can keep track of the available bandwidth resources. H.323 does not define a specific mechanism for call control; so, it is left to the manufacturer's discretion. A gatekeeper provides an address translation between an external telephone number and a network address; hence, phone numbers can be used to communicate between terminals regardless of their network addresses.

Call setup in H.323 is done two ways. In the first way called direct call signaling, endpoints signal among each other to establish a call. The second way involves a gatekeeper. A gatekeeper relays all call signaling between the end terminals. This allows multipoint conferencing when the terminals do not contain a MC. H.323 uses multiple logical channels for various types of communication. There is a channel for H.245. Audio, video, data, and RTCP are carried over different channels. By using separate logical channels by means of different transport addresses, multiplexing and demultiplexing overhead is avoided.

H.323 is a lower layer mechanism than DMMF. It is a protocol specification and not a middleware; therefore, control and communication is done by the lower layer
protocols. No programming framework is defined that easily allows creation of distributed multimedia systems. It is specifically designed for networks that do not support QoS. So, it does not cover all the network types possible; hence, it is not possible to select network protocols and type on the fly at run-time. Audio and video coding formats are standardized in H.323, but because they are at the lower level in the protocol stack, changes and additions may require considerable work. There is no object based framework. H.323 can be one of the protocols in DMMF.

1.4.1.3 Digital Audio Video Council (DAVIC)

DAVIC is an organization that develops standards for digital audio/video broadcast and interactive applications [85, 77], so that these applications can interoperate across countries and services. The goal of DAVIC is to develop specifications of interfaces, protocols, and architectures for digital audio/video applications and services. The reference model of DAVIC is shown in Fig. 1.6. It consists of 5 major components. They are:

- Content Provider System (CPS): It deals with all the specifications related to a provider of digital audio/video system content.

- Service Provider System (SPS): SPS deals with service related infrastructure of digital audio/video system. SPS is in the middle of CPS and SCS.
• Service Consumer System (SCS): This specification has end-user system functionalities. One of the examples is a specification for a television set-top box.

• CPS-SPS Delivery System: Contains specifications on how media, control, and management data can be delivered from the content provider to the service provider system.

• SPS-SCS Delivery System: Contains specifications on how media, control, and management data can be delivered from the service provider to the service consumer.

![Figure 1.6: DAVIC system representation](image)

These components interact through interfaces. Interfaces are called reference points. As long as components conform to reference points standard, they should
be interoperable with components on the other side of the reference point. A wide variety of applications with digital audio/video content can be supported by DAVIC standard. The approach taken by DAVIC is to examine these applications and identify common core functionalities among them. These core functionalities are basic to system operation, integrity, and development. DAVIC will standardize these core functionalities and new ones can be added for future applications. The current DAVIC core functions are shown in Fig. 1.7 in a layered structure.

![DAVIC core functional groups](image)

Figure 1.7: DAVIC core functional groups

- Bit Transport

These functions deal with the physical and the data link layers of the OSI model. Logical connections are provided between communicating entities. Point-to-point, point-to-multipoint, and multipoint-to-multipoint connection capabilities are considered. Bit level data can be multiplexed between multiple applications. An application should be able to negotiate its QoS requirements.
• Session

Session functions provide common facilities, such as data encryption, file transfer, verification, and establishment of logical channels. This group uses functions provided by the bit transport group. QoS establishment criteria will be mutually agreed upon by the session and bit transport group.

• Access Control

This group of functions verify and determine access rights to the network and authenticate a user. This group controls a user's access to specific applications and its related content; it also controls access to goods and services and payment methods.

• Navigation, Program Selection and Choice

Functions in this group provide facilities to select an application or content. Selection choice is augmented by user specific criteria, such as access-control rights, user preferences, user interests, prior behavior, etc. A selection results in launch of a resident or downloaded application.

• Application Launch
These functions provide facilities to run an application. An application is downloaded from a server or it may be resident in a set-top unit. A launched application obtains system resources needed and creates and destroy sessions in an orderly and clean manner.

- Media Synchronization Links

Synchronization links between media objects, such as sound segments, subtitles, still images, and moving images are provided by these functions so that a multimedia presentation is synchronized. An example of an application that uses these functions is that a user viewing an ad in a TV broadcast may select and order the product seen in the ad.

- Application Control

Application control is different from presentation control. Based on user interaction, behavior of an application may change. Content options are selected according to user commands and preference. Examples are pause, rewind, audio pitch control, etc.

- Presentation Control

These functions provide user control of the delivery and display of multimedia information, such as positioning, subtitle activation, language, etc. This control does not involve control of the server, which is described in the application
control nor the flow of information; it just changes how the multimedia is displayed.

- Usage Data

In a commercial environment, a customer is charged for the services used. This group of functions collect, store, and supply data related to the use of resources and materials. The data is used for payment methods and provides feedback to content providers for market research and planning purposes.

- User Profile

These functions store and utilize information about individual users. This information is user supplied and learned by the system based on prior user behavior. This information is used for access control, to assist in navigation, and for correct billing for services received by a user. Demographic, socio-economic, and geographic information from user profile are used by the service and content providers.

1.4.1.4 Distributed Audio and Video Environment (DAVE)

DAVE was developed at Sandia National Labs to support development of distributed multimedia applications [7, 41]. DAVE provides a model that integrates multimedia ready computers into a distributed environment. It uses object-oriented analysis and design, multimedia technologies, and distributed computing approach
to facilitate a distributed multimedia development environment. It provides an application programming interface and abstraction of media devices using object oriented techniques. Application developers treat media devices, such as microphone and speaker, as distributed resources. Developers also define additional devices and media types and integrate them into DAVE. A programmer creates media devices on local or remote machines and then connects them using the specific calls. Once connected, the system enters the run state, thus transmitting media information. Major components of DAVE are application programming interface (API), a connection manager, an object manager, and device objects. Relationship between these components is shown in Fig. 1.8.

Figure 1.8: DAVE component architecture
• Application Programming Interface

DAVE provides a simple and intuitive programming model for application developers. It provides an interface between the connection manager and the application. With these API's, complex applications can be created using a few calls. The API component does lexical analysis of the input to check its validity and then parses the input for valid sequencing. Once checked, the input is put into a new format; it is then passed to the connection manager. Information received from the connection manager is also passed back to the application through API.

• Connection Manager

The connection manager takes care of all the non real-time functions. These functions are resource allocation, exception handling, configuration management, and state information. It is implemented as a UNIX daemon process. The connection manager is an interface between an application and the object manager. All the commands pass through the connection manager; it validates, stores, forwards, and executes the commands. A session exists between an application and the connection manager. The connection manager can participate in only one session. This is one of the limitation of DAVE. Multiple applications using different media devices on the same host are not possible.

• Object Manager
The object manager component manages objects and devices. It is responsible for creation, deletion, sampling, and configuration of objects. It keeps track of application objects and device connection. Devices that are created are part of the object manager process. The object manager gets its commands from the command manager and responds to the commands. All real-time functions are handled by the object manager. It is also implemented as a UNIX daemon process. It receives its requests through the socket interface. Exception conditions are passed to the application through the command manager.

- Devices

DAVE has a hierarchy of devices. The root object is Dev which defines basic functions that all the devices require. Network, VideoDev and AudioDev inherit from Dev. Network is further specialized into source and sink. VideoDev is specialized into CameraDev, VideoWindow, and Compression. AudioDev is specialized into SpeakerDev and MicDev.

DAVE is implemented on UNIX, so it does not support computing platform heterogeneity. Compression device has to be selected by an application developer, so it is not transparent. There are multiple device classes to differentiate devices with different compression formats. DAVE does not take advantage of a standard distributed framework, such as CORBA, so all the communication is done via sockets.
Plugging of a camera device to a speaker is allowed; therefore, it is possible to create an application that will not work at run time.

1.4.1.5 Multimedia System Services (MSS)

MSS is a middleware specification developed by Interactive Multimedia Association (IMA) for distributed multimedia system development [86]. It is based on CORBA, but MSS was developed when CORBA architecture was not fully specified. MSS uses CORBA lifecycle services, property services, and event services. The MSS specification was left unfinished. MSS provides a good starting point for middleware architecture for distributed multimedia systems. The major components of MSS are:

- Port

Port provides media data, an entrance, and an exit to a media device. A port is not directly accessed by a client. It exists to perform data movement operations to and from a media device. Ports are distinguished by an index. A processing element in a media device can have any number of ports associated to it based on its function.

- Stream

Stream is a flow of media data through a device or a connection. The stream interface and stream object allow monitoring and control of a stream. The stream interface allows a client to pause, resume, stop, and start a stream.
With the stream interface, a client has a generic way to deal with stream control, which is independent of the type of media data in the stream.

• Format

The format object provides an abstraction of the details of media formatting at a particular port. The format specification is separate from media processing and flow-control. Various audio and video formats are represented by a subclass of format object. The format object allows a client to specify details about the media encoding. In a connection establishment procedure, virtual connection interacts with the format objects associated with the ports to be connected to make sure the formats are compatible.

• Virtual device

A virtual device abstracts distributed media device resources. These devices are physical devices or software devices. This abstraction provides a client with a common environment for all devices. Virtual device handles resource management of a physical media device and all the necessary system resources. A virtual device contains a processing element, ports, formats, and stream objects. Multiple virtual devices may share a physical device. Control of stream within the device is provided by the stream object. Input/output ports and their associated formats provide an external interface to a device. The processing element in the device actually processes the media data.
• Virtual connection

The virtual connection is an abstraction for transport between the virtual devices. The actual work of sending and receiving data is done by the devices using their ports, but the virtual connection participates in establishing the connection. It negotiates network resources and device resources. A virtual connection also has a stream object to provide media transport related information. Media devices can be connected point-to-point or point-to-multipoint.

• Group

A group object allows a client to manage a dataflow graph of multiple resources as a single group. It is convenient for the client to manage all the distributed resources from a single point rather than contacting each of them individually. Virtual devices and a virtual connection can be included in a group. The group object allows a mechanism for end-to-end QoS specification and resource allocation of all the objects involved in the graph. The group object also has a stream object to provide end-to-end management of media stream.

Fig. 1.9 provides a client's view of MSS. A client will use a factory to create device objects. Then, it will use a virtual connection object to connect the virtual media devices. A virtual connection and media devices will be added to a group so that all can be controlled from just one object. DMMF has some of the MSS concepts, but
MSS does not have any media transport specified. A stream object supports only one type of media stream. Detail specification of how negotiation between devices is done for formats, transport protocol, and addresses is not specified. The specification is not complete.

1.4.2 Multimedia Related Protocols

In this section, network and transport layer protocols, which are important for multimedia communications, are described. Protocols are the set of rules or conventions that govern the ways in which two or more entities cooperate to exchange data.
Communication network architecture is made up of a stack of different layers and protocols. Each layer performs a set of specific tasks. A layer uses services of the layer below it and provides services to the layer above it. Information is switched in the communication network by using three main techniques. The first technique is circuit switching. In this technique, a fixed bandwidth circuit is established before information can be transferred. If there is nothing to send, the bandwidth resource is wasted. In the second technique message switching, information is sent as messages. Messages travel in the network from a source to a destination. Network elements inside the network first receive a whole message before forwarding it towards its destination. The third technique is packet switching. Messages are divided into smaller size packets, and these packets are then forwarded in the network from a source to a destination. Packet switching can be connection-oriented and connection-less. In the connection-oriented mode, a connection request is made before a source can send information. This is called virtual circuit (VC). In the connection-less mode, a source sends information without making a connection. The next sections specifically examines packet data network protocols.

1.4.2.1 Asynchronous Transfer Mode (ATM)

ATM is a connection-oriented packet switching technology. An ATM packet is of fixed size, and it is called a cell. In the past, communications networks were constructed for providing a specific service. Various services, such as data service and
voice service, have their own networks. These networks are inflexible for providing a new type of service. The ATM technology networks make it possible to scale the network bandwidth, and it allows a service provider to provide various types of services with different bandwidth requirements using the same network. It also makes efficient use of the network resources. ATM provides high-speed, low-delay multiplexing and switching networks to support any type of user traffic, such as data, voice, or video. ATM technology is used to provide Broadband Integrated Services Digital Network (B-ISDN). Fig. 1.10 shows the reference model for ATM and B-ISDN.

Each plane of the reference model is described as follows:
• Control plane (C-plane) is responsible for setting up a network connection and managing the connections. It is also responsible for connection release. This plane contains Q.2931 signaling protocol that is used to set up connections. Q.2931 uses the signaling ATM adaptation layer (SAAL) to transport messages between machines. Common part of SAAL detects corrupted traffic transported across any interface using procedures of this plane. The service specific connection-oriented part (SSCOP) of SAAL supports the transfer of variable length traffic across the interface and recovers from errored or lost data units. DMMF can use this plane to setup connections with various QoS in the ATM network. Two types of connections are possible. Switched virtual circuit (SVC) is an on-demand packet data connection. A permanent virtual circuit (PVC) is a permanent connection.

• The User plane (U-plane) is responsible for providing user information transfer, flow control, and recovery operations. It contains application specific protocols, such as TCP/IP or FTP. The U-plane protocols can only be invoked if the C-plane has successfully setup a connection or the connection was preprovisioned. DMMF can use this plane to transport multimedia data in the ATM network.

• The Management plane (M-plane) is divided into two sections: plane management and layer management. Plane management has no layers. It is responsible for coordination of all planes. Layer management manages the entities in the
various vertical layers and performs operation, administration, and maintenance services (OAM). Three major categories of OAM are:

- Fault management services use specific cells to indicate problems, such as loss of a connection, a failed interface, or a failed component. Cells of this category are also used to do the loopback test to ensure the continuity of the network.

- Performance management services use specific cells to monitor and report the performance of connections. Statistics, such as errored cells, lost cells, or severely damaged cells, are reported.

- Activation and deactivation services are used to perform performance monitoring and continuity check of connections.

• The lowest vertical layer is the physical layer. This layer is responsible for sending and receiving bits of data over a physical medium. Physical medium can be space, fiber, or coaxial cable, etc. One of the primary way of carrying ATM cells in the network is through Synchronous Optical Network (SONET) frames.

• The ATM layer is responsible for sending, receiving, and processing ATM cells. Routing and forwarding of the cells is done at this layer in the network. ATM cells have a 5 byte header and a 48 byte payload. An ATM cell at User to Network Interface (UNI) is shown in Fig. 1.11. An ATM cell at Network to
Network Interface (NNI) is shown in Fig. 1.12. NNI is used within the network. The difference between a UNI and a NNI cell header is the Generic Flow Control (GFC) field. Each header contains a Virtual Path Identifier (VPI) and a Virtual Circuit Identifier (VCI). The value of the VCI and the VPI identifies a cell to a unique connection. Each virtual path contains multiple virtual connections. In SVC, VPI, and VCI, values are assigned when a connection is established. In PVC, VPI, and VCI, values are fixed. An ATM network uses these identifiers to relay the traffic through high speed switches from a sending customer premises equipment (CPE) to a receiving CPE.

ATM does not provide any error correction or detection for user payload data; however, error detection/correction is used for header information. An eight bits field header error control (HEC) is used for header error detection and correction in the header. One bit errors are corrected and other are detected. No retransmission service is provided. This allows switching of ATM cells at a gigabit rate. A payload Type Identifier (PTI) field is used to identify the type of data in a cell. Is it a user payload data or OAM data? The C field, which is of 1 bit, is used for marking priority of a cell. A cell with a 0 value in the C field has a higher priority than a cell with a 1 in the C field. Values of VPI and VCI for a particular connection are different between user and network link and between each network switch to network switch link. Switches use VPI and VCI values to determine the route for a particular cell. It also changes the
value of the VPI and VCI in the cell header before forwarding it to the next switch in the path. Fig. 1.13 shows an example ATM switch with VPI's and VCI's.

- **ATM Adaptation Layer (AAL)**

  AAL provides an interface between applications and the ATM layer. There are different types of adaptation layers to support different types of applications, which has different types of data traffic. These traffic types can be of multiple media, such as voice, video, and data with different service guarantees. AAL
isolates the ATM layer from various operations necessary to support diverse types of traffic. There are two main parts to AAL. The convergence sublayer (CS) operations are tailored to application specific needs. CS is further subdivided into common part CS and service specific CS. This is done to avoid duplication of functionalities between various AAL's. Segmentation and re-assembly sublayer (SAR) segments user payload into 48 bytes payload needed for ATM cells. SAR also reassembles ATM cells into the original payload at the receiver.
Multimedia data have different needs. For example, voice and video applications can tolerate some loss of data as long as it is not perceived by the user, but voice and video applications are sensitive to the delay in the network and jitter in the packets if they are real-time and not playback. For data traffic, delay can vary, and it can be transmitted without precise timing relationship, although data traffic cannot tolerate any loss of information such as audio and video data. Based on this observation, AAL defines five classes of traffic. The following operations were considered in determining the traffic classes.

- Whether the data is time-sensitive or not?
- Variable or constant bit rate traffic.
- Connection-oriented or connection-less session between the sender and the receiver.
- Sequenced or out-of-sequence delivery of data.
- Accounting for user traffic.
- SAR of the data units.

The five traffic classes are:

- **Class A**
  
  This class is for connection-oriented constant bit rate (CBR) traffic. This traffic is time-sensitive. An example of Class A traffic is a constant bit rate video or audio.

- **Class B**
  
  This class is for connection-oriented variable bit rate (VBR) traffic. This traffic is time-sensitive. An example of Class B traffic is a variable bit rate video.

- **Class C**
  
  This class is for connection-oriented variable bit rate (VBR) traffic. This traffic is not time-sensitive. An example of Class C traffic is bursty data service.
— Class D

This class is for connection-less VBR traffic. This traffic is not time-sensitive. An example of Class D traffic is datagram service as provided by the Internet Protocol (IP).

— Class X

There is no specific traffic type and timing restriction for this class. It is defined by the user.

Various AALs are designed based on these traffic classes. AAL 1 is designed for class A traffic. AAL 2 is designed for class B traffic. AAL 3/4 is designed for class C and D traffic. People in the data networking felt that AAL 3/4 had too much overhead for data traffic, so they designed AAL 5 for class D traffic.

1.4.2.2 Internet Protocols

In this section the Internet Protocol version 4 (IPv4), which is used currently in the Internet is examined. A newer version, Internet Protocol version 6 (IPv6 or IPng) is also examined. These protocols operate at the network layer of the OSI model. Transport protocols, transmission control protocol (TCP), and user datagram protocol (UDP) are also described. DMMF uses TCP and UDP for network transport, when the underlying network is the Internet. The Internet supports the best-effort delivery service model. It means that the network will do its best to deliver
a packet, but there is no guarantee about anything. IP connects diverse networks using a device called a router. A router runs protocols that build routing table used in the forwarding of packets. This mechanism is dynamic and the routing table adapts to the changing network load. Algorithms based on distance-vector and link-state mechanisms are used for routing table construction.

**IPv4**

![IPv4 packet](image)

Fig. 1.14 shows a IPv4 packet. IP is used to internetwork different networks. Internet protocol runs on top of different physical networks. Packets can be lost, resequenced, corrupted, or misdelivered. The version field is used to identify a version of the IP. HLen field specifies the length of the header in 32-bit words. Without any options, the IPv4 header is 20 bytes long. The TOS field is for defining a service
type for the packet by applications, but it has not been used. The total length field describes total length of the IP packet in units of bytes. This field is 16 bit long, so maximum size for IP datagram is 65535 bytes. The IP runs over different physical networks and they have different maximum size for packets. IPv4 allows for fragmentation of packets in the network by routers; later, at the receiver the fragments are reassembled. An identification field is used to identify fragments belonging to the same packet. Flags and Fragment offset fields are used to reassemble a packet. The time to live (TTL) field is used to limit how long a packet stays in the network. Each time a packet is forwarded, the TTL field is decremented; this way if there is an error, a packet does not keep looping in the network forever. The protocol field is used for demultiplexing. It identifies which higher-level protocol this IP packet belongs. CheckSum is performed on the header fields to determine if the header is corrupted. Packets that fail CheckSum are discarded. The next two fields are the 32 bit source address and destination address. IPv4 has five address classes.

- **Class A:** This address starts with 0 as the first bit. Bits 1 to 7 are network id and bits 8 to 31 are for host id within the network.

- **Class B:** This address starts with 10 as the 0 and 1 bits. Bits 2 to 15 are for network id and bits 16 to 31 are for host id within the network. This is the most widely used class of address.
• Class C: This address starts with 110 as the bits 0, 1 & 2. Bits 3 to 23 are for network id and bits 24 to 31 are for host id within the network.

• Class D: This address starts with 1110 as the bits 0, 1, 2 & 3. The rest of the bits are used for multicast address.

• Class E: This address starts with 11110 as the bits 0, 1, 2, 3 & 4. This class is reserved for future use.

Due to the explosive growth of the Internet, IPv4 addresses may not be able to accommodate all the demand for IP addresses. A new protocol is proposed, which is described in the next section.

IPv6

IPv6 addresses the following issues:
• Scalable routing and addressing. This feature solves the problem of depleting IPv4 addresses and also the size of the routing table in the routers.

• Support for real-time services which is discussed in more detail later.

• Security support. This option provides mechanisms for privacy and authentication. The first mechanism called IPv6 Authentication Header is an extension header that provides authentication and security, but no confidentiality. The second mechanism is IPv6 Encapsulating Security Header. This is also an extension header. It provides integrity and confidentiality to IPv6 datagrams.

• Auto-configuration of the host address. This feature makes it easy for novice users to use the Internet. When a host is connected to the Internet, it will configure automatically with the address and related information to communicate over the Internet.

• Support for QoS. This topic is discussed in detail in section 2.5.3. There is work going on to develop Integrated Services Internet. IPv6 will be used to support multimedia traffic in this new Internet. IPv6 has fields such as flow-label and priority which will be used to support multimedia. IPv6 has multicast capability. A group of interfaces belong to a multicast address. A packet sent to a multicast address is delivered to all the interfaces that are part of the multicast address. A multicast address can have link, node, site-local, organization-local,
or global scope. A scope defines how far from its source a multicast packet will travel in the network.

Fig. 1.15 shows the header of an IPv6 packet. Just like IPv4, the first field is the version number of the IP. The next field is used to specify a packet's priority compared to other packets. FlowLabel is used to describe the stream of data flow from a source to a destination with the same priority. The Payload Length field gives the length of the packet in bytes, excluding the header. If extension headers are used, Next Header field describes the next optional header. If there are no optional headers, then this field is used to specify higher-level protocol. The hop limit field has the same functionality as TTL field of the IPv4. The IP address is now 128 bits in size and it has multiple classes.

TCP

TCP is an end-to-end protocol, meaning it transports data from one end-host to another end-host. TCP is connection-oriented and offers reliable in-order byte-stream service. It is a full-duplex protocol, so each TCP connection supports a pair of byte streams. TCP also has a flow-control mechanism. It allows the receiver to limit how much data the sender can transmit so that the sender does not overrun the receiver buffer capacity. TCP uses sliding-window protocol for flow-control. TCP has a demultiplexing mechanism that allows multiple application programs on a host to use TCP connections. A TCP connection is uniquely identified by source address,
source port, destination address, and destination port. TCP also has a congestion-control mechanism to avoid congestion of data traffic in the network. If the network is overloaded, this mechanism prevents the TCP from sending more traffic, and in turn, regulates the data transfer rate of the TCP connection. TCP has a 3-way handshake mechanism for connection establishment. TCP retransmits packets using go-back-N protocol to provide reliable delivery.

The fields in the TCP packet which are depicted in Fig. 1.16. The source port is a 16-bit number that identifies TCP packets from different connections on the source host. The destination port is a 16-bit number that identifies a connection end-point on the receive host to which a TCP packet belongs. The sequence number field is used to specify sequence number of the first byte of data carried in the packet. The
acknowledgement field is used to define bytes successfully received by the destination. The advertised window field is used to specify how many bytes can be in transit by the sender of the TCP data before it receives an acknowledgement. HdrLen field specifies the header length in 32-bit words. Flags is used for control information. Some of the flags are SYN, FIN, RESET, PUSH, URG, and ACK. The URG flag tells that this data is urgent. The Urgent pointer points to the part in data where the non-urgent data begins. This means that in the data section, urgent data is at the beginning. CheckSum is used to put the checksum value computed over the TCP header, the TCP data, and the pseudo-header made up of the source address, destination address, and length fields from the IP header.

**UDP**

```
<table>
<thead>
<tr>
<th>0</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td></td>
</tr>
<tr>
<td>CheckSum</td>
<td>Length</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 1.17: UDP packet

UDP is an end-to-end protocol that provides demultiplexing on top of IP. Its packet format is shown in Fig. 1.17. UDP has fields for source and destination ports for multiplexing. To ensure that a UDP packet has reached the correct destination,
it performs checksum using a UDP header and length, the source IP address, and the
destination IP address from the IP header. UDP provides connection-less datagram
delivery facility to applications. It does not guarantee delivery of packets nor does
it order them. UDP is used by real-time applications that do not need any protocol
overhead.

1.4.2.3 Multicast Backbone (MBONE)

Multimedia applications, such as distance learning and telemedicine, require sending media data from a source to multiple receivers. There are several ways to achieve this. One way is to send media data to each destination separately. This is called unicast. The second way is to use a broadcast mechanism in the network. In this case, receivers who are not interested in the information will also receive it. The third mechanism is multicasting. In this mechanism, packets sent by a source are selectively forwarded to a group of receivers. A virtual network on top of the Internet which provides multicasting capability is called MBONE [88].

Fig. 1.18 shows MBONE topology. C are hosts connected to a LAN. M are multicast routers which are connected to one another via unicast tunnels. Tunnels are logical unicast links that carry MBONE packets using UDP in the Internet. MBONE packets are encapsulated in the UDP packets. Each tunnel has a metric and a threshold. The metric specifies a route cost and is used for routing calculations.
The threshold is used to limit the scope of a multicast packet. A packet has a time-to-live (TTL) field which is decremented each time it is forwarded. When TTL is less than the tunnel threshold, the packet is dropped; otherwise, it is forwarded. MBONE uses class D addresses of the IP for multicasting. When a host wants to receive packets addressed to a particular multicast address, it issues an Internet Group Management Protocol (IGMP) request. The multicast router for that subnet will inform other routers to forward packets to it. The router places the packets on the LAN in which the requesting host belongs. The local router frequently polls the hosts on the LAN to see if they are still listening to the multicast group. If no one is listening, then the packets belonging to that group are no longer placed on the LAN. In MBONE, the sender doing multicasting does not know who the receivers are. MBONE can be used
in DMMF as one of the network transport technologies to support the multicasting of media.

1.4.2.4 Real-time Transport Protocol (RTP)

RTP provides end-to-end network transport functionality to applications that transmit real-time data such as audio, video, and simulation data over unicast or multicast networks [10]. RTP does not do any resource reservation or provide any guarantee on the performance of the service. RTP is usually used in conjunction with another protocol, real-time transport control protocol (RTCP). RTCP allows monitoring of the data delivery, which can be scalable to large multicast networks, and provides minimal control and identification functionality. RTCP is based on the periodic transmission of control packets to all the participants in the session. RTCP provides feedback on the quality of data distribution. This feedback is used for flow and congestion control. RTP is intended to be malleable to accommodate the information required by a particular application. It will be implemented as part of the application processing in most cases rather than as a separate layer.

A RTP packet contains payload data and a list of the contributing sources. If the data is not mixed, contributing sources list will be empty. RTP and RTCP use the demultiplexing mechanism of the underlying protocols to differentiate packets belonging to sessions at the end-host. In a multimedia session, each media is sent in a separate session. These multiple sessions are distinguished by their different port
numbers. In RTP, the synchronization source sends a stream of RTP packets, which are identified by the 32-bit numeric synchronization source identifier part of the RTP header. Receivers can use this information to synchronize various media from the same source. A mixer in RTP takes RTP packets from one or more sources; it may then change the data format, combine the packets in some manner and then forwards a new RTP packet. A mixer will make timing adjustments among the streams and generate its own timing for the combined stream, because timing among multiple input source will not be synchronized in general. A translator is an intermediate system that forwards RTP packets without changing their synchronization source identifier. A translator can be used to change the encoding of media without mixing. Monitor applications receive RTCP packets that are sent by the participants in a RTP session. It can monitor quality of service, fault diagnosis, and long-term statistics.
CHAPTER 2
Multimedia in Distributed Collaborative Systems

In this chapter, different types of multimedia data are described. Some of the distributed systems that use multimedia data are described. These systems are Remote Consultation and Diagnosis (RCD) used in the telemedicine field and Java enabled InteractiVe Environment (JIVE) used in the distance learning environment. The requirements of the distributed multimedia systems are investigated. Information on quality of service support given by the two networking protocols described in the previous chapter is given.

2.1 Example System Applications

In this section, information on RCD and JIVE applications are given. They represent the broader base of all types of distributed multimedia applications.

2.1.1 Remote Consultation & Diagnosis (RCD)

RCD [90] is a collaborative telemedicine system for remote radiology and pathology consultations. It allows physicians to access patient information from distributed databases. Patient information can be history, textual diagnosis, pathology images,
radiology images, and voice diagnosis. Physician can do diagnosis alone or can consult with other physicians by joining in a multi-party consultation session. Fig. 2.1 shows RCD consultation session between physicians at two sites.

![Diagram of RCD session between physicians]

Figure 2.1: RCD session between physicians

When multiple physicians are in a session, they all see the same patient information. All the images on the screen and the image layouts are the same. Each physician has the ability to annotate images using rectangle, circle or free-hand annotations. When a RCD session is going on, multimedia data consisting of text, image, and annotation are exchanged among all the physicians' workstations in real-time. RCD is designed using Object Oriented Design and Analysis methodology and
is implemented in JAVA language. RCD uses tool kits for GUI, threads, I/O, image processing, and networking, which are part of the JAVA development kit [92].

RCD is a part of the Global Picture Archiving and Communication System (GPACS) [21, 91]. Physicians collaborating in a session can be located anywhere using heterogeneous machines. Communication among different places is achieved using the Internet. To provide these kinds of features, RCD has six major components:

1. A client RCD used by physicians.

2. A communication server to support multi-party group RCD consultation session.

3. A web server to provide access to RCD from a Web browser and to transport patient data.

4. A database server to manage patient data at each site of the global network.

5. A recording component to record a multimedia RCD session.

6. A playback component to playback a recorded RCD session using a Web-browser or in stand-alone mode.

A playback of a previously recorded RCD session is an important functionality of the RCD system. This functionality allows digital recording of all the annotation commands and audio during a consultation session. Later on, these recordings can be retrieved and played in the RCD software or in a Web browser. Various objects
collaborate to make the playback happen. Timing is extremely important especially with real-time multimedia data. The RCD is a dynamic system where various discrete events occur at different times, and the RCD system responds to those events. Some events, such as playback of audio and annotation commands, occur at fixed time intervals; other events such as user interaction can occur at any time.

The RCD lacks the functionality of real-time audio and video exchange between consulting physicians. It also lacks remote capture of audio and video patient data from diagnostic equipments. DMMF will be used to add this functionality to RCD. Using the framework it will be very easy to augment the current RCD.

2.1.2 Java enabled InteractiVe Environment (JIVE)

JIVE is a collaborative distance learning application. Distance education is defined as a formal approach to learning where the majority of the instructions occur while the educator and the learner are at a geographically distant locations. With advances in the computer and communication technology, distance learning can be enriched with multimedia to give the participants a feeling of being present at the same location as the instructor. With the use of new technology, we can take the education instructions to the student using multimedia technology rather than the student to the instruction.

There are several advantages to using distance learning. First, it makes sharing of educational resources possible. A course being taught at one school can be offered at several schools if proper distance learning technology is available. Specially, students
at remote locations can have access to teaching that was not possible before. Students have access to more courses and instructors. It can also make access to instructors easy.

Learning and teaching is a very media intensive endeavor. It involves exchange of multimedia information. In a conventional classroom, sight and sound are exchanged between teacher and students. In the distance learning environment, these media have to be exchanged over a long distance. Typical media used for distance learning are:

- Audio
- Video
- Computer graphics, data, text
- Still video

There are different modes of interaction in the distance learning environment. They are:

- One-way, One-to-Many sites. In this mode, multimedia data is sent from instructor to teacher, but there is no communication back from students at remote locations to the instructor. This method is used in satellite-based distance learning network.
• Two-way, One-to-one site. Here two-way communication is possible between the students and the teacher. This method is limited between two sites and can be achieved using commercial video-conferencing technology.

• Partial two-way, multiple sites. In this mode, multimedia from the instructor site is sent to all the students sites. Return media from a student to the instructor site is restricted to one site. This site can be selected by the instructor based on a question from a student. Therefore, at any given time, there can be two-way communication between 2 sites only. This mode is feasible using video-conferencing tools or broadcast video and telephone lines for audio.

• Two-way, multiple sites. In this mode, multimedia data can be sent and received from all the sites. A data network is used to interconnect all the participants. JIVE falls into this mode of distance learning. An application is developed that creates a virtual classroom. Participants can see and hear one another and use the computer screen to share the teaching material. JIVE will use DMMF for audio and video transfer among students and teacher site.

2.2 Application System Requirements

There are different types of telecommunication networks. Distributed multimedia application has to be able to work using all these different types of telecommunication networks. If a programmer has to deal with the details of all these networks, the
programmer's productivity is hampered. Also dealing with details makes the process more error prone. Middleware framework provides abstractions that are easy to deal with. The following are general requirements of the multimedia application system:

- The multimedia application system requires programming interface to create media objects in a distributed environment easily.

- Of the media objects that are created, some will be source and some will be sink objects. The multimedia application system requires the presentation of media objects. These will be sink objects.

- A system also requires control and interaction of media objects after they are created.

- Since multimedia data is time-critical, the application system requires end-to-end QoS guarantee. This means a guarantee from the network and the two end-systems.

2.3 Isochronous Media

Traditional computer data is not sensitive to time. Data is stored, retrieved, and deleted. These data are not continuous in time. Multimedia data, such as audio and video, are continuous media. They are time-sampled continuous data. Such media is called isochronous media. An isochronous media requires that delay and jitter be tightly bounded from the point of generation or retrieval to the point of presentation.
Timing requirements within the media are called intramedia synchronization. Multiple media streams from different sources also have synchronization requirement. Such requirement is called intermedia synchronization. This synchronization requirement requires proper design of resource managers in the network and the end-systems. Audio and video data are discussed below.

2.3.1 Audio

In computers, audio data are represented in a digital form. Analog audio signals are sampled and then quantized, which results in digital audio. The human ear can hear signals between 20 Hz and 20 kHz of frequency. This requires that sampling should be done at 40 kHz and above to fully reproduce the original signal. In the early days of digital audio, sample frequencies of 44.1 kHz and 48 kHz were adopted to handle the full range of audio frequencies. 32 kHz is also a common sampling frequency. PC multimedia systems use submultiples of 44.1 such as 22.05 kHz and 11.025 kHz sampling frequencies. Quantization makes the strength level of audio signal discrete. The more quantization levels, the more accurate the signal, but it also results in more bits per sample; this, in turn, creates audio signal of higher data rate. To reduce the bandwidth required to transmit or to reduce the space required to save digital audio, various encoding formats have been created. These encodings compress the audio data. Some encodings create a continuous stream of data; that mode is called continuous bit rate (CBR). Encodings that produce a variable rate of
digital data are called VBR. Fig. 2.2 shows several digital audio encodings with bit rate information, which are described in detail in [93].

<table>
<thead>
<tr>
<th>Encoding type</th>
<th>uncompressed kbits/sec</th>
<th>compressed kbits/sec</th>
<th>mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>peak</td>
<td>avg</td>
</tr>
<tr>
<td>CD audio</td>
<td>1400</td>
<td>192</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td></td>
<td>384</td>
<td>192</td>
</tr>
<tr>
<td>ADPCM</td>
<td>64</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>PCM</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>32-21</td>
</tr>
<tr>
<td>FM Stereo</td>
<td>1024</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

Figure 2.2: Bandwidth requirement for digital audio

CD quality audio requires 1400 kbits/sec in uncompressed form and 192 kbits/sec in compressed CBR mode. When bit rate is variable, peak and average data rate are shown. In VBR, CD quality audio requires 384 kbits/sec peak rate and 192 kbits/sec average bit rate. Advanced differential pulse code modulation (ADPCM) requires 32 kbits/sec in CBR mode. Pulse code modulation (PCM) encoding requires 64 kbits/sec in CBR mode. In VBR mode, PCM requires 64 kbits/sec peak and 32 kbits/sec average bit rate. FM stereo quality encoding needs 128 kbits/sec in CBR.
mode. DMMF uses $\mu$-law format, which is type of ADPCM. $\mu$-law effectively handles signals that change quickly and signals that change slowly. Step size encoded between adjacent samples vary based on the signal. If the volume is changing rapidly, large steps can be quantized; otherwise, small steps will be used. The sampling frequency is 8 kHz and each sample is 8-bit, but the effective range is 12-bit due to $\mu$-law.

2.3.2 Video

Video is made of multiple image frames. A distributed multimedia environment requires the transfer of continuous frames of video across the network. Based on the application need, the frame rate for video can be up to 30 frames/sec. The amount of information generated depends on the size of the video frame in terms of pixels. In the raw form, video generates a lot of data; hence it is compressed. In a lot of cases, a specialized hardware card is used to compress images and video. There are several places where redundancy of information occur in digital video. They are:

- In a picture where the same color occurs in multiple pixels adjacent to one another; all these pixels will have the same value.

- When a scene or part of a scene contains mostly vertically oriented objects, there is a possibility that two adjacent lines will be partially or completely the same. So there is redundancy between the lines.
• When there is no or minimal change between subsequent frames of the video, only the difference between the frames needs to be coded. This saves a lot of data.

The first two types of redundancy are called spatial redundancy. It is also called intraframe redundancy. The last type is called temporal redundancy, which is also called interframe redundancy. Here are the different types of compression techniques:

• Simple: Examples are truncation and run-length.

• Interpolative: Example is subsample.

• Predictive: Examples are DPCM and motion compensation.

• Transform: Example is Discrete Cosine Transform (DCT).

• Statistical: Example is Huffman.

Table 2.1 shows the bandwidth requirements for video using different compression techniques, image sizes, frame rates, and sample sizes. This information is given in [93]. 360x288 pixels window with 30 frames/s, 8 bits/sample, Common Intermediate Format (CIF) coding in CBR mode needs 0.768 to 1.5 Mbits/sec. With Quadrature CIF (QCIF) and 10 to 15 frames/s, the compressed rate is 0.384 Mbits/s. With MPEG1 compression format, the interlaced standard TV quality video needs 4 to 9 Mbits/s and cable TV non-interlaced needs 2 to 4 Mbits/s. Motion Joint Photographic Experts Group (MJPEG) is another format for digital video. In this format,
Table 2.1: Bandwidth requirements for digital video

<table>
<thead>
<tr>
<th>Coding/resolution format</th>
<th>frames/s</th>
<th>bits/sample</th>
<th>mode</th>
<th>compressed rate Mbits/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>360x288 pixels CIF</td>
<td>30</td>
<td>8</td>
<td>CBR</td>
<td>0.768-1.5</td>
</tr>
<tr>
<td>360x288/180x144 QCIF</td>
<td>10-15</td>
<td>8</td>
<td>CBR</td>
<td>0.384</td>
</tr>
<tr>
<td>standard TV 720x480</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>interlace, MPEG1</td>
<td>30</td>
<td>8</td>
<td>CBR</td>
<td>4-9</td>
</tr>
<tr>
<td>cable TV, 360x480</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>non-interlace, MPEG1</td>
<td>30</td>
<td>8</td>
<td>CBR</td>
<td>2-4</td>
</tr>
</tbody>
</table>

multiple JPEG frames are sent. There is no temporal compression. Below the JPEG image compression and Motion Picture Experts Group (MPEG) video compression are given in detail.

JPEG

JPEG is an international standards organization that developed the image compression standard. JPEG provides different modes of operation and options within it. Applications select the mode that fits their need. Fig. 2.3 shows block diagrams of different coding modes in JPEG. Modes a, b, and d are lossy compression schemes with DCT encoding of 8x8 pixel block, which is followed by statistical encoding. In lossless mode c, it is implemented with predictive coding, followed by statistical coding. The architectures in the figure are for a gray color image. For a color image, the compression scheme has to be used on each of the component colors. The following are different compression modes which are shown in Fig. 2.3.
Figure 2.3: Block diagram for various types of JPEG coding
- Sequential-mode encoding is shown in part a. DCT encoding is done on the blocks of the image as it is scanned. DCT coefficients are transferred as they are calculated from the scanned image.

- Part b is progressive mode encoding. In this mode, an image buffer is added after DCT encoding. Different portions of the DCT coefficients are read from the buffer to achieve progressively improved quality over several scans.

- For part d hierarchical mode encoding, processing is added before the DCT encoder to filter and subsample the source image before encoding. Subsampling and encoding is done repeatedly with decreasing subsampling to get images of increasing resolution.

- Lossless encoding mode part c does not use DCT. It uses a simple predictor. For color images, this scheme delivers approximately 2:1 compression.

Fig. 2.4 shows the output of a DCT encoder. The upper left corner has DC coefficients; the AC coefficients are arranged with increasing spatial frequency horizontally and vertically. Quantization of DCT coefficients is done according to the quantizing table provided by the application to the encoder; this table becomes part of the image data. There is a step size for each coefficient. This allows coefficients to be represented with desired image quality. Zigzag ordering, as shown in Fig. 2.5, arranges the DCT coefficients so that the statistical encoding is more effective. Use
Figure 2.4: 2-D matrix of DCT coefficients

Figure 2.5: JPEG zigzag ordering
of the zigzag sequence creates a bit stream where nonzero coefficients are placed before coefficients that are more likely to be zero. Zigzag places zero-value coefficients together at the end so they are coded with very few bits. Statistical encoding does lossless compression of quantized DCT coefficients. JPEG uses Huffman coding or an arithmetic coding type of statistical encoding. For Huffman coding, the Huffman table has to be included in the encoded image.

MPEG

MPEG-1 standard for digital video is designed to deliver acceptable quality video at compressed data rates of 1 to 1.5 Mbits/s. After compression, random-access playback is possible. The MPEG standard is bit stream specification. The MPEG standard has three versions, MPEG-1, MPEG-2, and MPEG-4. The architecture is based on a sequence of images. The order of transmission of frames is not necessarily the same order that is used for display. There are four types of frames.
• I frame: This frame is intracoded; so, it is independent of any other frame. An I frame must exist at the start of any video stream. The I frame is a random-access entry point into the video stream. The I frame is similar to a JPEG image.

• P frame: This is a predicted frame, which is coded using motion compensation from a previous I or P frame. The P frame requires about 1/3 of the data of an I frame.

• B frame: This frame is created by interpolating a previous and a future I or P frame. The B frame requires anywhere from 1/5 to 1/2 the data of a P frame.

• D frame: This is a special format frame used only for implementing the fast search modes.

More compression can be obtained by using more B frames. A I or P frame has to be transmitted first before transmitting a B frame for decoding. Fig. 2.6 shows a typical MPEG frame sequence. There are follow-on standards to MPEG. The video component of MPEG-2 uses bandwidth of 2 to 15 Mbps. It is targeted for TV broadcasts. MPEG-4 provides very low bit rate encoding suitable for wireless applications.
2.4 Non-isochronous media

Non-isochronous media is not continuous in time. Media that fall into this category are image, text, annotation, and data.

2.4.1 Image

Image can be encoded in various formats, such as GIF, JPEG, TIFF, etc. An image can be retrieved, stored, displayed, or captured from various media devices in DMMF. These operations on the image are done as necessary, but are not continuous in time. Continuous transfer of images becomes video, for example MJPEG. In the RCD application, images can be X-ray or pathology slides. In the JIVE application, images can be used for presentation slides. Data size of an image depends on the quality of image, compression scheme, and pixel size.

2.4.2 Text

Text media data are the most common forms of data used in computer systems. In a CORBA based system, text will be transferred as a parameter of an operation or as the return value of an operation. Text can be used to convey any type of information. In the RCD system, text is used for image diagnosis and patient information. In the JIVE system, text will be used to give instructions to students and to save student profiles. Text is transferred based on events at discrete times, which require access to text data. Data size of text data is very small compared to image and video.
2.4.3 Annotation

Annotation data is used in collaborative applications. Annotation is used to point and identify information in an image in a shared environment. It can also be used in white boarding applications to draw pictures. An annotation can be a rectangle, a circle, or free-hand.

2.5 Quality of Service (QoS)

In a distributed multimedia system, the QoS received by the media is very important because of its time-sensitive nature. Distributed multimedia applications require a QoS guarantee from the end-systems and from data network. Framework, such as DMMF, that supports system applications have to provide this functionality in them. Two types of QoS are presented below:

2.5.1 Qualitative

For non-isochronous media, such as text, annotation, image, and data, reliability is very important. These media cannot tolerate loss of data. They are less sensitive to delay and jitter. They may have a bandwidth requirement if the size of the image is large.

For isochronous media or continuous media, such as audio and video, delay, jitter, and sequencing of packets are important. These media can tolerate loss of packets to some degree, but they are time sensitive. If a packet comes too late, it will be
useless. If a packet is out-of-sequence, then the application has to provide protocol for buffering and reordering the packets. This will introduce delay in playing the packet. If reordering functionality is not provided, then the packet is useless.

2.5.2 Quantitative

To receive a guarantee on performance of data transport and processing, an application has to specify its service requirements and traffic parameters qualitatively to the network and end-system performance manager. The parameters are:

- Average packet delay
- Average bandwidth
- Peak bandwidth
- Packet loss probability
- Jitter (variation in delay)

2.5.3 QoS in IPv6

In this section, how QoS can be provided in the Internet is summarized. The current Internet model is based on the best-effort delivery of datagram packets. In this model, all the packets from a variety of applications receive the same class of service. If the network is loaded with a lot of traffic, packets are delayed significantly. Some of the packets are lost due to the buffer being full at the routers in the path
of the packets. For some applications, such as email, the current Internet model is sufficient. Applications that will be supported by DMMF require real-time or non-default QoS support from the network. These applications need more than the best-effort delivery of the packets; they need some kind of guarantee about the QoS their packets will receive from the network. Various applications need various types of service; hence, it is necessary that the network offers multiple QoS classes. The current version of the Internet protocol IPv4 has an 8 bit type of service field in the header, but it is not used to support QoS that is required by some of the applications described above. A new version of the Internet protocol has been standardized, which is called IPv6.

A stream of packets from an application that requires special handling in terms of the QoS that it receives from the network is called a flow. If the network is to give a guarantee on the QoS that is requested by the application, then the network needs to know the characteristics of the application’s flow and the type of quality requested for the flow. This means that there has to be a way to describe a flow and a type of service to the network. Some of the parameters that can be used to describe a QoS are bandwidth, delay, and jitter. Number and type of parameters can be different depending on the QoS class. There also has to be some type of admission control mechanism by the network to either grant or reject an application’s request, because the network resources are not unlimited.
2.5.3.1 IPv6 features related to QoS support

In this section, the IPv6 features that are relevant in supporting QoS are described.

- Larger address space: Due to the explosive growth of the Internet, there are not enough IPv4 32 bit addresses left for new networks. As shown in Fig. 1.15, IPv6 has 128 bit address space so that many more addresses can be supported. Addresses can be assigned based on QoS. When the routers see the packets with the destination address as one of the special QoS addresses, they will process them with the appropriate priority. This requires that the applications run on a host connected to a particular QoS address be known. This mechanism forces address per service. If a host runs applications that use different QoSs, the host needs to be configured with multiple addresses. IPv6 supports multiple host addresses. The QoS per address scheme may be used for corporate Intranets where the network environment is more controllable. For the Internet, this scheme wastes a lot of addresses.

- Simplified header format: Header format in IPv6 is simplified so that less fields need to be examined by a router in the network to perform its routing function. This is done by having few fields in the required header and putting the rest of the additional information in the optional headers. This saves processing time and enables the router to process more packets because most of the time the router has to examine only the required header.
• Packet fragmentation: Packet fragmentation is not permitted in the IPv6 routers. A source has to find the maximum packet size on the path from the source to the destination. Routers are not permitted to fragment the packets. If there is a need for fragmentation, the source has to fragment the packets. This increases overhead at the source, but it may improve the efficiency of the routers in the network.

Another reason for this is that the routers must guarantee the QoS to a particular flow of packets from an application. A router may have guaranteed the QoS to the flow assuming that there will be no fragmenting. If the router is loaded with work, and it has to do fragmenting of packets for which it did not account any resources, it may not be able to satisfy the guarantee it gave about the QoS. On the flip side, when deciding about whether to give a guarantee about a particular QoS, if a router allocates extra resources to account for the fragmentation, which it may not have to do the most of the time, then it wastes resources.

Maximum transfer unit (MTU), which is the maximum packet size that can be transmitted by the source for a flow, has to be determined prior to the flow setup. Even though there is an overhead associated in finding the MTU, it needs to be done only once for a flow. It allows a source to send packets of the size of MTU rather than some common denominator (size which can be assumed will be supported by all the networks in the Internet) default size which is a lot smaller than the MTU. If a packet sent by a source is fragmented in the network and one of the fragments has an error or is lost, the source has to transmit the whole packet again. If the
source sends packets of the size of the fragment, then only a packet of the size of the
fragment needs to be retransmitted. With a smaller size, more fragmenting has to
be done at the source and de-fragmenting at the destination, but it is more efficient
than fragmenting at the routers in the network. Using MTU and no fragmentation
in the network, QoS can be supported more efficiently.

- Authentication and privacy: IPv6 has an extension header to provide authentication
  and privacy services. A use of this feature for QoS support can be to authenticate
  and identify the source that is making a request for QoS. Once the request is granted,
  the fields can be used to authenticate the packets belonging to a particular flow. This
  way, a source cannot use the resources allocated to someone else.

- Hop-by-hop options header: It carries optional header information that every router
  along the path of the packet must examine. This header can be used to deliver QoS
  related information to the routers on a packet’s path.

- Routing header: This header can be used to define a specific route that the packet
  should take on its way to the destination. Selection of a route can be based on the
  QoS provided by the route. This option requires that the source has knowledge of
  which route to use.

- Priority field: This field is provided in the required header of IPv6. It contains 4
  bits. The priority field can be used to categorize packets into different priorities which
  the routers in the network can use for forwarding the packets. Priority classifies how
important different packets are from the same source. There are two major classes of packets:

1. Packets for which the source provides congestion control. Let’s call this class category-I. An example of this type of packet is the packet that belongs to a TCP connection. In this class, when a source detects congestion in the network by means of a long delay in receiving acknowledgments or loss of packets, it reduces the amount of packets it sends. It gradually sends more packets to get more bandwidth. It is possible that the packets of this category may be lost due to the buffer overflow at the routers. They can be delayed by variable amounts, and they may also arrive out of sequence. Eight different levels of priorities can be assigned to the packets in this class. A value of 0 to 7 is used in the priority field to categorize the packets, with 0 meaning the lowest priority and 7 means the highest priority. Now, different categories in this class are briefly summarized with increasing priority:

(a) Uncharacterized traffic (Value 0): If no indication is given on the priority of the data, it is put into this class.

(b) Filler traffic (Value 1): This priority is used for packets which can be processed in the background; it is particularly suitable for applications such as network news.
(c) Unattended data transfer (Value 2) : This priority is used for applications such as e-mail. A user invokes an action but doesn’t expect it to be completed immediately. The task is completed some time in the near future.

(d) Value 3 : This value is reserved.

(e) Attended bulk transfer (Value 4) : This priority is used for applications that have a large amount of data to transfer. The user waits for the transfer to finish, but is willing to accept a larger delay. An example of the protocols that may use this type of priority is File Transfer Protocol (FTP) and Hyper-text Transfer Protocol (HTTP).

(f) Value 5 : This value is reserved.

(g) Interactive traffic (Value 6) : This priority can be used for applications which transfer a small amount of data, but require very low delay. The user wants to see a quick response to the actions. This priority is different from the attended bulk transfer because here the user is not willing to accept a longer delay. An example of an application that may use this priority is telnet.

(h) Internet control traffic (Value 7) : Control traffic has the highest priority because in the event of congestion in the network, the network has to be managed properly. Simple Network Management Protocol (SNMP) is
used to send information about the congestion in the network, so that the parameters and configuration can be changed to relieve the congestion. When the congestion is relieved, lower priority traffic is able to reach its destination. Routing protocol packets are also assigned this category because they are used to update routing tables in the routers.

Applications can assign these priorities to their traffic. There is no mechanism in the network to verify that the priority used is correct for a particular type of application. To put a mechanism in the network for verification is not a good idea, because it increases the processing overhead in the routers. Everyone using the Internet will have to follow these rules voluntarily.

2. Packets for which congestion control is not provided by the source: This type of packet is used for real-time data. Let’s call this class category-II. A value of 8 to 15 is assigned to the packets in this class. A value of 8 has the lowest priority and 15 has the highest priority. These values will be used for the traffic for which relatively constant data rate and some bound on the packet delay is required. Assigning priority values does not really give any solid number on the data rate and packet delay expected by the application. A separate control mechanism is needed to convey those values to the network. For example, a value of 15 can be used for packets that contain audio data so that in the event of congestion, these packets are not dropped. This is needed because the effect
of the loss or delay of an audio packet may be highly perceptible to the end-user. A value of 8 can be used for packets that can be discarded in the event of congestion. Packets with a priority field value of 8, if delayed, may not be useful to the application, so it is permissible for the network to drop them.

In the layered encoding of a digital video or an image, different data streams are generated for different resolutions. There is a stream for the low resolution coarse grain data, and more streams are added to create a fine grain image or video. The highest priority can be assigned to the coarse grain stream, and then decrease priorities going from the coarse to the fine grain streams. By using this scheme, in the event of congestion, packets that provide finer details can be dropped and a user is able to see a coarser image or a video. It is important to note that there is no relationship specified between the priorities of congestion controlled and non-congestion controlled packets. It seems that a router has to divide its resources between these two different categories, or somehow merge these two categories and re-prioritize them for its internal processing. A router gets information from the Priority field that enables it to process packets more intelligently, but the service model supported is still the best-effort model.

- Flow-label: Flow-label is a 24 bit field. A source can assign a number (flow-label) to a sequence of packets addressed to a particular destination for which the source wants special handling. The purpose of the flow-label is to identify the packets to the network that belongs to a particular flow. A flow-label can be used by a network
to find the service requirements for a flow so it can take appropriate action. How flow-state information can be setup in the network is analyzed in a later section. A flow is uniquely identified in the network by its source address and flow-label fields.

An application can generate multiple flows based on its requirements for different types of data. For example, in a Remote Consultation and Diagnosis (RCD) application [21], flows can be assigned to audio and annotation data. Audio and annotation flows have different bandwidth, delay and jitter characteristics. A same flow number also can be used by multiple applications from the same host if they all require the same QoS.

By using the flow-label, the handling of packets by a router can be simplified. A router can look at the flow-label and retrieve the appropriate routing and scheduling information about the packet from a state-table if a state has already been established for that flow. A flow-label has to adhere to the following rules:

1. Routers and hosts that do not support the flow-label field should do the following:

   - Set flow-label field to 0 when generating packets.
   - Ignore the flow-label in received packets.
   - Keep the flow-label unchanged when forwarding the packets with a flow-label.
2. A number for a flow-label should be randomly and uniformly distributed between 1 and $2^{24} - 1$. A router uses a flow-label to associate the state information about a flow. During a packet forwarding operation, if a router uses a linear search to find the information about the packet's flow, then it will take too much time. If a field is kept for each flow number in the whole range, the memory requirement in the router will be unsatisfactory. Since the flow-labels are randomly and uniformly distributed, they can be used to create a key to a hash table to enable routers to get the information about a flow quickly.

3. For a unique source address and a non-zero flow-label combination, destination address, priority, hop-by-hop option header (if present), and routing option header (if present) should contain the same information. This makes sense because all the packets in the flow need to go between the same end points with the same QoS. A router can process these packets by looking at the source address and the flow-label.

In this section, the IPv6 features that can be used to provide QoS support were examined. These features let us define the priority for unguaranteed service traffic and flow-label for guaranteed service traffic. There is no specific mechanism provided to establish a state for a flow in the network. The service model of the network has to change from the best-effort to more than the best-effort with a variety of services to support different QoSs. A proposed new model [19] is called the Integrated Services
(IS) model of the Internet. In the next section, various issues that have to be resolved to provide QoS in the Internet are summarized and analyzed.

2.5.3.2 Issues to be resolved to offer QoS

IPv6 and the Integrated Services framework of the Internet extends the current best-effort QoS model of the Internet. The new model combines the best-effort service and other new services that are needed by the real-time applications. A new separate network can be created to satisfy the need of the real-time applications, but this means setting up a totally new infrastructure. With separate networks, we will loose the advantage gained by statistical multiplexing data from different types of services, so the Integrated Services model is more efficient. The requirements necessary to offer different QoSs are discussed below:

- Accounting and verification

Since various types of service will be offered, it is necessary that the users are charged based on the type of service they use. A user who uses the guaranteed service should be charged more than the user who uses the best-effort service. With the use of the priority field in IPv6, best-effort traffic can also receive different network performance. A user of the high-priority traffic should be charged more than the user of the low-priority traffic. A user should also be able to verify that he or she is receiving the type of service he/she asked for and is paying for. The network should
be able to police a user’s traffic to make sure that the user is conforming to the traffic specification it gave.

- Description of the traffic

A set of standard classes can be created to describe a source’s packet data traffic characteristics. The network will be required to give a guarantee that it will support a particular QoS for the traffic, so it needs to know the type of traffic for which it is giving a guarantee. A packet traffic can be described in terms of various parameters or one of a standard predefined class. Most of the applications will generate different types of traffic. One application may generate different types of traffic based on the user input at the invocation or based on the host hardware capability. If we chose to use a set of standard predefined classes, we will have to define a lot of classes to accommodate a wide variety of traffic that can be generated by the applications. The routers in the network will have to remember all these classes to give the QoS guarantee and to process them. We also do not know what kind of applications will emerge in the future, so we may have to add more classes and upgrade the routers every time there is a new class of service.

If parameters are used, then the routers can have generic algorithms that examine these parameters and make decision about admission control, scheduling, etc. All the variety of present and future applications can specify different values for the parameters. Consider a case where an application’s requirement is not met by the predefined traffic classes. Since the application cannot specify the exact values, it has
to select the next higher traffic class that meets its requirements. The routers have
to allocate more resources than needed, so using parameters to describe the traffic is
more efficient. On the other side, predefined QoS classes also have some advantages.
With predefined classes, routers can be designed to optimize handling of packets that
belong to these classes. This can result in faster and better QoS support.

In the Integrated Services framework, traffic specification is called TSpec [96]. A
host will ask for a specific QoS for the traffic described by a TSpec. If the network
approves, the host’s traffic will receive the QoS as long as its traffic conforms to the
TSpec. If the host’s traffic does not conform to the TSpec, the routers will have to
take some action. A router can decide how it wants to handle non-conforming traffic.
If it has resources, it can give the non-conforming traffic the guaranteed QoS. It can
also discard the packets or give them best-effort service. Elements of TSpec are:

1. Token Bucket rate $r$: This is the rate at which credit is given to the source to
   transmit its data. Bucket will fill up at this rate with credits. To transmit, a
   source uses the credits equivalent to the size of the data it sends. If no credits
   are available, then the source waits until enough credit is available for it to
   transmit.

2. Bucket size $b$: This is the size of the credit bucket. When enough credits are
   available, a source can transmit. A credit bucket gets filled up at rate $r$. If the
   bucket is full, credits from rate $r$ are lost. Bucket size limits the burstiness in
the data transmitted because a source cannot continuously transmit data more than the bucket size.

3. Peak rate $p$: This is the maximum rate at which a source can transmit. It is greater than $r$. When there is credit in the bucket, a source can transmit at a rate greater than $r$ but not more than $p$.

4. MTU: Maximum transmission unit.

5. Minimum policed unit $m$: In policing and conformance testing to TSpec, packets of less than $m$ bytes are counted as of size $m$.

A QoS definition may not need to use all the parameters in the TSpec; fields for the parameters that are not used can be left empty.

- Receiver capability

A source knows the type of traffic it is generating, but does it know the capability of the receiver? A receiver may have a low bandwidth connection or it may not have appropriate hardware to deal with high bandwidth data. Even if a receiver may be equipped, what if a user wants a lower bandwidth service. The price for the use of the network service should reflect the amount of usage and the QoS used. It is important that a source does not reserve more resources when a receiver is not going to use it. Some type of mediation is necessary between the source and the receiver before or during a flow setup in the network.

- Sender or Receiver specified resource reservation
Figure 2.7: Source initiated resource reservation in a multicast tree
A lot of multimedia applications that need QoS are based on multicasting. Usually, a large multicasting group is very dynamic because members join and leave the group at any time. If the source has to change reservations every time a member joins or leaves, the source will be overwhelmed with it. Also as described previously, the members may have different requirements. It is necessary that the source and the receiver mediates the desired QoS and traffic characteristics. A scheme, where a source specifies its traffic characteristics first, then using the information from the source, a receiver specifies the traffic characteristics it is interested in receiving with the desired network, QoS seems efficient. Routers will make the appropriate changes to handle the receiver requests. Fig. 2.7 shows a source initiated resource reservation. Even though destination D1 and D2 are interested in 1.5 Mbps and 1 Mbps bandwidth respectively, all the links carry 2 Mbps traffic because the source reserved it and it has no idea what the receivers want. If the receivers reserve resources, then as shown in Fig. 2.8, only 1.5 Mbps bandwidth is required between nodes 1-3,3-4,4-D1 and 1 Mbps is required between nodes 3-5,5-D2. So receiver-oriented resource reservation scheme handles receiver heterogeneity better in a multicast group.

• Flow-state in the routers

There is a need for a control protocol to transport the reservation information, so that a flow-state can be setup in the routers in the network. There are two possible options. The first possibility is to create a separate reservation protocol for explicit flow-state setup, and the second is to make the reservation information part of the
Figure 2.8: Receiver initiated resource reservation in a multicast tree
IPv6 optional headers. There can be any number of control protocols, but everyone has to agree to implement them. Consider the following two options:

1. Resource Reservation Protocol (RSVP) [16] is a resource reservation control protocol that is not part of IPv6. Here is a brief description of the RSVP:

   (a) It is receiver oriented. The receiver specifies traffic parameters and the QoS it wants for the specified traffic.

   (b) Each receiver may reserve different amount of resources; it may receive different data streams sent to the same multicast group.

   (c) RSVP aggregates reservations from multiple receivers who are in the same multicast group. If multiple receivers are interested in the same data and they are all in the same part of the sub-tree, then resources equal to the highest request among all of them are reserved. In a source oriented reservation, it is difficult to know how much resources should be reserved at each tree node due to the receiver heterogeneity.

   (d) RSVP allows dynamic membership change.

   (e) A receiver should request bandwidth for the maximum number of simultaneous streams it wants to receive. A receiver gets the source TSpec information before it establishes a state in the router.

2. Another way to set up resource reservation is to use the hop-by-hop optional header of IPv6. Every router on the way from the source to the destination is
required to examine and process this header. All the reservation information, such as the traffic specification and the QoS required, can be put in the fields of the header. A router can associate the information found in this header to the flow-label and remembers it. A receiver can specify the flow-label and its specifications (traffic and QoS) in the hop-by-hop header and send it back to the source. Therefore, the routers along the path from the receiver to the source are able to make a reservation.

Besides establishing the flow-state, there are other issues associated with the flow-state in the routers as described in [95]. These issues are discussed below.

1. Stale flow-state and reuse of a flow-label number

Flow-state in the routers is kept using a flow-label; hence a source should not reuse a flow-label until all the previous state information for that flow-label is flushed from all the routers in the Internet. There may be stale flow-labels in the routers even though the source ended the flow because the Internet may have been partitioned and the packets take different routes, a deletion message is lost or a source crashed before sending a message to end the flow. To remedy this situation, a timer can be kept for every flow-state in the router. If the timer expires, a router can delete the flow-state. A source/receiver has to send a refresh message to reset the timer or when a router processes packets from a flow, it refreshes the timer. The process of sending refresh messages creates an
overhead for the source/receiver. If a router resets the timer every time it sees a packet from a flow, it will be tremendous overhead for the router. Previous argument is based on the assumption that the rate of data packets will be a lot higher than the rate of refresh messages. A source should at least wait up to the time equivalent to the timer in the routers to reuse a flow-label number. As described before, a source should number flow-labels uniformly distributed; that means, it will be a while before it reuses the same label anyway.

2. What should a router do with the flow-label for which it has no state?

If a router crashed, and the flow-state information was not saved in a persistent storage, the router will not have any flow-state information when it comes back up. A router can ignore the flow-label and process the packet in the best-effort class by looking at the priority field. If the router uses a wrong outgoing link for the packet due to the loss of the state, it will not meet the QoS guarantee. If every packet carries the flow traffic specification and QoS information in an optional header, a router can establish a new flow-state for that flow-label. This means that the size of the packet will increase and it will waste bandwidth due to transmission of the optional header in every packet. The one-time setup method will make the service less robust because there will be a break in the QoS until the flow-state can be re-established in the router by explicit flow-setup mechanism.
Consider a scenario where a flow-state information is part of every packet and the state for a flow is lost. A router receives this packet and finds that it does not have a state for this flow. The router attempts to establish the flow-state, but what if it does not have enough resources? One option is that a router should set aside some resources for lost flow-states. Alternatively, a router can use some of the best-effort resources, and then adjusts its admission control and scheduling policy to accommodate the flows whose status is lost. In non-explicit flow-state setup scenario, a router cannot distinguish between a packet from the flow whose state is lost and a packet from a new flow. In this case, a router can ask its neighbor from which it received the packet whether it knows about this flow. If the packet is from an old flow, then the router should try to establish a state, otherwise it can apply its admission control policy.

In the explicit flow-state mechanism, a router can send Internet Control Message Protocol (ICMP) to the source indicating that it does not have a state for a flow-label. This will tell the source to send the appropriate information to the router. If resource reservation is receiver initiated, then the router should send a message to the receiver. When the receiver receives the control message, it will establish the flow-state again.

In a receiver-oriented state establishment scheme, when a flow information is included in every packet from the source, network resources are not used efficiently because in this situation the source instead of the receiver defines the
parameters needed to establish a state. It is obvious that a receiver needs equal or less resources for a flow than a source because the source specifies the maximum resources needed for its data. Initially, in the recovery phase, resources defined by a source can be used for the flow-state, when a message from the receiver is received, appropriate changes can be made.

3. Which datagrams should have a flow-label?

The intention of putting the flow-label field in the IPv6 header is to support non-default QoS, so that data requiring QoS guarantee can use a flow-label to identify itself to the network. There is an overhead associated with maintaining and establishing a state for a flow-label in the network; therefore flows that are very short in duration should avoid using a flow-label and use only the priority field. A flow-label can be used for every TCP connection. Routers can use the flow-label to provide a better service for a TCP connection by learning the traffic pattern and doing scheduling accordingly. A flow-label can also be used for efficient demultiplexing at the receiver, but this means that a flow-label cannot be shared between different applications requiring the same QoS. Views that oppose using a flow-label for every TCP connection are: Currently in IPv4, packets are routed using the destination address, so the state information has one entry per destination. Fig. 2.9 shows source S1 and S2 which send data to destination D1. R1-R5 are routers in the network. There are four TCP
connections TCP1-TCP4. Partial state information using IPv4 is shown in the tables with a solid-line border which have one entry for each destination. For example, at router R4, a packet with destination address D1 is forwarded to router R2 no matter to which TCP connection it belongs. Packets from multiple TCP connections with the same destination are routed using this one state. Now, if every TCP connection uses a flow-label, then a separate entry for each flow is kept in a router. This is shown in the tables with a dashed-line border in Fig. 2.9. At router R1, packets from TCP4 without flow-label are forwarded to R2, but with a flow-label it is forwarded to R4. This may be to satisfy QoS requirements. State tables can contain other information, such as scheduling priority, which is shown as Q. Flow-label per TCP connection will increase the memory requirement for the routers.

- Admission control

A router uses an admission control policy to decide whether to accept or reject a service request for a flow. Admission control should be performed when a flow-state is established. Admission control is tied to the flow setup because the end-system establishing a flow must receive feedback, whether the network is able to support its request or not. In the Internet, there are multiple routers along the way from a source to a destination. To provide QoS to an end-system, all the routers along the path have to approve the requested QoS for a given traffic specification. RSVP or the information in the hop-by-hop optional header of IPv6 can be used to provide
Figure 2.9: State information at a router
information to the routers. Routers can implement any admission control policy they wish to decide if they support a request or not. This provides flexibility to the routers in implementing their own internal processing of the packets. If the end-systems want to know the capability of the network between themselves, special fields for hop-by-hop optional header can be developed. The hop-by-hop optional header is a feature of IPv6, mentioned in section 2.5.3.1. A packet will be sent between the end-systems with the optional header. There can be a set of standard parameters that each router along the path will supply as they process the packet with this special optional header. When the packet reaches the end-system, it can see the capability of the network. RSVP diagnostics also can be used to find the capability of the network between end-systems. By using this information, an end-system can request a QoS that it knows can be admitted by the network.

• Packet Scheduling

Routers in the new Integrated Services Internet with IPv6 will have to process different types of packets. We discussed different classes defined by the priority field in the IPv6 header. With a flow and its associated QoS, routers have to make sure that they meet the QoS granted to a flow. A packet classifier identifies the packets into different categories as they come in. A packet scheduler schedules them for forwarding so that the requirements of all the packets can be satisfied. A packet classifier and scheduler has to be tightly integrated because the scheduler has to know how to schedule the classified packets so that QoS can be met efficiently. They
can be implemented separately with a well-defined interface to provide modularity. Routers can implement any scheduling policy they think is efficient. VirtualClock [15] algorithm is one of the possible scheduling algorithms.

One policy that comes to mind is that initially all the resources can be assigned to congestion-controlled (category-I) and non congestion-controlled (category-II) traffic. As requests for QoS from traffic with a flow-label are received, resources can be assigned to guaranteed QoS traffic and removed from category-I & II traffic. There should be some minimum resources reserved for category-I & II traffic; so, that when a router receives QoS requests beyond maximum resource limit, the requests can be denied. This way a router can make sure that at least a reasonable level of service is provided to category-I & II traffic. When there is less guaranteed traffic, more resources become available to category-I & II traffic, so the resources are not wasted.

- Routing

A new way of routing packets is needed which couples with resource reservation, admission control and QoS to find the best route for a given requirement. One possibility is to add metrics, such as bandwidth and propagation delay for an outgoing link to the routing-table. When a flow request with a QoS guarantee comes in, a router may wish to investigate various paths to decide which one is the most efficient. In the routing table, there can be one entry for the non-guaranteed traffic based on the destination address and separate entries for each flow. This increases the size of the routing table and makes the routing algorithm more complex.
In RSVP, a receiver decides the type of service it wants. In the current multicast routing protocols, the route that the receiver's packets travel is determined by the multicast-tree based on the source. So after looking at the receiver's request of QoS, it is not possible to change the route. A new routing protocol is needed to meet the receiver-oriented QoS request more efficiently.

Another option is that if a source knows that a particular network provider can meet the QoS it needs, then the source-routing option of IPv6 can be used. A source can use a flow-label and the source-routing optional header for a flow. This way all the packets from a flow travel the same route and receive the particular service provided by the network provider.

- End-to-end QoS

Usually, a host is connected to a Local Area Network (LAN). Ethernet, which is the most popular LAN technology, uses a random access protocol to transmit packets on the LAN. A router is used to communicate from a LAN to the rest of the Internet. With IPv6 and new algorithms in the routers, we will be able to support the QoS in the Internet, but the QoS cannot be guaranteed in a random access LAN. To provide the QoS up to the end-host, a new protocol for the LAN is needed that supports QoS. A step in this direction is IEEE 802.9 protocol for isoEthernet [22, 23]. By combining isoEthernet with the Integrated Services Internet end-to-end QoS can be provided.
2.5.4 QoS in ATM

ATM technology was designed to provide QoS guarantee in mind. ATM is a connection-oriented technology, which provides an opportunity to specify and negotiate QoS with the network before a connection is established. Some of the issues described in the previous section remains the same. These issues are admission control, packet scheduling, routing, and description of the traffic. ATM uses sender specified resource reservation. The size of the packets (cells) in ATM is fixed 53 bytes. One reason for the small fixed packet size is to simplify the design of switches. With the small cell size, a cell that carries real-time data in its payload does not have to wait too long before it gets switched. If there are multiple size packets, and the packet with real-time data is behind a large packet in the switch queue, it is not switched until the large packet is switched.

ATM technology is more advanced than IP technology in terms of providing QoS. The functions used to achieve traffic control and congestion control so that QoS can be guaranteed are:

- Call admission and control procedure. ATM standard for call signaling is Q.2931.

- Monitor and regulate traffic at UNI. This is called usage parameter control.
• If a user traffic does not confirm to specifications, change the priority of non-confirming cells using the cell loss priority bit. Non-confirming cells can also be dropped.

• Use of traffic shaping mechanism at the UNI so that traffic is managed. This is needed for bursty traffic.
CHAPTER 3

Review of CORBA Middleware Protocols

Networks interconnected in an internet are rarely of the same type. An overall network is made of interconnected heterogeneous networks. User end-systems are also diverse. They are made of different hardware and operating system platforms. On top of this, applications are written in different programming languages. Various organizations in a large company have diverse information technology needs. A solution suitable for one organization may not be suitable for the other; this creates heterogeneity. Still, there is a need for these applications to communicate with one another. A traditional way to create a system applications was to use functional programming. Object-oriented analysis and design of software systems provides a better way to build complex systems. Companies have investments in their current systems which they want to leverage while moving to new systems and new technology. CORBA provides distributed object-based architecture that allows integration of all systems. In this chapter, CORBA 2.0 [100, 9] is described briefly.

3.1 Object Management Architecture (OMA)

OMA is made of two models, Object model and Reference model. The Object model defines how objects are described in a distributed heterogeneous environment. The
Reference model describes interaction between objects. Distributed object systems developed using these models will be interoperable. An OMA object has a unique identity and is located in a distributed environment. An object has well-defined interfaces that define the services offered by an object. A client issues requests to objects to perform some service. Details of how an object performs a particular service is hidden from the client.

Fig. 3.1 shows the OMA reference model with its components. The object request broker (ORB) acts as a global bus and provides communication between objects. There are four main components based on different types of interfaces.

3.1.1 Object Services

These interfaces provide services that are used by the most of the distributed object programs. These services are basic in nature, so they are needed by the
distributed object programs in order to operate in the CORBA environment. Object services augment and complement the functionality of the ORB. There are several object services defined. They are:

1. Life Cycle Service: It defines operations for creating, copying, moving, and destroying objects.

2. Naming Service: Allows clients to find objects based on names. It also allows objects to bind to existing network directories such as X.500, OSF DCE naming service, and SUN NIS.

3. Trading Service: This service is like a yellow page service. It allows clients to find objects based on their properties.

4. Persistence Service: Provides interface for storing objects persistently.

5. Event Service: Object components on the CORBA bus can dynamically register and unregister their interest in receiving specific events. An event-channel is used which collects and distributes events.

6. Concurrency Service: This service provides lock manager which can obtain locks for transactions or threads concurrency.

8. Relationship Service: Allows objects to create dynamic associations between objects that know nothing about each other. This service can be used for referential integrity constraints, containment relationship, etc.

9. Property Service: Allows association of name, value pair to any object. Properties are used to attach something outside of state. Operations to set and retrieve properties are provided.


11. Query Service: Provides query operations for objects.

12. Licensing Service: Needed to monitor use of object services. It provides operations for metering the use of object services so that proper amount can be charged.

Currently, new basic services are being developed by OMG participants.

3.1.2 Common Facilities

These interfaces are horizontally oriented like object services, but they are oriented more toward the end-user applications. An example is Distributed Document Component Facility (DDCF) which allows the presentation and interchange of objects based on a document model. Other common facilities are for system management, task management, and information management.
3.1.3 Domain Interfaces

These interfaces are vertically oriented and are for specific industry application domain. Example of such domain interfaces are CORBAmed for health-care, Telecom for telecommunication, and financial for financial services industry. Multiple boxes for domain interfaces are shown in Fig. 3.1 to indicate this property.

3.1.4 Application Interfaces

These are interfaces developed for a specific application. Since these are application specific interfaces, OMG does not standardize these interfaces. A group of objects that interact with one another can be created that provides customizable solution within an application domain. Such a group of objects is called framework. This dissertation develops a framework for multimedia applications.

3.2 CORBA Components

CORBA is the key component of OMA. Everything revolves around ORB. In this section, logical components of CORBA are described and are shown in Fig. 3.2.

3.2.1 IDL

Interface Definition Language (IDL) is a declarative language that is used to define CORBA object interfaces. An object’s interface, defined using IDL, describes which operations are supported and defines what are the parameters and return value types
for the operations. OMG IDL is language independent. IDL provides a separation of the interface from the object implementation. This is very important in a distributed heterogeneous environment, because all languages may not be supported on all the platforms. An example OMG IDL is:

```idl
module myDiss
{
    exception noSuchName{};
    
    interface goodjob : CosPropertyService::PropertySet
    {
        void write( in string name ) raises (noSuchName);
    }
}
```

IDL has basic types, such as long, double, and boolean. It has constructed types, such as struct and discriminated union, and template types sequence and
string. As shown in the example above, an IDL starts with a module. The module allows grouping of related interfaces. Exceptions and template types are defined first. Then the interface goodjob is defined, which inherits from the PropertySet interface of the CosPropertyService module. The interface goodjob supports an operation write which has one input parameter of type string and it may raise the noSuchName exception. Interface inheritance allows reuse of previously defined interfaces. An object, which inherits an interface, has to support the operations of the inherited interface. The interface inheritance mechanism allows extension of the system while keeping it closed for modification. All CORBA objects inherit from CORBA::Object interface implicitly. This means that the basic operations defined in the CORBA::Object interface are supported by all the objects in the CORBA framework.

3.2.2 Implementation Skeleton

A skeleton provides an interface through which a method receives a request. Requests received by the implementation skeleton are defined statically. The skeleton has a priori knowledge of the interfaces of an object, which are defined in IDL. The skeleton is built into the object implementation. The skeleton cooperates with the ORB to unmarshal a request (converts it from Internet Inter Orb Protocol (IIOP) to a programming language invocation) and dispatches it to the object implementation.
The response goes back in the opposite direction of the request. So the skeleton provides an upcall mechanism of delivering requests to the object implementation.

### 3.2.3 Object Adapter (OA)

Object adapter serves as a glue between an object implementation and the ORB. It adapts the interface of an object implementation to the interface expected by a caller (client). Object adapter acts as a mediator between a client and an object implementation. Object adapter makes it possible to keep the ORB as simple as possible. Object adapter also does the following:

- **Object registration**: It supplies operations that allow implementations to register in the appropriate programming language.

- **Server object activation**: It instantiates and activates objects of the implementation classes if they are not already active. The instances created are dependent upon client requests.

- **Server process activation**: If a server process is not running, it will start a server process in which the server object can be instantiated.

- **Object reference generation**: OA assigns and manages references to object implementations. It provides mapping between the ORB-specific and the implementation-specific representation of object references.
• Call routing: OA cooperates with the ORB and skeleton to deliver requests to the appropriate implementation.

• Multiple connections: OA works with the ORB to handle requests that are received over multiple connections so that no one is blocked.

OA provides flexibility in supporting different implementation styles over a single ORB.

3.2.4 ORB Core

The ORB core delivers requests to the object implementations and returns responses back to the clients who made the requests. It acts as a global bus on which components are attached. The ORB provides a transparent object to object communication in a heterogeneous environment. The ORB interface provides a small set of ORB operations that are common to all objects. The ORB hides the following:

1. Object location: A client gets a reference to an object using the naming or trading service or some other mechanism. It does not know where the object is located. It can be on the same machine or across the network.

2. Object implementation: A client does not know anything about the implementation. It can be in any language and running on any computer platform.
3. Object execution state: A client does not need to know whether the implementation is active or not. If an object is not active, the ORB transparently activates the object using OA.

An object reference refers to a single unique object. An object reference is immutable and opaque, so a client cannot modify it. CORBA defines Generic Inter Orb Protocol (GIOP) for communication between the ORBs. For the Internet, specific version of GIOP is defined which is called the Internet Inter Orb Protocol (IIOP).

3.2.5 Dynamic Invocation

The dynamic invocation by CORBA is supported by two interfaces. The first is the *Dynamic Invocation Interface (DII)* that supports the client side dynamic request and the second is *Dynamic Skeleton Interface (DSI)* that provides the server side dynamic dispatch. These interfaces are not dependent on any object IDL interface. They are provided directly by the ORB. With dynamic invocation, static knowledge of stubs and skeleton of object interfaces is not required. This is useful in a gateway, which doesn't have to be compiled every time an object implementation changes. Interactive programs like browsers can use DII to create dynamic requests based on a user's input. *create_request* operation is supported by every CORBA object, which is used to create a dynamic request. The operation returns a *Request* pseudo-object. A request is created by filling in the appropriate parameters in the pseudo-object and then invoking a request on it.
3.2.6 Client Stub

A stub is created for static invocation of requests. Stubs are created by compiling the IDL of the object interface. The stub is a mechanism that creates and issues a static request on behalf of a client. A client invokes a request in a language-dependent manner which is taken by the stub and it marshals the request by working with the client ORB. The request is then transmitted by the ORB. The stub also returns the return values that are received from the object implementation. The path of a static request and its reply through various CORBA components is shown in Fig. 3.3. A static request goes through client stub to ORB to object adapter to implementation skeleton to object implementation. Response to the request goes back from implementation to implementation skeleton to object adapter to ORB to client stub and finally to the client.
3.2.7 Implementation Repository

The implementation repository provides a run-time repository of information about implementation. It has information about what classes a server supports, what objects are instantiated, and their IDs. All other additional information about implementation can be stored in the implementation repository. Examples of additional information are trace information, security, audit log, and other administrative information.

3.2.8 Interface Repository

A CORBA-based application requires access to the OMG IDL type system to execute. An application uses the type system information to create the proper parameter values of an operation and to discover the types of interfaces supported by the objects. Static knowledge about the interface is incorporated into the code when the IDL is compiled and it is used by the client stub. When creating the dynamic requests, a client has no static knowledge of the object interface; the interface repository is useful in this type of situation.

The interface repository is a run-time database, which has knowledge of the registered component interfaces, their methods, and parameters of the methods. The OMG IDL type system information can be dynamically accessed, stored, and updated using the interface repository operations because the interface repository is a CORBA object. The interface repository organizes type information hierarchically,
and it can be traversed from the top to the bottom to examine the information. The interface information can also be found by using `get_interface` operation on an object reference.
CHAPTER 4

CORBA-based Middleware framework design

This chapter describes the distributed multimedia framework design developed in this work. First, it reviews the object oriented analysis and design principles. Then the detail design of the middleware framework is presented.

4.1 Object Oriented Analysis and Design (OOAD)

Software systems are getting more complex. It is very hard to design and maintain a very large and complex software system using the traditional functional design and programming approach. Multimedia framework and distributed multimedia system applications are complex in nature. The object-oriented analysis and design approach has key features that permits better handling of complex software systems. The major concepts involved are described in the following section.

4.1.1 Key Concepts

- Object: In software, an object is a piece of software that has a unique identity, state, and defined behavior. Objects with similar structures and behavior belong to a common class. Instance of a class is an object.
• Abstraction: Abstraction allows us to deal with complexity. Abstraction keeps the essential details and ignores other details to simplify the understanding of the problem. The essential details kept in abstraction distinguish an object from all other objects. Abstraction provides a clearly defined conceptual boundary relative to the perspective of the viewer. Abstraction gives a more higher level, simpler view of an object.

• Encapsulation: Encapsulation hides all the details of an object that do not contribute to its essential characteristics. Encapsulation is information hiding. This information is the structure of an object, what data elements are part of an object, and details of the implementation of its methods. A user can only use the outside interface of an object and does not have access to an object’s internal details.

• Decomposition: Decomposition means dividing a complex system into parts. Rather than dealing with the whole system, decomposition allows us to deal with the parts of the system at a time, thus allowing greater control of the process. It simplifies the task of building the system.

• Modularity: Modularity is a property of a system which can be built by using loosely coupled, but cohesive modules. There is a well defined boundary between different modules and each module is independent from the other. A modular software system is easier to maintain because it provides plug-and-play
type of capability using software modules. A part of the software system that requires change can be updated without affecting other parts by changing the internals of the affected modules. As long as the interface remains the same, other parts of the software do not need to be changed.

- Hierarchy: A set of abstractions may have a hierarchical relationship among them. By identifying hierarchies, understanding and design of the software is simplified. Hierarchy defines a kind of relationship. Hierarchy is used to go from high-level abstraction to a more detailed abstraction of objects. For example, multimedia device is a more detailed abstraction of device, and camera device is an even more detailed abstraction of multimedia device. So a hierarchy is formed from device to multimedia device to camera device.

- Polymorphism: Polymorphism defines a property that one thing can have many forms. Object-oriented programming languages allow us to use polymorphism in constructing complex software systems. Polymorphism allows a message to be sent to an object, and the result of the response to the message depends on the type of object. According to their behavior specification, different objects behave differently to the same message. Polymorphism helps in the understanding of the written software and also promotes code reuse.
4.1.2 Relationship between objects

There are three major relationships between objects that are used in the DMMF.

These relationships are:

- Aggregation: This is a *whole and part* type of relationship. An aggregation relationship exists when an object is contained in another object. In the physical world for which the software system is being designed, such relationships exist. The contained object is part of the whole object which may contain other objects.

- Inheritance: The inheritance relationship is formed by identifying hierarchy in the abstraction. Inheritance is a *kind of* relationship. Use of inheritance promotes code reuse by writing the common code once in the highest class of the hierarchy and then inheriting it in the subclasses.

- Using: Some objects have neither aggregation nor inheritance relationships among them, but an object or its reference is passed to another object to be used. In using relationship, an object uses other class's methods. It supplies data for the method and receives the result.

4.1.3 Software design steps

In this section, the steps that are taken to build a software system are described. These steps are shown in Fig. 4.1.
1. Requirements: In this phase, detailed requirements of the project are generated. Requirements describe the functionality of the software system and user expectations of the system.

2. Analysis: Once the requirements are defined, the next step is to analyze them carefully. A model of the behavior of the system is created by analyzing the requirements.

3. Design: Once the analysis is done, software architecture is created. Components of the software are designed in detail.

4. Implementation: After the detail design, software is implemented using a programming language.
5. Test: Implemented software system is tested against its functional requirements to see if they are met. Bugs in the software are removed at this stage.

6. Maintenance: Once the software is deployed in a production system, it needs to be maintained for changes or extensions.

In an iterative development cycle, steps 2, 3, 4, and 5 will be repeated.

4.1.4 OOAD steps

The following outlines the steps taken to do object oriented analysis and design of a software system.

1. Gather requirements: This step requires studying the functional requirements and description of the system.

2. Identify objects: Based on the study of the requirements, objects in the system are identified.

3. Responsibilities: Each object identified is assigned attributes and its job in the overall system.

4. Collaboration: Identify the objects that collaborate to meet the identified requirements.

5. Hierarchy: From the identified objects, search for a pattern of hierarchical abstraction. Create the hierarchy of objects.
6. Subsystems: A group of objects that sends very frequent messages to one another but communicates very little outside can be converted into a subsystem. A subsystem contains a cohesive group of objects that communicate among themselves to provide subsystem functionality.

7. Protocols: This step creates the actual methods of the objects. Method semantics and method signatures are defined concretely.

4.2 Overall framework architecture

The multimedia framework was designed using object-oriented analysis and design principles discussed in the previous section. DMMF uses CORBA middleware protocols. General approach was to make each computer a node in the CORBA framework. Each computer that is part of the framework runs an ORB. It was decided to use lifecycle services of CORBA to manage object lifecycle at these nodes. Multimedia data from one node to another consist of a stream, which is described later. DMMF uses different network transport protocols to send multimedia data. Next, DMMF is described in detail.

DMMF architecture uses CORBA lifecycle services to create and destroy media device objects. It also uses CORBA property services to define various properties to the objects involved in the DMMF. As shown in Fig. 4.2, DMMF consists of factories of media devices on various computing platforms in the distributed network.
Each factory is capable of creating various media devices. A factory can be found by a client using *factory finder* or *trading services* of CORBA. *factory finder* is a CORBA object that allows resolution of factory names. When a factory name is given to *factory finder*, it returns a reference to the desired factory. A client can be anywhere in the network. Since we are using CORBA for control and management communications, computers running factory and client need CORBA ORB running inside them. Fig. 4.3 shows the steps involved in the lifecycle of a media object. First, a client goes to a factory finder to get a reference to a factory that can create a media object of the type and location that the client is interested in. Once the client gets the factory reference, it invokes an operation on the factory to create a media device. The factory who receives the request, creates the media object and returns a reference to the media object to the client. Now the client can invoke operations on the media object. When the client is done using the media object, it destroys the object.

### 4.2.1 Stream and Flow

Once media objects are created using factories, a stream of multimedia data is created and started between the media objects. Conceptually, a stream is a continuous transport of multimedia data between media devices. A stream can contain multiple flows. A flow contains media data of a particular type. For example, there can be an audio flow and a video flow. A device for videoconferencing has audio
Figure 4.2: DMMF MediaDevice factory architecture

Figure 4.3: Media object lifecycle
and video flows in its stream. This approach is taken to incorporate multiple media data exchange between the two devices. A stream can be point-to-point or it can be point-to-multipoint. Fig. 4.4 shows the relationship between stream, flow, and media objects.

4.2.2 Framework Objects

In our framework there are four key objects.

- MediaDevice: This object provides abstraction of a device. It can be MicDevice for a microphone, CamDevice for a video camera, etc. MediaDevice acts as a factory for StreamEndPoint and VirtualDevice.

- StreamEndPoint: For each stream, there is a StreamEndPoint object associated with it on both ends. This object accesses the physical media device. It is a termination point for the media transport.
• VirtualDevice (VDev): For each stream, there is a VirtualDevice object created for it by the MediaDevice. VirtualDevice also knows about its associated StreamEndPoint.

• StreamCtrl: StreamCtrl object abstracts continuous media transfer. It provides a single point of control for all the flows within a stream. This object is responsible for converting the application level QoS to network QoS.

4.2.3 A typical connection

Fig. 4.5 describes the connection between all the framework objects for an audio stream. As seen in the figure, on the source side there is MicDevice for a microphone and its associated MicStreamEndPoint and MicVDev. On the sink side, there is SpkDevice for a speaker and its associated SpkStreamEndPoint and SpkVDev. StreamEndPoint access physical devices on the computing platform and it also connects to the network for media transport. A common control for this audio stream, which consists only one audio flow, is provided by the AudioStrCtrl object. This removes the burden of contacting individual objects at the end-point of a stream.

4.2.4 Point-to-multipoint stream

A client creates media devices and binds them to create a point-to-point stream. For each stream connection in DMMF, there is a VirtualDevice object and a StreamEndPoint object. To create a point-to-multipoint (multicast) stream, a client can
pass the reference to the media source device to the new client that wants to join the stream. The new client does not have to create the media source device. The new client has to create a media sink device. Then, the new client has to bind the source device and the sink device. Multicasting can be implemented using two different methods. The first one is to create multiple point-to-point network connections inside the source StreamEndPoint object. The second method is to use multicasting capability of the underlying network. If the network support multicasting, the source StreamEndPoint sends data to the multicast address for the stream. All the sink StreamEndPoint objects will receive data from the multicast address.

4.3 Media Object Interfaces

To provide interoperability between the media objects, it is essential that the interface of the media objects is well defined. We achieve this by using CORBA IDL. An object agrees to provide services for the operations that are part of its IDL. The IDL is independent of any programming language. Implementation that supports
operations defined using the IDL can be written in any language for which there is an IDL to programming language mapping and an ORB available. Appendix A has the IDL used in DMMF. Next, the various interfaces of DMMF are provided in detail.

1. Fig. 4.6 shows the interface inheritance of the media devices. Microphone(Mic), speaker(Spk), camera(Cam), and display(Disp) are the four devices that are currently implemented in the framework. They all inherit their interfaces from the PropertySet interface of CORBA PropertyServices and the LifeCycleObject interface of CORBA LifeCycleServices. This means that all the operations defined in these interfaces have to be supported by the media objects. PropertySet interface allows definition, retrieval, and removal of properties to these objects, while the LifeCycleObject interface allows destruction of these objects. MediaDevice supports the following operations:

- create_A or create_B: This operation is used by the stream control object to create a StreamEndPoint and a VirtualDevice object for a particular
stream connection. Name of the flows that are part of the stream and QoS for the flows is specified.

• bind: This operation is used by the client to establish stream connection between the media devices. QoS, flows, and peer device are passed as a parameter.

• destroy: This operation destroys the media object on which it is invoked.

2. Fig. 4.7 contains an interface inheritance diagram of StreamEndPoint and VirtualDevice interfaces. They both inherit from the PropertySet interface. Multiple StreamEndPoint and VirtualDevice objects can be associated with a media device. StreamEndPoint supports the following operations:

• stop: This operation takes a list of flows as an input and stops them. If it cannot find a flow, it raises an exception.
• start: This operation takes a list of flows as an input and starts them. If it cannot find a flow, it raises an exception.

• destroy: This operation destroys the flows that are specified as its input parameter. If it cannot find a flow, it raises an exception.

• modify_QoS: This operation is used to specify new QoS values for the flows. Actual QoS values met are returned to the requester. It is possible that to change QoS a connection may have to be torn down and reestablished.

• connect: This operation contains a responder StreamEndPoint, a QoS, and flows as parameters. This operation is provided by the source side StreamEndPoint (MicStreamEndPoint, CamStreamEndPoint) in DMMF. Return boolean value tells if the connection was established or not.

• request_connection: This operation contains an initiator StreamEndPoint, a QoS, flows, and whether this connection is a multicast flag as parameters. This operation is provided by the sink side StreamEndPoint (SpkStreamEndPoint, DispStreamEndPoint) in DMMF. Return boolean value tells if the operation was successful or not.

VirtualDevice supports the following operations:
Figure 4.8: Stream interface inheritance

- set_peer: This operation takes as input StreamCtrl, peer VDev, QoS, and flow specifications. This method does verification of network compatibility, flow names, media formats, and media direction. If peer devices are compatible, true is returned; otherwise false is returned.

- modify_QoS: This operation is used to specify new QoS values for flows. Actual QoS values met are returned to the requester.

3. An interface inheritance diagram for StreamControl is shown in Fig. 4.8. Basic_StreamCtrl inherits from PropertySet interface so that all StreamControl objects will support PropertyServices. Specific StreamControl objects, such as AudioStrCtrl for audio stream and VideoStrCtrl for video stream, inherit their IDL interface from Basic_StreamCtrl. Basic_StreamCtrl supports the following operations:
• stop: Stops the flows in the stream to which this object is associated. It will invoke stop on StreamEndPoint of the affected media devices.

• start: Starts the flows in the stream to which this object is associated. It will invoke start on StreamEndPoint of the affected media devices.

• destroy: Destroys the flows in the stream to which this object is associated.

• modify_QoS: This operation enables modification of QoS associated with the flows within the stream.

AudioStrCtrl & VideoStrCtrl supports the following operations:

• bind_devs: This operation takes as input references to the devices that needs to be connected by multimedia stream. Other input parameters are QoS for the flows and flow specification.

• bind: This operation is similar to bind_devs, but instead of using a MediaDevice reference, it uses a StreamEndPoint reference to establish a stream.

• unbind_party: This operation is used to unbind a flow from a particular stream. StreamEndPoint and flow are specified as inputs.

• unbind: This operation unbinds all the flows associated with a particular stream.
4.4 Connection Setup

In this section, the messages that are passed between various objects to establish a multimedia stream connection in DMMF are described. An example of an audio stream setup is shown in Fig. 4.9 so that it is easier to understand, but the messages passed are the same for any stream in the framework. As a result of a bind operation by a client on a source media device (MicDevice), the MicDevice creates an AudioStrCtrl object to provide control operations for the audio stream. The MicDevice makes AudioStrCtrl available through the ORB; so other objects (such as client), who get reference to the AudioStrCtrl, can invoke operations on it. Then, MicDevice invokes bind_devs on AudioStrCtrl and tells it to bind to SpkDevice. The AudioStrCtrl calls create_A on MicDevice to create MicStreamEndPoint and MicVDev for the stream. Then it calls create_B on SpkDevice to create SpkStreamEndPoint and SpkVDev for the audio stream. Once it has references to virtual devices, it calls set_peer on both SpkVDev and MicVDev to make sure that both are compatible and a stream can be established between the devices. AudioStrCtrl invokes connect operation on source StreamEndPoint (MicStreamEndPoint). This causes the source StreamEndPoint to invoke request_connection on sink StreamEndPoint (SpkStreamEndPoint). If successful, success is returned by both the sink and the source StreamEndPoint. The StreamCtrl returns success to the source MediaDevice which invoked bind_devs. Details of the connect operation is described next.
Figure 4.9: Message flows during bind_devs operation

Figure 4.10: Connect operation steps
Details of a connection setup are shown in Fig. 4.10. As a result of connect call from StreamCtrl, source StreamEndPoint (MicStreamEndPoint) invokes request_connection on sink StreamEndPoint with forward flow_spec. Grammar for the flow_spec is given in [8]. The simple flow protocol (SFP) described in [8] is not used. Flow protocol used in DMMF (UAFP) is based on RTP [10]; packet formats for UAFP are described in the next two sections. To be able to understand the format of the media transport packets, it is important that the devices use a common protocol negotiated through flow_specs. In the case of an audio stream, the forward flow_spec is given as:

```
audio1\in\MIME:audio/mu-law\UAFP1.0\\TCP=gpacs1.ece.arizona.edu:4020
```

The flow_spec tells the sink StreamEndPoint (SpkStreamEndPoint) to connect flow “audio1”. The direction of the flow is into the sink, the format for audio media is \(\mu\)-law, and the flow protocol is UAFP. The flow_spec also tells the sink to use TCP protocol with the given host address and port to send credits. The sink StreamEndPoint inspects the forward flow_spec and returns the following reverse flow_spec for the audio stream:

```
audio1\UDP=pcpacs2.ece.arizona.edu:3142\64
```

It says use UDP transport protocol, use address pcpacs2.ece.arizona.edu at port 3142 to send audio1 flow data. Initial credit given to the source is 64. In this design, 1 credit is a 512 bytes \(\mu\)-law compressed audio sample.
For video stream, the forward flow_spec is given as:

```
video1\in\MIME:video/mjpeg\UAFPl.0\TCP=gpac\ece.arizona.edu:4021
```

The flow_spec tells the sink for video, which is StreamEndPoint for display to connect
flow "video1". The direction of the flow is into the sink (display), the format for
video media is MJPEG, and the flow protocol is UAFP. The flow_spec also tells the
video sink to use TCP transport protocol with the given host address and port to
send credits. DispStreamEndPoint inspects the forward flow_spec and returns the
following reverse flow_spec for the video stream:

```
video1\UDP=pcpacs2.ece.arizona.edu:3143\16
```

With this flow_spec, display tells camera to use UDP transport protocol, use address
pcpacs2.ece.arizona.edu at port 3143 to send video1 flow data. Initial credits given to
the camera are 16. In this design, 1 credit in a MJPEG format video flow is equal to
1 JPEG frame. Flow_spec is an important part of the architecture because it allows
a source and a sink device to negotiate transport, media format, flow control, and
flow direction related parameters.

### 4.4.1 Credit mechanism in UAFP

Credits are used in UAFP to control the flow of media data from a sender to a
receiver. It is possible that in a flow, the receiver may be slower than the sender
or the receiver may have less buffer space. Credit mechanism is used to prevent
the sender from over running the receiver in a flow. During the flow setup, the sender is given some initial credits by the receiver. One credit is equal to one media packet in DMMF. The sender sends media packets only if it has credits. The receiver periodically sends more credits to the sender to keep the flow of media data going. If the sender has no credits, it waits for the credits before sending more media data packets.

4.5 Audio Packet

Fig. 4.11 shows the format of an audio packet in UAFP. The total size of an audio packet is 535 bytes. Bytes 0-9 are used for the sequence number, 11-22 are used for
the timestamp, and 23-534 are used for the audio sample. Byte 10 tells how many of the 0-9 bytes are used for the sequence number.

4.6 Video Packet

Fig. 4.12 shows the format of a video packet in UAFP. Bytes 0-10 are used for sequence number information, 11-16 are used for image size, byte 17 tells how many of 11-16 bytes are used for the image size, 18-29 are used for timestamp, and the rest of the bytes are for the video frame. Compression of a video frame varies based on the details in the frame, so the size of a video packet is variable. Currently, DMMF uses MJEPG as the video format.
CHAPTER 5

CORBA Implementation of Multimedia Streams

DMMF was implemented using Visibroker 2.5 JAVA ORB and JDK 1.1.3 (JAVA Development Kit). It was implemented on Solaris 2.5.1 environment and Windows NT 4.0 to demonstrate its use on heterogeneous computing platforms. JNI (JAVA to Native Interface) was used to create JAVA classes that are not available in JDK. This chapter describes the JAVA programming environment, JAVA processes, and threads used in the framework. Mapping of IDL to JAVA and CORBA PropertyService is described in section 5.2 and 5.3. Then, class diagrams of the components of the framework using Booch notation [3] are described in detail. Legend section of Fig. 5.2 shows the terminology used for the class diagrams in this chapter.

5.1 JAVA Programming Environment

JAVA is the latest programming language by Sun Microsystems. In the next section, important features of JAVA are described.

5.1.1 JAVA Features

- Object-Oriented methodology: Object-Oriented analysis and design of software allows more modular design. This method of modelling resembles to the real life
environment. Data are encapsulated in the object and protection is provided for private data. Clean interface is provided to access the data. Only methods that are part of an object can change and access its private data.

- C++ like Syntax: JAVA has removed bad parts (pointers and global variables) of C++, but has kept its syntax similar to C++ so it is very easy to learn.

- Platform Independence: Once JAVA code is compiled into byte code, it can run in JAVA interpreter. A JAVA interpreter is available for many operating systems and hardware platforms. This is specially helpful for DMMF because it should be accessible to everyone regardless of their computing platform.

- JAVA packages: User interface, input/output, network connection, and socket interface functions are nicely converted into packages in JAVA. It makes it very easy to write distributed internet based application. Programming for a graphical user interface (GUI) is also at a more abstract level than using individual toolkits for a particular platform. Various classes in the packages support these functionalities.

- Run-time linking: When a JAVA code is compiled, byte code is created. There is no compile-link-load-test-debug cycle involved in JAVA. You just compile your code and run it in a JAVA interpreter or a JAVA-enabled browser.
• Security: JAVA has multiple security checks going from source code to execution. The JAVA compiler ensures that a source code meets the safety rules. Byte codes are verified by the run-time system when they are executed. Class loader checks safety related to class name space and access when loaded into the system. Access to the local file system is not allowed from applets, so applets cannot write any viruses to the local hard disk.

• Memory Management: JAVA has an automatic garbage collection of memory that is not explicitly freed by the program. This relieves the programmer from worrying about memory management.

• Multiple Threads: JAVA provides thread support. This is particularly important for DMMF because it deals with multimedia data. Separate threads can be assigned to deal with different types of data. Because thread switching has a lower overhead than a process context switching, the application can run faster.

• JAVA applets: JAVA programs can run stand alone or can be run in a browser as part of the HTML page. The JAVA program that runs in a browser is called an applet. Applets provide the capability of dynamically downloading the code and running it in the browser. This approach eliminates user maintenance of software versions.
5.1.2 JAVA Threads

In this section, the JAVA multithreading capability is explored in more detail. When a program executes in a single threaded environment, a process is created. The process sequentially goes through initialization, method calls etc., until the processing task is complete or the program exits. Threads give parallelism in execution, while still staying within the boundary of a single process. The code in a separate thread executes independently from other threads. When performing synchronous tasks that waste CPU time by staying idle for long time, these tasks can be put in threads to allow overlapping of execution. Threads of a particular process share global resources of that process. Switching from one thread to another is less taxing on the system than switching between processes. Using multi-threading, performance of a software can be improved.

In JAVA, classes can be inherited from the Thread class. A subclass of the Thread class is capable of executing in a separate thread. So the interface provided to use the threads is very easy. Here are some more details.

- Thread Synchronization

Threads are used to execute code in parallel and independently, but still there is a need for threads to share data and communicate with each other. If proper care is not taken, shared data can be changed by one thread before it is used by another thread. For this purpose, synchronization among threads is necessary.
Part of the code that accesses shared resources is called the *critical section* of the code. JAVA provides "synchronize" keyword to define the *critical section* of the code. Only one thread can execute the code that is defined between the synchronized block. If one thread is executing in the *critical section* of the code, other threads have to wait until the *critical section* execution is completed. Deadlock should be avoided.

- **Thread states**

Threads at a particular point in time can be in one of the following states:

1. runnable: This thread is waiting to be run by the scheduler.

2. new thread: This thread was just created. When it is started, it will become runnable.

3. not runnable: A Thread is waiting, sleeping, blocked or suspended. This thread cannot be scheduled to run at the current time.

4. dead: This thread has finished execution. It cannot be run anymore.

5. running: This thread is currently executing.

- **Thread Scheduling**

The scheduler in a system decides the order in which runnable threads run. There are two prominent ways of scheduling.
Preemptive scheduling: In this method, the currently executing thread is removed after some time of execution and a new thread is started. All the state information for the previous thread is saved so when it is started again, its previous state can be restored. All runnable threads are run according to a specified scheduling algorithm used by the scheduler.

Non-preemptive scheduling: In this method, the currently running thread is replaced with other thread only when the running thread exits or yields to another threads. JAVA implements preemptive scheduling.

5.2 IDL to JAVA

Interfaces in CORBA are defined using IDL. DMMF is implemented in JAVA, so the interfaces of the objects in the framework have to be converted to JAVA. The idl2java compiler provided by Visibroker is used to compile an IDL file to JAVA classes. In this section part of the IDL to JAVA mapping is described. A name in IDL is converted to a name in JAVA. A module in IDL is converted to a JAVA package. Constants defined in the IDL are made public static variables in JAVA, meaning they cannot be changed. Table 5.1 shows mapping IDL basic types to JAVA types. Table 5.2 shows mapping from IDL to JAVA for constructed types.

An IDL interface is mapped to a JAVA interface and the IDL exceptions are mapped to exception classes derived from the java.lang.exception class. The compiler creates Holder and Helper classes for the JAVA classes that are created from the IDL.
Table 5.1: Mapping of IDL basic types to JAVA types

<table>
<thead>
<tr>
<th>IDL type</th>
<th>JAVA type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>char</td>
<td>char</td>
</tr>
<tr>
<td>octet</td>
<td>byte</td>
</tr>
<tr>
<td>string</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>short</td>
<td>short</td>
</tr>
<tr>
<td>unsigned short</td>
<td>short</td>
</tr>
<tr>
<td>long</td>
<td>int</td>
</tr>
<tr>
<td>unsigned long</td>
<td>int</td>
</tr>
<tr>
<td>long long</td>
<td>long</td>
</tr>
<tr>
<td>unsigned long long</td>
<td>long</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
</tr>
</tbody>
</table>

Table 5.2: Mapping of IDL constructed types to JAVA types

<table>
<thead>
<tr>
<th>IDL type</th>
<th>JAVA type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence</td>
<td>array</td>
</tr>
<tr>
<td>array</td>
<td>array</td>
</tr>
<tr>
<td>enum</td>
<td>final JAVA class</td>
</tr>
<tr>
<td>struct</td>
<td>final JAVA class</td>
</tr>
<tr>
<td>union</td>
<td>final JAVA class</td>
</tr>
</tbody>
</table>
The Holder classes are used to hold a particular class when it is used as a "inout" or "out" parameter in IDL. The Helper classes are used to convert a particular class to and from CORBA.Any class. A _tie_X class is used to prepare class X to receive the operation invocation requests through the ORB when implementation of class X does not inherit from its ORB skeleton class. Next, an example IDL and its mapping to JAVA is shown:

```java
//example IDL
module myDiss
{
    exception noSuchName{};
    interface yourscore
    {
        void result (in string name) raises (noSuchName);
        boolean putscore (in string name, in long score);
    };
};

//mapping of an IDL interface to a JAVA interface
package myDiss;
public interface yourscore extends org.omg.CORBA.Object
{  
    public void result (java.lang.String name) throws  
        myDiss.noSuchName;  
    public boolean putscore (java.lang.String name, int score);  
}

5.3 CORBA PropertyService

Property is defined as a name,value pair. The value is class CORBA_Any and the
name is of type String. PropertyService is helpful when a user wants to associate
some attribute to an object, but it is not part of the object type. PropertyService
allows a client to set and get property values. It also allows a client to dynamically
create and destroy properties. Visibroker does not provide implementation of Prop­
ertyService, so a subset of it was implemented for DMMF. An object that supports
PropertyService should implement the operations defined in PropertySet interface.
PropertySet interface is supported by objects in DMMF. It supports the following
operations:

- define_property: to define a property using name,value pair.
- define_properties: to define a list of properties.
- get_number_of_properties: to get the value of the number properties defined for
  a particular object.
• get_all_property_names: returns a list of property names.

• get_property_value: takes property name as an input and returns its value.

• get_properties: returns property values associated with a list of property names.

• get_all_properties: returns all the properties of an object.

• delete_property: allows a client to delete a property by name.

• delete_properties: deletes all the properties specified in a list of names.

• delete_all_properties: deletes all the properties of an object.

• is_property_defined: returns true if property is defined. Input is property name.

5.4 Processes and threads

Fig. 5.1 shows process, threads, and objects at a particular node in the DMMF. Each node runs a server process which makes services of a factory object available through an ORB. Using the operations of a factory object, a client application can create media device objects. During the binding process between two media devices, StreamEndPoint and VDev objects are created by media devices. StreamEndPoint object has 2 threads. These threads are used for transport and flow control communications, so there is one process and multiple threads in the current architecture implementation.
Figure 5.1: Process, threads, and objects at a node
5.5 Device

In this section, a detailed class diagram of the media device objects is shown. In the object diagram figures, objects with dashed line are generated by the idl2java compiler. The objects with solid line are implemented as part of the framework; objects with dark dashed line are implemented in JAVA and they interface with C language code.

• MicDevice

In Fig. 5.2, a class diagram of MicDeviceImpl is shown. When the IDL is converted to JAVA using the IDL-to-JAVA compiler, it creates abstract classes for MicDevice, MicStreamEndPoint, and MicVDev. MicDeviceImpl, which is the implementation of MicDevice, implements MicDevice operations and it inherits from the PropertySetImpl object, so that it can reuse the code of the PropertySet object to support the PropertyServices operations. MicDeviceImpl contains implementation of MicVDev and MicStreamEndPoint. It uses AudioStrCtrlHelper, streamQoSHolder, MicVDevHolder, MicDeviceHelper, CORBA ORB, and CORBA Any. It also uses _tie_MicStreamEndPoint, _tie_AudioStrCtrl, CORBA BOA, MicStreamEndPointHelper, MicVDevHelper, and _tie_MicVDev, which are created by the IDL-to-JAVA compiler. MicDeviceImpl also uses SpkDevice, AudioStrCtrlImpl, MicStreamEndPointImpl, and MicVDevImpl classes.
Figure 5.2: MicDeviceImpl object diagram
• SpkDevice

The object diagram of SpkDevice, which is a sink device is shown in Fig. 5.3. SpkDeviceImpl, which is the implementation of SpkDevice, implements SpkDevice operations and it inherits from the PropertySetImpl object so that it can reuse the code of PropertySet object to support PropertyServices operations. SpkDeviceImpl contains the implementation of SpkVDev and SpkStreamEndPoint. It uses AudioStrCtrlHelper, streamQoSHolder, SpkVDevHolder, MicDevice, SpkDeviceHelper, CORBA.ORB, CORBA.Any, AudioStrCtrlImpl, _tie_SpkStreamEndPoint, _tie_AudioStrCtrl, CORBA.BoA, SpkStreamEndPointHelper, SpkVDevHelper, SpkStreamEndPointImpl, SpkVDevImpl and _tie_SpkVDev.

• CamDevice

CamDevice abstracts the camera media device and it is implemented by CamDeviceImpl object, shown in Fig. 5.4. CamDeviceImpl inherits from the PropertySetImpl object and it implements operations defined in the CamDevice interface. CamDeviceImpl contains the implementation of CamVDev and CamStreamEndPoint. It uses VideoStrCtrlHelper, streamQoSHolder, CamVDevHolder, CamDeviceHelper, CORBA.ORB, CORBA.Any, _tie_CamStreamEndPoint, _tie_VideoStrCtrl, CORBA.BoA, CamStreamEndPointHelper, CamVDevHelper, and _tie_CamVDev which are created by the
Figure 5.3: SpkDeviceImpl object diagram
Figure 5.4: CamDeviceImpl object diagram

IDL-to-JAVA compiler. CamDeviceImpl also uses DispDevice, VideoStrCtrlImpl, CamStreamEndPointImpl, and CamVDevImpl.

- DispDevice

The DispDevice object abstracts the display device in the framework. It is implemented by the DispDeviceImpl object. Fig. 5.5 shows the object design of DispDeviceImpl. This device implementation inherits from PropertySetImpl to support the PropertySet operations by reusing code and implements the operations defined in the DispDevice abstract class. DispDeviceImpl contains
Figure 5.5: DispDeviceImpl object diagram

the implementation of DispVDev and DispStreamEndPoint. It uses VideoStrCtrlHelper, streamQoSHolder, DispVDevHolder, CamDevice, DispDeviceHelper, CORBA_ORB, CORBA.Any, VideoStrCtrlImpl, _tie_DispStreamEndPoint, _tie_VideoStrCtrl, CORBA_BOA, DispStreamEndPointHelper, DispVDevHelper, DispStreamEndPointImpl, DispVDevImpl and _tie_DispVDev.
5.6 VirtualDevice

This section describes in detail the design and implementation of the VirtualDevice objects in the framework. In the object diagram figures, objects with dashed line are generated by the idl2java compiler. The objects with solid line are implemented as part of the framework; objects with dark dashed line are implemented in JAVA and they interface with C language code.

- MicVDev

Fig. 5.6 shows the class diagram of a microphone (source) VirtualDevice implementation called MicVDevImpl. It inherits from MicVDev and PropertySetImpl. It uses the streamQoSHolder, Property, CORBA_Any, CORBA_ORB, SpkVDev, AudioStr Ctrl, and flowStatus classes.

- SpkVDev
Fig. 5.7 shows the class diagram of a speaker (sink) VirtualDevice called SpkVDevImpl. It inherits from the SpkVDev abstract class and PropertySetImpl. It uses flowStatus, streamQoSHolder, Property, CORBA_Any, CORBA_ORB, MicVDev, and AudioStrCtrl.

- CamVDev

Fig. 5.8 shows the object diagram of the CamVDevImpl object which implements the CamVDev object. CamVDev represents a virtual device for a camera device. A camera virtual device is created for each video stream connection. It negotiates formats with its peer display virtual device. CamVDevImpl inherits from PropertySetImpl and implements the operations of CamVDev. It uses the streamQoSHolder, Property, CORBA_Any, CORBA_ORB, DispVDev, VideoStrCtrl, and flowStatus classes.
• DispVDev

The object diagram of DispVDevImpl, which implements a display virtual device, is shown in Fig. 5.9. It inherits from the DispVDev abstract class and PropertySetImpl. It uses flowStatus, streamQoSHolder, Property, CORBA_Any, CORBA_ORB, CamVDev, and VideoStrCtrl.

5.7 StreamEndPoint

This section describes in detail the design and implementation of the StreamEndPoint objects in the framework. In the object diagram figures, objects with dashed line are generated by the idl2java compiler. The objects with solid line are implemented as part of the framework; objects with dark dashed line are implemented in JAVA and they interface with C language code.
Fig. 5.9: DispVDevImpl object diagram

- MicStreamEndPoint

Fig. 5.10 shows a class diagram of a microphone (source) StreamEndPoint object implementation, which is MicStreamEndPointImpl. It inherits from MicStreamEndPoint and PropertySetImpl. It uses CORBA_Any, CORBA_BOA, CORBA_ORB, streamQoSHolder, Property, MicVDevHelper, MicVDev, MicStreamEndPointHelper, SpkStreamEndPoint, and flowspecHolder. MicStreamEndPointImpl contains the Talkmic and credit_listen objects.

The class diagram of credit_listen is shown in Fig. 5.11. This class inherits from the Thread class and uses the DatagramPacket class. It contains the Socket, MicStreamEndPointImpl, DatagramSocket, DataInputStream, and ServerSocket classes. This class listens for credit from a speaker (sink) so that more audio media data can be sent. The flow chart of the algorithm
Figure 5.10: MicStreamEndPointImpl object diagram

Figure 5.11: Microphone credit_listen thread object diagram
create server socket

wait for connection

create receive socket

read new_credit

Can enter critical section?

no

yes

enter critical section

old_credit = credit
credit = credit + new_credit

old_credit = 0?

no

yes

notify data sender to start

leave critical section

Figure 5.12: Source credit listen thread algorithm
of credit_listen is described in Fig. 5.12. For a TCP-based credit transport, it first creates a ServerSocket and waits for a connection from the sink. It blocks until it gets a credit from the sink. When it gets the credit, it waits to enter the critical section. After entering the critical section, it saves the value of the current credit in the old_credit and adds the new credit value to the credit variable. If the old_credit value is 0, meaning that before this credit was acquired there was no credit left, the thread that sends the media data must be blocked. If this is the case, then it notifies the sender and leaves the critical section; otherwise it leaves the critical section. After leaving the critical section, it waits for a new credit value from the sink.

The class diagram of Talkmic is shown in Fig. 5.13. It inherits from the Thread class. It contains MicStreamEndPointImpl, FileInputStream for reading audio samples, and DatagramPacket to send audio packets. It uses DatagramSocket,
Date, and InetAddress. The Date class is used to get the timestamp value. DatagramSocket is used to send audio packets using UDP and MicStreamEndPointImpl is used for synchronization. The flow chart of the algorithm of Talkmic is described in Fig. 5.14. Initially, Talkmic parses the transport protocol and the address to be used to send data to the sink. Then it creates a DatagramSocket. Initially, it waits for credits. After that, it checks if credit is greater than 0. If not, it blocks; otherwise it checks credit values, sends audio data, and decreases credit value in a loop using the JAVA thread synchronization capability to avoid race conditions with credit.listen. After credit value is 0, it blocks and waits for credit.listen to wake it up.

- SpkStreamEndPoint

The class diagram of the implementation of a speaker (sink) StreamEndPoint, in this case SpkStreamEndPointImpl for a speaker, is shown in Fig. 5.15. SpkStreamEndPointImpl inherits from SpkStreamEndPoint and PropertySetImpl. It contains the Playspk object which plays audio and the Listenspk object that receives audio over the network. It uses CORBA.Any, CORBA.Boa, CORBA.ORB, streamQoSHolder, Property, SpkVDevHelper, SpkVDev, SpkStreamEndPointHelper, MicStreamEndPoint, and flowspecHolder. Now the two classes that make the speaker StreamEndPoint different from the microphone StreamEndPoint are looked at in detail.
Figure 5.14: Source datasync send thread algorithm

**parse protocol type, address & port**

create datagram socket to send data

Can enter critical section?

yes

<table>
<thead>
<tr>
<th>enter critical section</th>
</tr>
</thead>
</table>

credit = 0?

no

| wait for credit |

yes

| leave critical section |

credit > 0?

no

| syn. credit — |

yes

| send data packet |

syn. = synchronized access to shared variable using JAVA
The object diagram of ListenSpk is shown in Fig. 5.16. It inherits from the JAVA Thread class. It uses the Date and InetAddress class. ListenSpk contains DatagramSocket, DatagramPacket, and SpkStreamEndPointImpl. The algorithm of the thread that receives media data packets in a sink device, in this case ListenSpk, is shown in Fig. 5.17. This thread creates a datagram socket to receive media packets. After receiving the media data, it stores the data in a shared memory buffer in the StreamEndPoint class using the synchronization mechanism. It also updates the two variables that keep track of how much data are in the buffer to play and how many credits the source has left using the synchronization mechanism. If packets to play after receiving the current packet is just one, then the buffer was empty. It notifies the play...
thread that the media data to play are now available; otherwise, no notification is necessary. After this, it goes back and waits to receive more packets.

Fig. 5.18 describes the object diagram of the object Playspk. It inherits from the Thread class. It uses InetAddress for network communication to send credits, the Date class to get delay value, and the AudioPlayer class to play the audio. It contains SpkStreamEndPointImpl for accessing the network address and synchronization; it also contains DataOutputStream, DatagramPacket, DatagramSocket, and Socket for network communications. It contains JavaFonPipeIn and JavaFonPipeOut to connect itself with AudioPlayer. The flow chart of the algorithm used by the play thread in a sink is shown in Fig. 5.19. It first parses the network address and then creates a socket connection to that address to send credits. It accesses shared data (pkts_to_play) mutually

Figure 5.16: Speaker data receive thread object diagram
create DatagramSocket

receive data packet

syn.
save data

syn.
update source_credit
pkts_to_play++

syn.

pkts_to_play = 1?

yes

syn.
notify player

no

syn. = synchronized access to shared variable using JAVA

Figure 5.17: Algorithm for sink receive thread
exclusive to see if there are any media data packets to play. If there are none, then it blocks until notified by the receive thread. If there are packets to play, it writes a packet to the media device and updates the value of how many packets are left to play. Next, it checks if it is the time to send a credit value to the source. Whether it sends a credit or not, it goes back in the loop to play more data packets. When no more data packets are left to play, it blocks and waits for the receive thread.

- CamStreamEndPoint

Figure 5.18: Object diagram for speaker (sink) play thread
Figure 5.19: Algorithm for media play thread in sink
Fig. 5.20: CamStreamEndPointImpl object diagram

Fig. 5.20 shows the class diagram of a camera (source) StreamEndPoint object implementation, which is CamStreamEndPointImpl. It inherits from CamStreamEndPoint and PropertySetImpl. It uses CORBA_ANY, CORBA_BOA, CORBA_ORB, streamQoSHolder, Property, CamVDevHelper, CamVDev, CamStreamEndPointHelper, DispStreamEndPoint, and flowspecHolder. CamStreamEndPointImpl contains the Sendcam and the cam_credit_listen objects.

Fig. 5.21 depicts the cam_credit_listen object, which listens for credits from a display to send more JPEG video frames. This object runs in its own thread. It inherits from the Thread class and uses the DatagramPacket class to receive the video frame packets. It contains the Socket, CamStreamEndPointImpl, DatagramSocket, DataInputStream, and the ServerSocket objects. The algorithm
for this object is similar to that described in Fig. 5.12. Credits are received over a TCP connection established by the display.

The object diagram of the Sendcam object, which sends video frames over the network, is shown in Fig. 5.22. It inherits from the Thread class and uses the Date class, InetAddress, and the DatagramSocket object that are part of JDK. It contains physically or has reference to Camera, DatagramPacket, and CamStreamEndPointImpl. The semantic for this object is similar to that depicted in Fig. 5.14. This object gets video frames from the Camera object and then it transmits it to the display over the network. The camera object provides JAVA methods to initialize and destroy the physical camera device.

The size of the video frame is determined by the initialization method. The
camera object also provides a JAVA method to capture a JPEG compressed frame from the physical camera. It uses JAVA-to-Native-Interface to interface the JAVA code to the C language code to capture the frame. Once, a frame is captured by a C library routine, it is copied from the C code memory to the JAVA virtual machine memory, so that it can be used by the JAVA classes. If there is no credit, it waits for the credit-listen thread to wake it up.

- **DispStreamEndPoint**

The class diagram of the implementation of a display (video sink) StreamEndPoint, in this case DispStreamEndPointImpl, is shown in Fig. 5.23. DispStreamEndPointImpl inherits from DispStreamEndPoint and PropertySetImpl. It contains the DispFrame object which displays the video JPEG frames and the Listendisp object that receives the video over the network. It uses CORBA_Any,
CORBA_BOA, CORBA_ORB, streamQoSHolder, Property, DispVDevHelper, DispVDev, DispStreamEndPointHelper, CamStreamEndPoint, and flowspecHolder. Now, the two objects Listendisp and DispFrame that make display StreamEndPoint different from other StreamEndPoints are looked at in detail. The first is the details of Listendisp and the second is the details of DispFrame.

Fig. 5.24 shows the object diagram of Listendisp. This object is used by the display stream endpoint to receive the video frames. It executes in a separate thread. It inherits from the JAVA Thread class. It uses the Date and the InetAddress classes. It contains DatagramSocket to receive datagrams, and it
Figure 5.24: Design of Listendisp object that receives video

contains DatagramPacket to put in the received datagram. It contains reference
to DispStreamEndPointImpl to access common information and to synchronize
with the display frame. The algorithm for the display listen thread is similar
to that of the speaker listen thread. It receives video frames and puts it in the
common buffer so that DispFrame can access it. If DispFrame is blocked, then
it is notified by Listendisp.

The DispFrame object design is shown in Fig. 5.25. DispFrame displays the
video frames that are received by the Listendisp object. DispFrame also ex¬
ceutes in its own thread, so it inherits from the Thread class. It uses the
InetAddress and the Date JAVA classes to send credit to the sender of the
video. DispFrame buffers three images in the JAVA Image class. Each im¬
age class contains one decompressed JPEG frame. It also contains reference to
Figure 5.25: Object diagram of DispFrame class

DispStreamEndPointImpl to access common information and to share the video frames. DispFrame contains a MediaTracker object which it uses to track the progress of JPEG frame decompression. DataOutputStream, DatagramSocket, Socket and DatagramPacket are used for sending credits and for performance measurement. The MyFrame object has the mechanism to display JPEG image on the screen. The algorithm of DispFrame is similar to that shown in Fig. 5.19. The MyFrame object is described next.

The MyFrame object inherits from the Frame object to get the functionality of a frame on the computer screen. It uses the GridLayout object to properly place the MCanvas object within the frame so that the frame will fully cover
the canvas. It contains the Label, Image and MCanvas objects. The MCanvas object is described next. It is used to paint the video frame image on the screen. The Image object holds the image to be displayed on the screen/canvas. Fig. 5.26 shows the object design of the MyFrame class.

Fig. 5.27 shows the design of the MCanvas class. It inherits from the JAVA Canvas class to provide canvas on which an image can be painted. It also contains the Image object. This image is supplied by MyFrame, which, in turn, received it from DispFrame. The only purpose of this object is to allow the painting of the video frame images.
5.8 Synchronization of media streams

Microphone, speaker, camera, and display media devices are implemented in DMMF. These devices either support an audio or a video stream. The audio and video stream are independent of each other. SpkStreamEndPointImpl object plays audio and DispStreamEndPointImpl object displays video. To synchronize audio and video streams, two new devices called srcaudiovideo device and sinkaudiovideo device should be created in DMMF. A stream between these two devices will have two flows, audio and video. Both audio and video data will be sent with time stamp information by srcaudiovideo StreamEndPoint. On the receiving side, sinkaudiovideo StreamEndPoint will use the time stamp information from audio and video data to synchronize them before it sends them to the physical media device. This way, synchronization between audio and video media can be achieved.
5.9 StreamControl

The StreamControl object in DMMF provides all the stream control functionality at a single point. In the object diagram figures, objects with dashed line are generated by the idl2java compiler. The objects with solid line are implemented as part of the framework; objects with dark dashed line are implemented in JAVA and they interface with C language code. Now, the design of the audio and the video stream control objects are given:

- Audio stream control

The object diagram of the StreamControl implementation for audio is shown in Fig. 5.28. AudioStrCtrlImpl inherits from AudioStrCtrl and PropertySetImpl. The abstract class AudioStrCtrl has all the operations that need to be implemented. It contains MicVDev, SpkVDev, SpkStreamEndPoint, and MicStreamEndPoint. The StreamControl object communicates with these four objects to establish, control and destroy a media stream. It uses streamQoSHolder, MicVDevHolder, SpkDevice, SpkVDevHolder, CORBA_Any. It also uses CORBA_ORB, CORBA_BOA, MicDevice, SpkVDevHelper, MicVDevHelper, and AudioStrCtrlHelper.

- Video stream control

The object diagram of the StreamControl implementation for video, which is VideoStrCtrlImpl is shown in Fig. 5.29. It inherits from VideoStrCtrl and
Figure 5.28: Object diagram of audio stream control
Figure 5.29: Object diagram of video stream control
PropertySetImpl. The abstract class VideoStrCtrl defines the operations that
the video stream control object needs to support in its implementation. It
contains the CamVDev, CamStreamEndPoint, DispVDev, and the DispStream-
EndPoint objects. These objects are used to connect, control, and destroy a
video stream. The video stream control implementation uses streamQoSHolder,
CamVDevHolder, DispDevice, DispVDevHolder, CORBA.Any, CORBA.ORB,
CORBA.BoA, CamDevice, DispVDevHelper, CamVDevHelper, and
VideoStrCtrlHelper.

5.10 Prototype Application

Several prototype applications were created in JAVA to show the usability of
the framework and to measure the performance of the various operations involved
in the framework. The prototype applications established point-to-point audio and
video streams between two computers in a LAN and WAN environment. The steps
involved in creating a distributed multimedia application using DMMF are shown in
the next section. The applications contained audio only, video only, or audio and
video streams. The LAN environment in which the prototype applications were run
is shown in Fig. 5.30. Now, the applications are described in detail.
Figure 5.30: LAN environment

Figure 5.31: Prototype application client messages in DMMF
5.10.1 Audio

Fig. 5.31 shows the messages that are sent between a client application and DMMF objects. In this particular case, an audio stream is established between a microphone on \textit{gpacs1} and a speaker on \textit{pcpacs2}, which are shown in Fig. 5.30. First, the client goes to factory finder object and binds to factory1 on \textit{gpacs1}. Second, the client binds to factory2 on a second computer (\textit{pcpacs2}) that will be involved in the stream. Next, the client issues a request to create the microphone device to factory1 and gets a reference to the microphone device on a computer on which factory1 is running. After that, the client issues a request to create a speaker device to factory2. Factory2 responds by creating a speaker device on the machine it is running and returns a reference to the speaker to the client. At this point, the client has references to the microphone and the speaker. Now, by invoking the bind operation on the microphone, the client tells the microphone to bind to the speaker. Microphone goes through messages described in Fig. 4.9 and returns the reference to the audio stream control object (AudioStrCtrl) to the client. At this point, the prototype application displays its GUI, which is shown in Fig. 5.32. A user starts the audio stream by clicking on the “start” button; after a while, he/she stops the stream by clicking on the “stop” button. The client application invokes start and stop operations on the audio stream control object respectively. The application is exited by selecting “done” in the upper left corner of the GUI. At this point, the application tells DMMF to destroy all the media and related objects.
5.10.2 Video

The prototype application for creating a video stream in DMMF involves the steps that are similar to that shown for audio in Fig. 5.31. In the case of video, after getting references to the factory objects, the prototype application creates a camera device on one machine and a display device on the other machine. It then tells the camera device to bind to the display device. Negotiation takes place and the reference to the video stream control object (VideoStrCtrl) is returned to the client. Now control of the video stream is possible. The prototype application displays the GUI of Fig. 5.32. When a user clicks on “start” or “stop”, the application invokes start or stop operation on the VideoStrCtrl object. Fig. 5.33 shows the video display window as it appears on the computer screen. The video window size is 210x160 pixels. Video format currently implemented in DMMF is MJPEG.
5.10.3 Audio and Video

The third type of prototype application creates the microphone and camera media objects on one machine and it creates the display and speaker media objects on another machine. It then binds the microphone with the speaker and the camera with the display device. The client application receives the reference to the AudioStrCtrl object to control the audio stream and the VideoStrCtrl object to control the video stream. The same GUI as shown in Fig. 5.32 is displayed. User actions on the GUI start and stop both audio and video streams. Upon exit from the application, all the audio and video media objects are destroyed by DMMF.
CHAPTER 6

Performance Evaluation of Framework Implementation

Using the prototype applications described in the previous chapter, the performance of the middleware framework was measured as described in this chapter. The time it took for various operations, from operation invocation to return was measured in the client. End-to-end delay was measured for audio and video. End-to-end delay means difference in time from media capture to media presentation. Protocol stack for the performance evaluation experiment is shown in Fig. 6.1.

6.1 Environment

6.1.1 LAN

In the LAN environment shown in Fig. 5.30, audio and video streams were sent from the SUN SPARCstation (gpacs1) with 32MB RAM memory, SUN video kit, and audio capability to the audio equipped 90 MHz Windows NT PC (pcpacs2) with 16MB RAM memory over 10 Mbps Ethernet network. gpacs1 and pcpacs2 machines ran the factory object. JDK 1.1.3 from SUN microsystems and Visibroker version 2.5 ORB from Visigenic software were installed on all the machines that are part of the framework. The client application ran on two different machines simulating different user scenarios. The two scenarios were:
Figure 6.1: Protocol stack for performance evaluation

- A client on the gpacs1 machine on which camera and microphone was created.
- A client on blackhorse on which there was no factory object, so no media object were local to the client machine.

6.1.2 WAN

In the WAN environment as shown in 6.2, a point-to-point stream between the SUN workstations at the University of Arizona and Bowman Gray School of Medicine (BG) was established. The client ran on the gpacs1 machine at the University of Arizona. The Internet was used to transport data.
6.2 Testing Method

In a distributed heterogeneous environment, each machine has its own clock. These clocks are not synchronized, so it is difficult to measure the delay seen by the media packets, which are captured on one machine and played on another machine. An algorithm was developed to synchronize the clocks by establishing a same
reference point in time. Fig. 6.3 shows this algorithm, which was used to calculate the delay. The algorithm was implemented in JAVA language. It is as follows:

1. First, the sender sends a packet with its local clock value (S_TS) as data.

2. When the receiver receives this packet, it records its current time (R_TS) and also the value of the sender time (S_TS); it sends the packet back to the sender right away.

3. The sender calculates the one-way delay when it gets the original packet back.

4. The sender sends this one-way delay value to the receiver.

5. The receiver calculates the value of its local time (M.S.TS) when the sender sent the first packet in step 1. \( M.S.TS = R.TS - o/wdelay \).

6. The source sends the media data with its local clock value (TS). The receiver calculates the delay as the time difference between the common reference point.

\[
\text{delay} = (\text{current.time} - M.S.TS) - (TS - S.TS).
\]

The algorithm described was implemented in the StreamEndPoints of the media devices. CamStreamEndPointImpl and MicStreamEndPointImpl implemented the sender side of the algorithm. DispStreamEndPointImpl and SpkStreamEndPointImpl implemented the receiver side of the algorithm. The StreamEndPoints executed steps 1 to 5 of the algorithm once before sending any media data. Step 6 was used to calculate end-to-end delay for each media packet. Time to perform the different
operations was measured at the client by using the local clock value. The experiments were repeated thirty times to get mean and standard deviation value. The next section shows the results of the performance experiment.

6.3 Results and Interpretation

This section gives the results of the performance test done in the LAN and the WAN environment described in section 6.1. *old fac. server* means that the server process with the factory object had been running for some time. Some clients did create objects using the factory object in the *old fac. server* process. *new fac. server* means that the server process with the factory object was just created. No object had been created yet. $\eta$ is the mean value for a particular operation and $\sigma$ is the standard deviation in the result. The following parameters were measured in the test.

1. The time it took the client to get a reference to media factory on the *gpacs1* machine.

2. The time it took the client to get a reference to remote factory on the *bones or pcpacs2* machine.

3. The time it took the create_object method to return.

4. The time it took the bind method to return when invoked on a device.

5. The time it took the *start, stop, and destroy* method invoked on the Stream-Control object to return.
Table 6.1: Time to get ref. to fac. on gpacs1 machine bound 1st

<table>
<thead>
<tr>
<th>Time to bind to gpacs1 fac. 1st (msec)</th>
<th>old fac. server</th>
<th>new fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td>client on gpacs1</td>
<td>1385</td>
<td>235.6</td>
</tr>
<tr>
<td>client on blackhorse</td>
<td>572</td>
<td>53</td>
</tr>
</tbody>
</table>

6. End-to-end delay (from media capture to media presentation).

7. End-to-end delay for audio in the LAN environment when sent alone and with video.

The round-trip network delay between machines in the LAN was 16 milliseconds (msec) and in the WAN was 196 msec. Table 6.1 shows the time it took to get the reference to the factory running on the gpacs1 machine in the LAN environment. The prototype applications created media objects at two different machines. Hence, the application needed to bind to two factories. The results in table 6.1 were obtained when the client application bound to the factory on the gpacs1 machine first in the LAN environment. It took twice the amount of time to the client running on the gpacs1 machine than to the client running on the blackhorse machine to bind to the factory on the gpacs1 machine. When the client and the factory were running on the same machine, machine computing power was shared between these two processes, so the performance was slow.
Table 6.2: Time to get ref. to fac. on gpacs1 machine bound 2nd

<table>
<thead>
<tr>
<th>Time to bind to gpacs1 fac. 2nd (msec)</th>
<th>old fac. server</th>
<th>new fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>σ</td>
</tr>
<tr>
<td>client on gpacs1</td>
<td>99.7</td>
<td>26.9</td>
</tr>
<tr>
<td>client on blackhorse</td>
<td>43.1</td>
<td>4.7</td>
</tr>
</tbody>
</table>

Table 6.3: Time to get ref. to fac. on pcpacs2 machine bound 1st

<table>
<thead>
<tr>
<th>Time to bind to pcpacs2 fac. 1st (msec)</th>
<th>old fac. server</th>
<th>new fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>σ</td>
</tr>
<tr>
<td>client on gpacs1</td>
<td>1359</td>
<td>443</td>
</tr>
<tr>
<td>client on blackhorse</td>
<td>572.8</td>
<td>17.4</td>
</tr>
</tbody>
</table>

Table 6.2 shows the results when the factory running on the gpacs1 machine was bound second; meaning that after binding to the pcpacs2 factory, the client application bound to the factory on the gpacs1 machine in the LAN. Again, when the client ran on the blackhorse machine, performance was better. It took less time to get the factory reference with old fac. server. This may have been due to the caching of information inside the ORB.

Table 6.3 shows how much time it took in the LAN to get the reference to the pcpacs2 factory, when the client bound to the pcpacs2 factory first. The results were similar to that of table 6.1.

Table 6.4 shows how much time it took in LAN to get the reference to the pcpacs2 factory, when the client bound to the pcpacs2 factory second, after binding to the
Table 6.4: Time to get ref. to fac. on pcpacs2 machine bound 2nd

<table>
<thead>
<tr>
<th>Time to bind to pcpacs2 fac. 2nd (msec)</th>
<th>old fac. server</th>
<th>new fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>σ</td>
</tr>
<tr>
<td>client on gpacs1</td>
<td>158</td>
<td>133</td>
</tr>
<tr>
<td>client on blackhorse</td>
<td>48.7</td>
<td>7.8</td>
</tr>
</tbody>
</table>

gpacs1 factory. Again, the results in table 6.4 were better than that in table 6.3. Comparing the performance of the two machines, it showed that it took the client running on the blackhorse machine 731 msec and 1012 msec respectively, to bind to the gpacs1 factory and the pcpacs2 factory first with the new factory server. When the client on blackhorse bound to both factories second, it took 224 msec to bind to the gpacs1 factory and 560 msec to bind to the pcpacs2 factory with new fac. server. The performance of gpacs1 machine was better, but it also had more memory and a faster processor than pcpacs2 so it was expected.

Table 6.5 gives the time delay values when the client created an object using the gpacs1 machine factory first. Table 6.6 gives the delay values when the client created an object using the gpacs1 machine factory after creating an object using the pcpacs2 factory. In both tables it can be seen that it took more time to create an object in the new server than in the old server. This could be due to caching. When the client was running on the gpacs1 machine, it took more time because multiple processes were running on the same machine.
Table 6.5: Time to create object on *gpacs1* machine bound 1st

<table>
<thead>
<tr>
<th>Time to create obj. on <em>gpacs1</em> 1st (msec)</th>
<th>old fac. server</th>
<th>new fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>σ</td>
</tr>
<tr>
<td>client on <em>gpacs1</em></td>
<td>211.8</td>
<td>108.7</td>
</tr>
<tr>
<td>client on <em>blackhorse</em></td>
<td>68</td>
<td>16.6</td>
</tr>
</tbody>
</table>

Table 6.6: Time to create object on *gpacs1* machine bound 2nd

<table>
<thead>
<tr>
<th>Time to create obj. on <em>gpacs1</em> 2nd (msec)</th>
<th>old fac. server</th>
<th>new fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>σ</td>
</tr>
<tr>
<td>client on <em>gpacs1</em></td>
<td>70</td>
<td>11.9</td>
</tr>
<tr>
<td>client on <em>blackhorse</em></td>
<td>38.8</td>
<td>11.5</td>
</tr>
</tbody>
</table>

Table 6.7 gives the time delay values when the client created an object using the *pcpacs2* machine factory first. Table 6.8 gives the delay values when the client created an object using the *pcpacs2* machine factory after creating an object using the *gpacs1* factory. In both tables it can be seen that it took more time to create an object in the new server than in the old server. This may have been due to caching. When a new factory server was started, it took less time to create an object on the *gpacs1* machine than on the *pcpacs2* machine.

Table 6.9 gives the values for the time taken to bind two devices. It took almost 7 seconds to bind two devices in *new fac. server* when the client ran on the *blackhorse* machine. With the old factory, time taken was less. Again this may have been due to caching of the intermediate objects involved. Table 6.10 gives the time it
Table 6.7: Time to create object on pcpacs2 machine bound 1st

<table>
<thead>
<tr>
<th>Time to create obj. on pcpacs2 1st (msec)</th>
<th>old fac. server</th>
<th>new fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>σ</td>
</tr>
<tr>
<td>client on gpacs1</td>
<td>283</td>
<td>303</td>
</tr>
<tr>
<td>client on blackhorse</td>
<td>66.3</td>
<td>9.8</td>
</tr>
</tbody>
</table>

Table 6.8: Time to create object on pcpacs2 machine bound 2nd

<table>
<thead>
<tr>
<th>Time to create obj. on pcpacs2 2nd (msec)</th>
<th>old fac. server</th>
<th>new fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>σ</td>
</tr>
<tr>
<td>client on gpacs1</td>
<td>68.5</td>
<td>23.1</td>
</tr>
<tr>
<td>client on blackhorse</td>
<td>66.3</td>
<td>9.8</td>
</tr>
</tbody>
</table>

Table 6.9: Time to bind 2 devices

<table>
<thead>
<tr>
<th>Time to bind 2 devices (msec)</th>
<th>new fac. server</th>
<th>old fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>σ</td>
</tr>
<tr>
<td>client on gpacs1</td>
<td>8530</td>
<td>1545</td>
</tr>
<tr>
<td>client on blackhorse</td>
<td>7232</td>
<td>539</td>
</tr>
</tbody>
</table>

Table 6.10: Time for destroy operation to return

<table>
<thead>
<tr>
<th>Time for destroy operation to return (msec)</th>
<th>new fac. server</th>
<th>old fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>σ</td>
</tr>
<tr>
<td>client on gpacs1</td>
<td>595</td>
<td>41</td>
</tr>
<tr>
<td>client on blackhorse</td>
<td>483</td>
<td>40</td>
</tr>
</tbody>
</table>
Table 6.11: Time for start operation to return

<table>
<thead>
<tr>
<th>Time for start operation to return (msec)</th>
<th>new fac. server</th>
<th>old fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>σ</td>
</tr>
<tr>
<td>client on <em>gpacs1</em></td>
<td>178</td>
<td>79</td>
</tr>
<tr>
<td>client on <em>blackhorse</em></td>
<td>115</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 6.12: Time for stop operation to return

<table>
<thead>
<tr>
<th>Time for stop operation to return (msec)</th>
<th>new fac. server</th>
<th>old fac. server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>η</td>
<td>σ</td>
</tr>
<tr>
<td>client on <em>gpacs1</em></td>
<td>2207</td>
<td>1141</td>
</tr>
<tr>
<td>client on <em>blackhorse</em></td>
<td>953</td>
<td>401</td>
</tr>
</tbody>
</table>

Table 6.13: Delay from media record to play in LAN

<table>
<thead>
<tr>
<th>media</th>
<th>delay alone (msec)</th>
<th>delay with audio &amp; video</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio</td>
<td>98</td>
<td>825</td>
</tr>
<tr>
<td>video</td>
<td>3255</td>
<td>7925</td>
</tr>
</tbody>
</table>
took from invocation to return of the\textit{destroy} operation. It took 500 msec for the\textit{destroy} operation to return. Whether the factory was old or new did not matter in this operation because no new objects were created. Table 6.11 shows how much time the\textit{start} operation took to finish. It took about 200 msec to 300 msec for the\textit{start} operation to return. As shown in table 6.12, it took 1 to 2 seconds for the\textit{stop} operation to return.\textit{Stop} operation has to stop all the running threads of StreamEndPoint. Based on the status of the threads, the\textit{stop} operation took more or less time. $\sigma$ for\textit{stop} was quite large because of the multiple threads.

Table 6.13 shows the end-to-end delay for audio and video in the LAN environment. When only audio stream was used, delay was 98 msec. This was quite good, but when both audio and video were sent, delay in audio increased to 825 msec. This was not adequate for real-time conversation. For playback, playback starting point can be adjusted to accommodate the delay [83]. The video window size was 210 $\times$ 160 pixels. The delay in the video was a lot larger than the audio because the video frames are about 7000 bytes compared to 534 bytes for the audio frames. The delay in video was quite noticeable. Delay in video frame was 3 seconds in LAN and 8 seconds in WAN. Performance in WAN was traffic dependent, so it fluctuated widely. When both audio and video were streamed, the threads involved with the audio stream were given the highest priority, and the threads involved with the video stream were given normal priority to prevent breakage in voice. Initially, UDP was used to transport the media data as well as the credit packets. The flow protocol
Table 6.14: Performance results for WAN

<table>
<thead>
<tr>
<th>operation</th>
<th>time delay (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bind to factory in bones machine</td>
<td>3755</td>
</tr>
<tr>
<td>create media object in bones machine</td>
<td>320</td>
</tr>
<tr>
<td>bind devices</td>
<td>9716</td>
</tr>
<tr>
<td>start operation returns</td>
<td>368</td>
</tr>
<tr>
<td>stop operation returns</td>
<td>1580</td>
</tr>
<tr>
<td>remove operation returns</td>
<td>1363</td>
</tr>
</tbody>
</table>

algorithm dealt with the loss in the media packets, but if credit the packets were lost, eventually the source had no credits to send and it stalled, so currently TCP is used to transport credit values.

Table 6.14 gives the test results in the WAN environment. In this case, the client was running on the gpacs1 machine at the University of Arizona (UA). The factory at UA was local and the client bound to it first. The media object at UA factory was created first. The WAN results are for old fac. server so they are compared with the old factory server results in the LAN environment. It took almost 3.7 seconds to get the reference to the factory at BG. In LAN, it was between 150 ms to 1.7 seconds. Time to create media object at BG was approximately 320 ms. It took almost 10 seconds to bind devices and get the reference to a stream. This was a considerable delay because it involved several round trips between UA and BG for negotiation. start, stop, and destroy operation took 368 ms, 1580 ms, and 1363 ms respectively. Results for stop operation varied significantly. WAN results were dependent on the Internet traffic patterns.
CHAPTER 7

Summary and Conclusion

In this dissertation, an architecture for CORBA-based distributed multimedia middleware framework was created, investigated, and implemented using JAVA. The framework adds distributed multimedia streaming capability to CORBA. Currently, support for audio and video stream is available but it is being worked on to extend it to support other media, such as annotation data. The framework showed multimedia streams in a heterogeneous environment using the SUN Solaris and the Windows NT machines. The same code of DMMF (excluding hardware interface to media devices) was run on both machines. Using DMMF, it was very easy to create distributed object based multimedia system as shown in the Result section. DMMF will enable rapid development of distributed multimedia applications, such as telemedicine and distance learning systems. DMMF architecture is very modular to support new devices and to accommodate change in the current devices. Our use of JAVA and incorporation of ORB by Netscape in its browser software will enable the downloading of an applet to do multimedia playback or real-time communication using a browser when JAVA multimedia APIs are available. It took 7 seconds in the LAN and 16 seconds in the WAN environment to establish a media stream, but this was only in the initial phase. Therefore, it should not be much of a problem considering that
it dynamically creates and binds media objects. Due to JAVA/C interface, we were only able to get 2.5 f/s of MJPEG video in the LAN environment. The end-to-end delay in audio was good for real-time conversation in the LAN environment when the video stream was not present. The experiments used shared LAN and the Internet. Use of networking technology that guarantees QoS will change the performance results dramatically. Valuable insight into how to create CORBA based framework for multimedia was gained in this work.

7.1 Current Constraints

The framework showed interoperability between different machines. In the experiments, the SUN machine was more powerful than the NT machine, so it was hard to differentiate ORB performance on the two machines. If both machines are of the same computing capacity, then a better observation can be made. The machines used had very low processing power for multimedia processing. LAN network capacity is only 10 Mbits/s, which limits the size of video frames and increases the delay for sending media. The framework has capability to specify and negotiate quality of service from the network, but such a network is not available in the lab yet. With ATM network, which provides quality of service support, the framework can provide appropriate performance to the application. The performance in WAN depended on the Internet traffic. At the time when data traffic was heavy, it was not possible to register the remote factory object. If a factory object cannot be registered in naming
service, the application cannot be run. JAVA to C interface was bottleneck in capturing the JPEG frames. In DMMF, we kept implementation in JAVA as much as possible. If there is a performance problem, then some part may have to be moved to C language.

7.2 Future Work

The next step should be to measure the performance of the framework on a high-bandwidth network, such as ATM and to develop devices that support synchronization between various media. To pass the multimedia application QoS requirements to the ATM network, interface should be created from DMMF to ATM switches. DMMF should be used to create a framework for a computer supported collaborative workspace (CSCW) that applications such as RCD and JIVE can use. Point-to-point streams should be extended to point-to-multipoint stream. Devices should be created to retrieve and store multimedia in a database. This will enable capture and playback of multimedia at a later time. For health care, media devices should be created for capturing and displaying radiology modalities. The DMMF media devices can be created at each equipment for capturing images, video, or audio. DMMF framework should be integrated with the RCD and the JIVE distributed multimedia applications.
Appendix A

Dist. Multimedia Middleware Framework IDL

module AVrfp
{
  struct QoS
  {
    string QoSType;
    CosPropertyService::Properties QoSParams;
  }
  typedef sequence<QoS> streamQoS;
  typedef sequence<string> flowSpec;
  typedef sequence<string> protocolSpec;
  typedef sequence<octet> key;
  struct SFPStatus
  {
    boolean isFormatted;
    boolean isSpecialFormat;
    boolean seqNums;
    boolean timestamps;
    boolean sourceIndicators;
  }
  enum flowState
  {
    stopped, started, dead
  }
  enum dirType
  {
    dir_in, dir_out
  }
  struct flowStatus
  {
    string flowName;
    dirType directionality;
    flowState status;
    SFPStatus theFormat;
    QoS theQoS;
  }
}
exception notSupported();
exception PropertyException();
exception FPError { string flow_name; };
exception streamOpFailed
{
    string reason;
};
exception streamOpDenied
{
    string reason;
};
exception noSuchFlow();
exception QoSRequestFailed
{
    string reason;
};
interface Basic.StreamCtrl : CosPropertyService::PropertySet
{
    void stop( in flowSpec the_spec ) raises (noSuchFlow);
    void start( in flowSpec the_spec ) raises (noSuchFlow);
    void destroy( in flowSpec the_spec ) raises (noSuchFlow);
    boolean modify_QoS( inout streamQoS new_QoS, in flowSpec
                        the_spec) raises (noSuchFlow, QoSRequestFailed);
};
interface SpkDevice;
interface MicDevice;
interface AudioStrCtrl;
interface MicStreamEndPoint;
interface SpkStreamEndPoint;
interface SpkVDev;
interface MicVDev;
interface CamDevice;
interface DispDevice;
interface VideoStrCtrl;
interface CamStreamEndPoint;
interface DispStreamEndPoint;
interface CamVDev;
interface DispVDev;
interface AudioStrCtrl : Basic_StreamCtrl
{
boolean bind_devs( in MicDevice a_party, in SpkDevice b_party, 
inout streamQoS the_qos, in flowSpec the_flows) raises 
(streamOpFailed, noSuchFlow, QoSRequestFailed);
boolean bind( in MicStreamEndPoint a_party, in 
SpkStreamEndPoint b_party, inout streamQoS the_qos, in 
flowSpec the_flows) raises (streamOpFailed, noSuchFlow, 
QoSRequestFailed);
void unbind_party( in SpkStreamEndPoint the_ep, in flowSpec 
the_spec ) raises (streamOpFailed, noSuchFlow );
void unbind() raises ( streamOpFailed );
};
interface MicStreamEndPoint : CosPropertyService::PropertySet 
{
  void stop( in flowSpec the_spec ) raises ( noSuchFlow );
  void start( in flowSpec the_spec ) raises ( noSuchFlow );
  void destroy( in flowSpec the_spec ) raises ( noSuchFlow );
  boolean connect( in SpkStreamEndPoint responder, inout 
  streamQoS qos_spec, in flowSpec the_spec ) raises 
  (noSuchFlow, QoSRequestFailed, streamOpFailed );
  boolean modify_QoS( inout streamQoS new_qos, in flowSpec 
  the_flows ) raises ( noSuchFlow, QoSRequestFailed );
};
interface SpkStreamEndPoint : CosPropertyService::PropertySet 
{
  void stop( in flowSpec the_spec ) raises ( noSuchFlow );
  void start( in flowSpec the_spec ) raises ( noSuchFlow );
  void destroy( in flowSpec the_spec ) raises ( noSuchFlow );
  boolean request_connection( in MicStreamEndPoint initiator, in 
  boolean is_mcast, inout streamQoS qos, inout flowSpec 
  the_spec) raises (streamOpDenied, noSuchFlow, 
  QoSRequestFailed, FPError );
  boolean modify_QoS( inout streamQoS new_qos, in flowSpec 
  the_flows ) raises ( noSuchFlow, QoSRequestFailed );
};
interface SpkVDev : CosPropertyService::PropertySet 
{
  boolean set_peer( in AudioStrCtrl the_ctrl, in MicVDev 
  the_peer_dev, inout streamQoS the_qos, in flowSpec the_spec 
  ) raises ( noSuchFlow, QoSRequestFailed, streamOpFailed );
  boolean modify_QoS( inout streamQoS the_qos, in flowSpec 
  the_spec ) raises ( noSuchFlow, QoSRequestFailed );
interface MicVDev : CosPropertyService::PropertySet
{
    boolean set_peer( in AudioStrCtrl the_ctrl, in SpkVDev the_peer_dev, inout streamQoS the_qos, in flowSpec the_spec ) raises ( noSuchFlow, QoSRequestFailed, streamOpFailed );
    boolean modify_QoS( inout streamQoS the_qos, in flowSpec the_spec ) raises ( noSuchFlow, QoSRequestFailed );
};

interface SpkDevice : CosPropertyService::PropertySet, CosLifeCycle::LifeCycleObject
{
    SpkStreamEndPoint create_B( in AudioStrCtrl the_requester, out SpkVDev the_vdev, inout streamQoS the_qos, out boolean met_QoS, inout string named_vdev, in flowSpec the_spec ) raises ( streamOpFailed, streamOpDenied, notSupported, QoSRequestFailed, noSuchFlow );
    AudioStrCtrl bind( in MicDevice peer_device, inout streamQoS the_qos, out boolean is_met, in flowSpec the_spec ) raises ( streamOpFailed, noSuchFlow, QoSRequestFailed );
    void destroy( in SpkStreamEndPoint the_ep, in string vdev_name ) raises ( notSupported );
};

interface MicDevice : CosPropertyService::PropertySet, CosLifeCycle::LifeCycleObject
{
    MicStreamEndPoint create_A( in AudioStrCtrl the_requester, out MicVDev the_vdev, inout streamQoS the_qos, out boolean met_qos, inout string named_vdev, in flowSpec the_spec ) raises ( streamOpFailed, streamOpDenied, notSupported, QoSRequestFailed, noSuchFlow );
    AudioStrCtrl bind( in SpkDevice peer_device, inout streamQoS the_qos, out boolean is_met, in flowSpec the_spec ) raises ( streamOpFailed, noSuchFlow, QoSRequestFailed );
    void destroy( in MicStreamEndPoint the_ep, in string vdev_name ) raises ( notSupported );
};

interface VideoStrCtrl : Basic_StreamCtrl
{
    boolean bind_devs( in CamDevice a_party, in DispDevice b_party, inout streamQoS the_qos, in flowSpec the_flows)
raises (streamOpFailed, noSuchFlow, QoSRequestFailed);
boolean bind( in CamStreamEndPoint a_party, in
DispStreamEndPoint b_party, inout streamQoS the_qos, in
flowSpec the_flows) raises (streamOpFailed, noSuchFlow,
QoSRequestFailed);
void unbind_party( in DispStreamEndPoint the_ep, in flowSpec
the_spec ) raises (streamOpFailed, noSuchFlow);
void unbind() raises (streamOpFailed);
};
interface CamStreamEndPoint : CosPropertyService::PropertySet
{
void stop( in flowSpec the_spec ) raises (noSuchFlow);
void start( in flowSpec the_spec ) raises (noSuchFlow);
void destroy( in flowSpec the_spec ) raises (noSuchFlow);
boolean connect( in DispStreamEndPoint responder, inout
streamQoS qos_spec, in flowSpec the_spec ) raises
(noSuchFlow, QoSRequestFailed, streamOpFailed);
boolean modify_QoS( inout streamQoS new_qos, in flowSpec
the_flows ) raises (noSuchFlow, QoSRequestFailed);
};
interface DispStreamEndPoint : CosPropertyService::PropertySet
{
void stop( in flowSpec the_spec ) raises (noSuchFlow);
void start( in flowSpec the_spec ) raises (noSuchFlow);
void destroy( in flowSpec the_spec ) raises (noSuchFlow);
boolean request_connection( in CamStreamEndPoint initiator, in
boolean is_mcast, inout streamQoS qos, inout flowSpec
the_spec) raises (streamOpDenied, noSuchFlow,
QoSRequestFailed, FPError);
boolean modify_QoS( inout streamQoS new_qos, in flowSpec
the_flows ) raise (noSuchFlow, QoSRequestFailed);
};
interface DispVDev : CosPropertyService::PropertySet
{
boolean set_peer( in VideoStrCtrl the_ctrl, in CamVDev
the_peer_dev, inout streamQoS the_qos, in flowSpec the_spec
) raises (noSuchFlow, QoSRequestFailed, streamOpFailed);
boolean modify_QoS( inout streamQoS the_qos, in flowSpec
the_spec ) raises (noSuchFlow, QoSRequestFailed);
};
interface CamVDev : CosPropertyService::PropertySet
{  
  boolean set_peer( in VideoStrCtrl the_ctrl, in DispVDev  
                  the_peer_dev, inout streamQoS the_qos, in flowSpec the_spec  
                  ) raises ( noSuchFlow, QoSRequestFailed, streamOpFailed );  
  boolean modify_QoS( inout streamQoS the_qos, in flowSpec  
                      the_spec ) raises ( noSuchFlow, QoSRequestFailed );  
};
interface DispDevice : CosPropertyService::PropertySet,  
CosLifeCycle::LifeCycleObject  
{
  DispStreamEndPoint create_B( in VideoStrCtrl the_requester,  
                              out DispVDev the_vdev, inout streamQoS the_qos, out boolean  
                              met_QoS, inout string named_vdev, in flowSpec the_spec  
                              ) raises ( streamOpFailed, streamOpDenied, notSupported,  
                              QoSRequestFailed, noSuchFlow );  
  VideoStrCtrl bind( in CamDevice peer_device, inout streamQoS  
                   the_qos, out boolean is_met, in flowSpec the_spec ) raises (  
                   streamOpFailed, noSuchFlow, QoSRequestFailed );  
  void destroy( in DispStreamEndPoint the_ep, in string  
                vdev_name ) raises ( notSupported );  
};
interface CamDevice : CosPropertyService::PropertySet,  
CosLifeCycle::LifeCycleObject  
{
  CamStreamEndPoint create_A( in VideoStrCtrl the_requester, out  
                            CamVDev the_vdev, inout streamQoS the_qos, out boolean  
                            met_qos, inout string named_vdev, in flowSpec the_spec )  
                            raises ( streamOpFailed, streamOpDenied, notSupported,  
                            QoSRequestFailed, noSuchFlow );  
  VideoStrCtrl bind( in DispDevice peer_device, inout streamQoS  
                   the qos, out boolean is_met, in flowSpec the_spec ) raises (  
                   streamOpFailed, noSuchFlow, QoSRequestFailed );  
  void destroy( in CamStreamEndPoint the_ep, in string vdev_name  
                ) raises ( notSupported );  
};
REFERENCES


