

# University of Stuttgart

## Faculty of Computer Science

**Program of Study:** Information Technology

**Examiner:** Prof. Dr. K. Rothermel

**Supervisor:** Dipl. Inform. Detlef Bosau

**Begin:** December 1, 2001

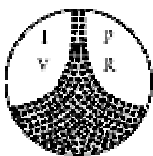
**End:** May 31, 2002

**CR-Classification:** C.2.1, C.2.2, C.2.4

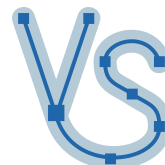
**Master Thesis Nr. 1978**

### **Development of Network Service Infrastructure For Transcoding Multimedia Streams**

Antony Pranata



Institute of Parallel and  
Distributed High-Performance Systems  
Distributed Systems Department  
University of Stuttgart



# **Abstract**

In the COMCAR project, distributed multimedia applications can be run on mobile terminals at different locations, e.g. a mobile terminal may have access to the Internet using 3<sup>rd</sup> generation mobile networks, or the mobile terminal may have access to a campus LAN using WaveLAN or HiperLAN.

Adaptive applications, in order to be run, can be adapted depending on the mobile terminal's location, its facilities and the network connection in use. One possibility of adaptation is the use of transcoding services for media stream when there is no common media format supported both, at the source and the sink of the stream, or none of the supported format matches the limitations of the available network connection. For example, a media stream is transcoded from a highly resource consuming format, Cinepak, to format which requires less resources, H.263.

Based upon Java, Jini and JMF, in this thesis a service infrastructure for transcoding services is to be designed and prototypically implemented. The service infrastructure is to provide 1) mechanisms to find and select transcoding services, which are appropriate for a given transcoding problem (service brokering) 2) mechanisms to construct the service chain from the source through one or more transcoding services to the sink according the definitions provided by the application.

# Acknowledgement

I would like to thank to Prof. Rothermel for giving me a change to do this thesis in Distributed Systems department and Ernoe Kovacs who has introduced this topic.

I would also like to thank to Detlef Bosau and Klaus Roehrle for assisting me during the writing of this thesis. Their feedback has helped me very much to improve the quality of this thesis.

Finally I would like to thank to my mother and all my friends here, especially Novi who has supported me, Larry, Irwan, Ferry, Inti, and all Indonesian students in Stuttgart.

# Table of Contents

<b>Abstract .....</b>	<b>ii</b>
<b>Acknowledgement.....</b>	<b>iii</b>
<b>Table of Contents.....</b>	<b>iv</b>
<b>List of Figures .....</b>	<b>viii</b>
<b>List of Tables .....</b>	<b>x</b>
<b>Chapter 1 Introduction .....</b>	<b>1</b>
1.1 Motivation .....	1
1.2 The COMCAR Project .....	2
1.3 Organization of the Thesis.....	3
<b>Chapter 2 Distributed Multimedia Systems.....</b>	<b>5</b>
2.1 Terminology .....	5
2.2 Applications of Distributed Multimedia Systems .....	6
2.2.1 Conversational Applications.....	6
2.2.2 Messaging Applications .....	6
2.2.3 Retrieval applications .....	7
2.2.4 Distribution Applications .....	7
2.2.5 Applications in This Thesis .....	7
2.3 Multimedia Compression .....	7
2.4 Networks.....	8
2.4.1 LAN .....	9
2.4.2 Wireless Network .....	9
2.4.3 Comparison.....	9
2.5 Network Protocols .....	10
2.5.1 Internet Protocol (IP) .....	11
2.5.2 Transmission Control Protocol (TCP).....	11

2.5.3	User Datagram Protocol (UDP).....	11
2.5.4	Real-time Transport Protocol (RTP) .....	11
2.5.5	Real-time Transport Streaming Protocol (RTSP).....	11
2.5.6	HyperText Transport Protocol (HTTP) .....	12
2.6	Quality of Service (QoS) .....	12
2.7	Summary.....	12
<b>Chapter 3 Java-based Middleware .....</b>		<b>13</b>
3.1	Introduction to Middleware .....	13
3.1.1	Distribution Transparency .....	14
3.1.2	Middleware Models.....	14
3.1.3	Middleware Services .....	15
3.2	Java RMI.....	15
3.2.1	RPC Failure Semantics.....	16
3.3	Jini .....	16
3.3.1	Architecture of Jini .....	17
3.4	Summary.....	18
<b>Chapter 4 Transcoding Infrastructure.....</b>		<b>19</b>
4.1	Example Scenario .....	19
4.2	Solution.....	20
4.2.1	Solving Heterogeneity and Mobility Problems .....	21
4.2.2	Transcoders to Solve Heterogeneity and Mobility Problems .....	22
4.2.3	Lookup Service and Service Broker.....	24
4.3	Requirements.....	25
4.3.1	Server.....	25
4.3.2	Transcoder .....	25
4.3.3	Client .....	25
4.3.4	Service Broker .....	26
4.3.5	Lookup Service.....	26
4.4	Summary.....	26
<b>Chapter 5 Architecture Design.....</b>		<b>27</b>
5.1	Service Brokering .....	27
5.1.1	Finding Source Format and Destination Format .....	27
5.1.2	Finding Transcoding Format .....	28
5.1.3	Assigning Priority to Transcoding Format .....	29

5.1.4	Cascade Filtering .....	31
5.1.5	QoS Parameters .....	33
5.1.6	Flow Chart .....	33
5.2	Service Chaining.....	35
5.2.1	Finding Lookup Service .....	35
5.2.2	Service Registration.....	36
5.2.3	Requesting Transcoder Service .....	39
5.2.4	Server-initiated Request .....	41
5.2.5	Streams Garbage.....	44
5.2.6	Transcoder Handover .....	45
5.3	Transcoder Configuration.....	46
5.4	N-Level Transcoding.....	47
5.4.1	Constructing Directed Graph.....	48
5.4.2	Optimizing Directed Graph .....	49
5.4.3	Adding Client and Server .....	50
5.4.4	Constructing Weighted Graph.....	52
5.4.5	Shortest Path Algorithm .....	53
5.4.6	Flow Chart .....	54
5.4.7	Implementation Problems.....	56
5.5	Summary.....	56
<b>Chapter 6 Implementation.....</b>		<b>57</b>
6.1	Platform .....	57
6.1.1	Programming Language .....	57
6.1.2	Communication Protocol.....	58
6.1.3	Service Discovery Protocol .....	58
6.1.4	Architecture .....	59
6.2	Transcoder .....	59
6.2.1	TranscoderInterface and TranscoderImpl.....	60
6.2.2	TranscoderPlayerInterface and TranscoderPlayerImpl .....	62
6.2.3	TranscoderDaemon.....	64
6.3	Service Broker .....	66
6.3.1	ServiceBrokerDaemon.....	66
6.3.2	ServiceBrokerInterface and ServiceBrokerImpl .....	67
6.3.3	ClientPreferences.....	67
6.3.4	SystemInfo.....	67
6.4	Client .....	68

6.5	Establishing Connection .....	69
6.6	Summary.....	70
<b>Chapter 7 Integration and Testing.....</b>		<b>71</b>
7.1	Integration.....	71
7.1.1	Hardware .....	71
7.1.2	Software.....	71
7.1.3	Service .....	71
7.1.4	Installation .....	72
7.2	Testing .....	75
7.3	Summary.....	76
<b>Chapter 8 Related Works .....</b>		<b>77</b>
8.1	Proxy-based Transcoding .....	77
8.1.1	Architecture .....	77
8.1.2	Contribution of This Thesis .....	78
8.2	KISS Project .....	79
8.2.1	Contribution of This Thesis .....	80
8.3	ICEBERG Project.....	80
8.3.1	Contribution of This Thesis .....	81
8.4	Summary.....	82
<b>Chapter 9 Summary and Future Works .....</b>		<b>83</b>
9.1	Summary.....	83
9.2	Future Works .....	83
<b>Appendix A Java Media Framework.....</b>		<b>85</b>
<b>Appendix B Class Hierarchy .....</b>		<b>89</b>
	Client .....	89
	Service Broker .....	89
	Transcoder .....	90
<b>References.....</b>		<b>91</b>

## List of Figures

Figure 1-1 <i>The COMCAR system (courtesy of the COMCAR project).</i> .....	3
Figure 2-1 <i>Architecture of distributed multimedia systems (courtesy of Coulouris et. al., 2001).</i> .....	6
Figure 2-2 <i>TCP/IP and OSI reference model.</i> .....	10
Figure 3-1 <i>Architecture of a distributed system as middleware (courtesy of Rothermel, 2001).</i> .....	14
Figure 3-2 <i>Principle of RPC between a client and server program (courtesy of Tanenbaum and van Steen, 2002).</i> .....	16
Figure 3-3 <i>The Jini architecture (courtesy of Sun Microsystems, 2001).</i> .....	16
Figure 3-4 <i>A flow diagram of Jini technology (courtesy of Sun Microsystems, 2001).</i> .....	17
Figure 4-1 <i>Heterogeneity of client devices and network connections.</i> .....	20
Figure 4-2 <i>The architecture of a simple transcoding infrastructure.</i> .....	22
Figure 4-3 <i>Transcoders for mobile clients.</i> .....	23
Figure 4-4 <i>Another advantage of the transcoding infrastructure.</i> .....	24
Figure 4-5 <i>Architecture of the network service infrastructure for transcoding multimedia streams.</i> .....	25
Figure 5-1 <i>How to find the transcoder?</i> .....	28
Figure 5-2 <i>An example of a list of transcoding formats and destination formats.</i> .....	29
Figure 5-3 <i>Cascade filtering to the list of transcoders.</i> .....	32
Figure 5-4 <i>Flowchart of service brokering.</i> .....	35
Figure 5-5 <i>Registration of service broker using multicast request.</i> .....	37
Figure 5-6 <i>Registration of service broker using multicast announcement and unicast discovery.</i> .....	37
Figure 5-7 <i>A time diagram of a service ID.</i> .....	38
Figure 5-8 <i>Stream and control flow of the client-initiated request.</i> .....	39
Figure 5-9 <i>Time diagram of client-initiated request for multimedia streams.</i> .....	41
Figure 5-10 <i>Architecture of the server-initiated service chaining.</i> .....	42
Figure 5-11 <i>Time diagram of server-initiated request for multimedia streams.</i> ...	44
Figure 5-12 <i>Time diagram of elimination of streams garbage.</i> .....	45
Figure 5-13 <i>Example of flat transcoding infrastructure.</i> .....	46



Figure 5-14 <i>Example of hierarchical transcoding infrastructure.</i>	47
Figure 5-15 <i>Service brokering and service chaining in N-level transcoding.</i>	48
Figure 5-16 <i>Example of four transcoders in a network.</i>	49
Figure 5-17 <i>Directed graph for the transcoders.</i>	49
Figure 5-18 <i>Directed graph after optimization.</i>	50
Figure 5-19 <i>Directed graph for the transcoders, the server and the client.</i>	51
Figure 5-20 <i>Another directed graph for the transcoders, the server and the client.</i>	51
Figure 5-21 <i>Two possible chains from the server to the client.</i>	52
Figure 5-22 <i>Weighted and directed graph for the transcoders.</i>	53
Figure 5-23 <i>Finding the shortest path using Dijkstra algorithm.</i>	54
Figure 5-24 <i>Flow chart of service brokering and service chaining of N-level transcoding.</i>	55
Figure 6-1 <i>Architecture of transcoder, client and service broker from Java perspective.</i>	59
Figure 6-2 <i>Architecture of the transcoder.</i>	60
Figure 6-3 <i>Stream flow in the transcoder player.</i>	63
Figure 6-4 <i>Attributes of a transcoder.</i>	65
Figure 6-5 <i>Not all source formats can be transcoded to the destination formats.</i>	66
Figure 6-6 <i>Architecture of the service broker.</i>	67
Figure 6-7 <i>Stream flow in the client.</i>	69
Figure 6-8 <i>Protocols to establish a connection from Java perspective.</i>	70
Figure 7-1 <i>Installing the components of the transcoding service into computers.</i>	72
Figure 7-2 <i>The configuration of testing purpose.</i>	75
Figure 8-1 <i>Basic architecture of proxy-based transcoding (courtesy of A. Fox, et.al., 1996).</i>	77
Figure 8-2 <i>Architecture of network infrastructure of KISS project (courtesy of K. Jonas, et. al., 1998).</i>	79
Figure 8-3 <i>One scenario of APC in the ICEBERG project (courtesy of Mao and Ratz, 2000).</i>	81

## List of Tables

Table 2-1 <i>Bandwidth requirements of uncompressed multimedia streams.</i> .....	7
Table 2-2 <i>Comparison of several audio compression algorithms.</i> .....	8
Table 2-3 <i>Comparison of several video compression algorithms.</i> .....	8
Table 2-4 <i>Comparison of network bandwidth.</i> .....	10
Table 4-1 <i>Different capabilities of some typical computer devices.</i> .....	20
Table 5-1 <i>Priority table of audio formats in the service broker.</i> .....	30
Table 5-2 <i>Priority table of video formats in the service broker.</i> .....	30
Table 5-3 <i>Priority table of audio and video formats in the service broker.</i> .....	31
Table 5-4 <i>Finding the shortest path using Dijkstra algorithm.</i> .....	54
Table 7-1 <i>List of transcoders in the integration part.</i> .....	74

# Chapter 1

## Introduction

### 1.1 Motivation

The explosive growth of the Internet and mobile computing introduces two main problems in distributed multimedia applications. The first problem is heterogeneity of client devices and their network connections. The client devices may vary from desktop PCs, notebook computers, PDAs to mobile phones, which their capabilities also vary along many axes, including screen size, color depth and processing power [8]. Furthermore, they may connect to the Internet via different networks, such as wired LAN, wireless LAN or wireless WAN.

The second problem is mobility of clients. The clients may be moving while they are accessing multimedia streams. It may cause a problem because the network connections may change from time to time, ranging from a very good network to a congested network.

The two problems described above make it difficult for a multimedia server to provide a streaming service which is appropriate for every client in every situation. A solution to the problems above, which is presented in this thesis, is by converting multimedia streams to the appropriate format on-the-fly. The converting process is also known as *transcoding*, which means converting multimedia streams from one format to another format. The transcoding process itself needs a new server, called *transcoder*.

The purpose of this thesis is to develop a prototype of a network service infrastructure for transcoding multimedia streams. The prototype allows a client on a network to request a multimedia stream and the transcoder transcode it to the appropriate format for the client. The service infrastructure should be able to find the appropriate transcoder and build chain from the server to the client.

The prototype proposed in this thesis is designed for a Campus LAN only. It may need some modifications to be applied for larger areas, such as the Internet. Scalability is an important issue to be considered when applying the infrastructure to the Internet because the number of users may be in order of millions.

There are a lot of areas which can use this transcoding service infrastructure. I will give four examples of scenarios here. The first scenario is

for exhibitions, such as CeBit or COMDEX. They may use the infrastructure to broadcast a video news or any multimedia information about the exhibition. The visitors, which use various types of devices, are able to watch the video because transcoders transcode the original stream to the appropriate format for them.

The second scenario is for sport events, such as Olympic Games or Football World Cup, which takes place in a city. The committee of the sport event may want to broadcast the latest news for reporters or visitors. They may install transcoders in the stadiums or buildings where the video can be watched using client devices, such as notebooks or PDAs.

The third scenario is for museums which offers multimedia streams to explain the content or history of their museums. Currently, many museums provide a tape-player or a head-set so that the visitors are able to listen or watch the history of some other stuffs there. Using the transcoder infrastructure, the visitors are now be able to use their mobile devices to watch or listen. Furthermore, they may be walking around while listening to the explanation.

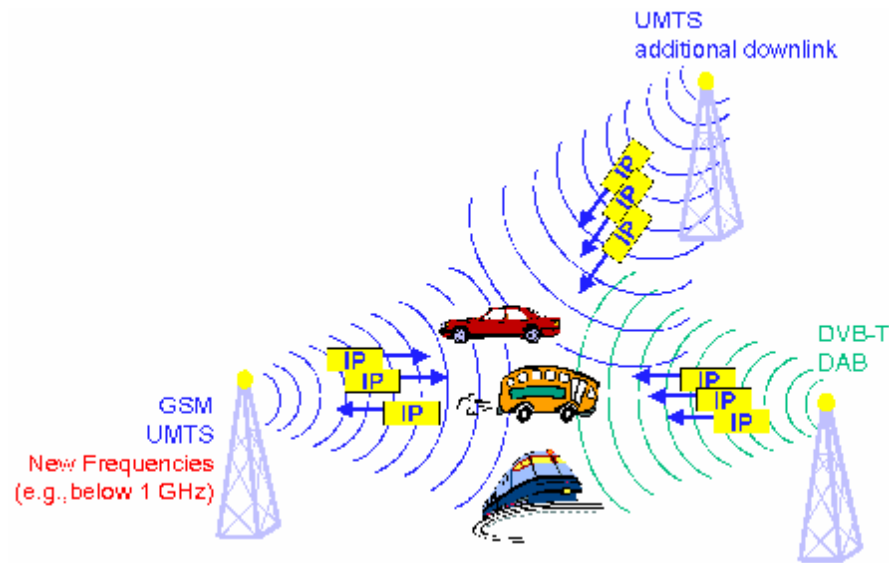
The last scenario is the most ambitious one and it is one goal of the COMCAR project. The end users on the cars or trains should be able to access multimedia streams while they are moving. For example, the transcoder should be able to transcode the Digital Video Broadcast (DVB) or Digital Audio Broadcast (DAB) depends on the network conditions.

## 1.2 The COMCAR Project

This thesis is part of the COMCAR project. The COMCAR project itself is a part of UMTSplus, a new system concept sponsored by the Germany Ministry for Education and Research (BMB+F), which aims at *Universality and Mobility in Telecommunication Networks and Systems*. Partners in COMCAR are DaimlerChrysler AG, Research & Technology Eurolab Deutschland GmbH, Sony International (Europe) GmbH, and T-Nova Innovationgesellschaft mbH [5].

The COMCAR project targets at the conception and prototypical realization of an innovative mobile communication network, which shall satisfy the increasing demand for IP-based multimedia and telematics services especially in cars and railways. The main focus in COMCAR is on asymmetrical and interactive IP-based services (see Figure 1-1). Existing and upcoming elaborated radio technologies and infrastructures such as GSM, UMTS, DVB-T, and DAB shall be used and optimized to bring asymmetric high-quality IP-based services to vehicles like cars and trains.

COMCAR will provide flexible communication environment in which QoS parameter will change on a wide scale. COMCAR will examine how this scenario might influence emerging Internet technologies for integrating QoS in IP networks. Furthermore, COMCAR will also develop mobile middleware technologies that allow adaptive multimedia applications to react user-tailored to the changing user situation.



**Figure 1-1** *The COMCAR system (courtesy of the COMCAR project).*

The result of this thesis might be used in the COMCAR project especially in the adaptation application [20]. Currently the adaptation is based on different streams available on the source. The introduction of transcoding infrastructure will make the adaptation algorithm more flexible because there are more different choices of streams which can be selected.

### 1.3 Organization of the Thesis

In general, this thesis consists of two main parts, concept part and design part. The concept part, which contains Chapter 1 to 3, discusses the introduction to distributed multimedia systems. The design part, which contains Chapter 4 to 9, discusses the requirements, design, implementation and evaluation of the transcoding infrastructure.

Chapter 1 (this chapter), *Introduction*, discusses the background of this thesis and gives some application scenarios in which the result of this thesis may be applied.

Chapter 2, *Distributed Multimedia Systems*, discusses some important aspects in distributed multimedia systems which have relations with this thesis, including the terminology and applications of distributed multimedia systems, multimedia compression, networking and network protocols for distributed multimedia systems and finally Quality of Service.

Chapter 3, *Java-based Middleware*, discusses middleware of distributed systems, but it focuses on Java-based middleware. There are two main discussions in this chapter, RMI and Jini.

Chapter 4, *Transcoding Infrastructure*, discusses the background why we need transcoding infrastructure in distributed multimedia systems. It also covers the requirements of a common transcoding infrastructure.

Chapter 5, *Architecture Design*, discusses the architecture design of the transcoding infrastructure implemented in this thesis. There two main discussions, service brokering and service chaining.

Chapter 6, *Implementation*, discusses the implementation of the transcoding infrastructure in Java platform. It explains how each component of the transcoding infrastructure can be implemented in Java technology and classes.

Chapter 7, *Integration and Evaluation*, discusses how the components of the infrastructure can be integrated in real-world. This chapter also gives an evaluation of the tests performed in this real infrastructure.

Chapter 8, *Related Works*, discusses some related works which also deal with transcoding infrastructure. This chapter also discusses the contribution of this thesis to the community.

Chapter 9, *Summary and Future Works*, gives summary of this thesis and some outlooks in the future.

## Chapter 2

# Distributed Multimedia Systems

This chapter gives an overview of some aspects in distributed multimedia systems which have relations with this thesis. It starts with some important terminologies and application scenarios in distributed multimedia systems. After that, it explains multimedia compression which is very important in distributed multimedia systems and used very intensive in this thesis. The next part is networks and networking protocols which can be used by distributed multimedia systems. At the end of this chapter, there is a discussion about Quality of Service (QoS).

### 2.1 Terminology

Before discussing distributed multimedia systems, I first discuss multimedia systems and multimedia communications. Lu [9] defines *multimedia systems* as any systems capable of handling *discrete media* as well as at least one type of *continuous media* in digital form.

Discrete media (also called *time-independent media* or *static media*) are media that do not have a time dimension, their meanings do not depend on the presentation time. Discrete media include alphanumeric data and graphics. On the other hand, continuous media (also called *time-dependent media* or *dynamic media*) have a time dimension, their meanings depend on the rate at which they are presented. Continuous media include animation, audio and video.

According to Wolf et. al. [35], *multimedia communications* deals with the transfer, the protocols, the services and the mechanisms of/for multimedia in/over digital networks.

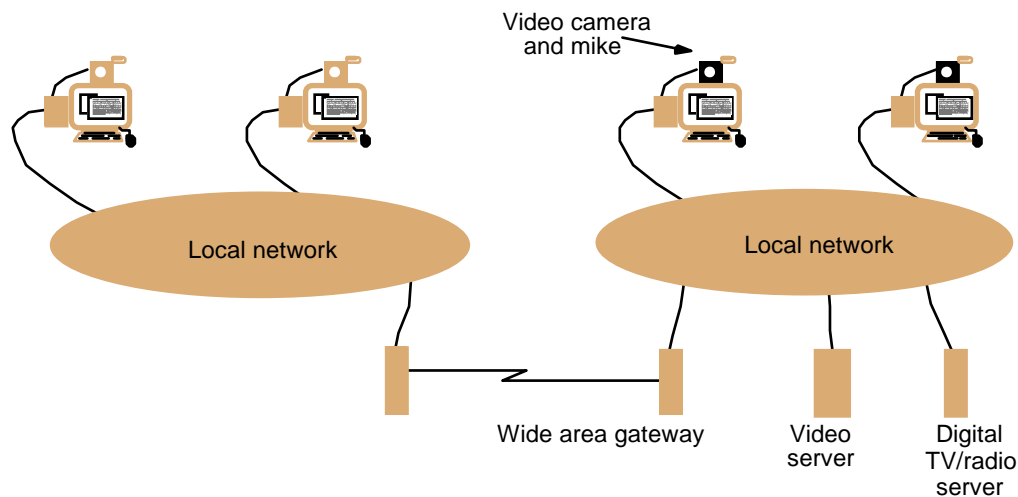
Multimedia systems can be classified into standalone multimedia systems and distributed multimedia systems [9]. Standalone multimedia systems use dedicated system resources and multimedia communications is not supported, while according to Steinmetz and Nahrstedt [26], in distributed multimedia systems, data of digital and continuous media are transmitted and information exchange takes place.

Moreover, in digital networks, transmitted information or media are divided into packets and subsequently sent away from the source to the destination. A sequence of individual packets transmitted in a time-dependent fashion is called *data streams* or *media streams*. Multimedia communications usually use

*isochronous transmission mode*, which means there are minimum and maximum end-to-end delay for each packet of media streams.

## 2.2 Applications of Distributed Multimedia Systems

Coulouris et. al. [6] proposed the architecture of distributed multimedia systems as shown in Figure 2-1. It shows some multimedia systems on LANs which are connected through a WAN. The multimedia systems here are capable of accessing digital video server and digital TV/radio broadcast.



**Figure 2-1** Architecture of distributed multimedia systems (courtesy of Coulouris et. al., 2001).

The typical distributed multimedia systems are capable of supporting variety of applications. In general, there are four types of applications [9], i.e.:

- Conversational applications.
- Messaging applications.
- Retrieval applications.
- Distribution applications.

### 2.2.1 Conversational Applications

*Conversational applications* (or *live applications*) deal with bi-directional communications with real-time, end-to-end media transfer. They imply a human user and another human user or a system.

The examples of conversational applications are video conference and video telephony.

### 2.2.2 Messaging Applications

Messaging applications cover the non-real-time or asynchronous exchange of multimedia data via electronic mailboxes.



### 2.2.3 Retrieval applications

Retrieval applications allows the users to retrieve media data stored in a server. The media data are available to the users anytime and they are exclusively transmitted from the server to the requesting client. Moreover, the users is able to control the media streams, for example play, pause, stop, rewind or fast forward.

The examples of retrieval applications are VOD (*Video On-Demand*) and AOD (*Audio On-Demand*).

### 2.2.4 Distribution Applications

Distribution applications are used to distribute media to a large number of users. There exists a point-to-multipoint connection between the media server and the users. The users are unable to control the media streams except in configurations where one master have the permission to control the streams.

The examples of distribution applications are audio broadcast, such as Internet radio, and TV broadcast.

### 2.2.5 Applications in This Thesis

This thesis only deals with conversational, retrieval and distribution applications. The messaging applications is not covered in this thesis because it is done asynchronously. However, the implementation part of this thesis considers only retrieval and distribution applications.

## 2.3 Multimedia Compression

Uncompressed multimedia data require a lot of storage capacity and very high bandwidth [26]. For example, uncompressed audio streams of CD quality is sampled at a rate of 44.1 kHz and is quantized with 16 bits per sample in 2 channels, hence the bandwidth requirements is  $44100 \times 16 \times 2 = 1.41$  Mbps.

Table 2-1 shows bandwidth requirements for some multimedia streams. It is shown in this table that uncompressed standard TV video cannot even be transmitted via 100 Mbps Ethernet LAN.

**Table 2-1** *Bandwidth requirements of uncompressed multimedia streams.*

	<b>Sample rates or Dimensions</b>	<b>Bandwidth Requirements</b>
Telephone speech	8 kHz, 8 bit, 1 channel	64 kbps
CD-quality sound	44.1 kHz, 16 bit, 2 channels	1.41 Mbps
Standard TV video	640 x 480 pixels x 16 bit, 25 fps	123 Mbps

The use of multimedia compression is therefore very essential. Since the source should encode the streams and the destination should decode them, multimedia compression imposes substantial loads on processing resources, such as CPU power [6]. Some compression methods even need special-purpose hardware called *codecs* (coders/decoders).

Table 2-2 and Table 2-3 compares some compression algorithms commonly used today. Steinmetz and Nahrstedt [26] and Lu [9] discuss the compression algorithms in more detail.

**Table 2-2** *Comparison of several audio compression algorithms.*

Audio Compression	Sampling Rate	Bits per Sample	Bit Rate	Computational
G.711 ( $\mu$ -Law)	8 kHz	8 bits	64 kbps	Low
G.721	8 kHz	8 bits	32 kbps	Low
G.723	8 kHz	8 bits	24 and 40 kbps	Low
DVI	8 kHz	4 bits	32 kbps	Low
GSM 06.10	-	-	13.2 kbps	Low
MPEG-1 Layer 1	32, 44 and 48 kHz	16 bits	32 – 448 kbps	High
MPEG-1 Layer 2	32, 44 and 48 kHz	16 bits	32 – 384 kbps	High
MPEG-1 Layer 3	32, 44 and 48 kHz	16 bits	32 – 320 kbps	High

**Table 2-3** *Comparison of several video compression algorithms.*

Video Compression	Resolution Format	Bit rate	Computational
H.261	CIF, QCIF	px64 kbps	Low
H.263	CIF, QCIF, SQCIF, 4CIF, 16CIF	28.8 – 768 kbps	Low
MPEG-1	352 x 240 pixels, 30 fps	1.5 Mbps	High
MPEG-1	352 x 288 pixels, 25 fps	1.5 Mbps	High
MPEG-2	720 x 480 pixels, 30 fps	15 Mbps	Very High
MPEG-2	1920 x 1080 pixels, 30 fps	80 Mbps	Very High
MPEG-4	-	28.8 – 500 kbps	High

## 2.4 Networks

As explained in the first section, distributed multimedia systems need digital networks to transmit the streams. Currently there are many types of digital networks, but Tanenbaum [31] and Coulouris et.al. [6] categorize them into five main types, i.e. LAN (*Local Area Network*), MAN (*Metropolitan Area Network*), WAN (*Wide Area Network*), wireless network and internetworking. Since this thesis focuses on a Campus LAN, only LAN and wireless network are discussed here.

### 2.4.1 LAN

LAN uses twisted copper wire, coaxial cable or optical fiber to connect computers in a department or a building. The examples of LAN are Ethernet (IEEE 802.3 standard), which offers 10 Mbps to 1000 Mbps bandwidth, and Token Ring (IEEE 802.5 standard).

### 2.4.2 Wireless Network

Wireless network is used for portable and mobile devices which require wireless communication. In general, wireless network can also be divided into three sub-categories, i.e.:

- WPAN (*Wireless Personal Area Network*)
- WLAN (*Wireless Local Area Network*)
- WWAN (*Wireless Wide Area Network*)

WPAN supports only communication within a few meters. There are two important standards of WPAN, i.e. IrDA and Bluetooth. WPAN is not considered in this thesis due to the limitation of its communication range.

WLAN, like wired LAN, covers a building or a department in a company. Nowadays there are two types of WLAN that are widely used, i.e. WaveLAN (IEEE 802.11 standard), which is also called Wi-Fi, and HiperLAN.

WWAN covers mobile phones networks which are currently used, called second generation of mobile networks. The examples of them are GSM (*Global System for Mobile communication*) used in Europe and CDPD (*Cellular Digital Packet Data*) used in the US. Currently, the 2.5<sup>nd</sup> generation mobile network, called GPRS (*General Packet Radio Services*), is entering the market with higher bandwidth than GSM or CDPD. The future of mobile networks is 3<sup>rd</sup> generation, called UMTS (*Universal Mobile Telecommunication Services*).

### 2.4.3 Comparison

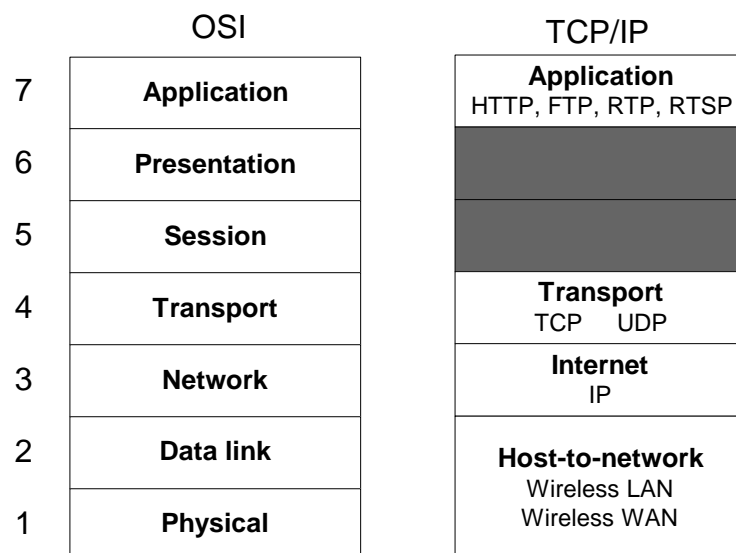
Table 2-4 summarizes some different networks of LAN and wireless network which is mostly used in the discussion of this thesis.

**Table 2-4** Comparison of network bandwidth.

	<b>Network</b>	<b>Bandwidth</b>
LAN	Ethernet (802.3)	10 Mbps and 100 Mbps
	Gigabit Ethernet	1 Gbps
	Token-Ring (802.5)	4 Mbps and 16 Mbps
Wireless LAN	WaveLAN (802.11b)	11 Mbps
	WaveLAN (802.11g)	54 Mbps
	HiperLAN/1	20 Mbps
	HiperLAN/2	54 Mbps
Wireless WAN	GSM	9.6 to 14.4 kbps
	CDPD	19.2 kbps
	HSCSD	56 kbps
	GPRS	56 to 144 kbps
	UMTS	144 kbps, 384 kbps, 2 Mbps

## 2.5 Network Protocols

As explained in the first section, distributed multimedia systems need a digital network to transmit the streams. This thesis is mostly based on IP network. Tanenbaum illustrates the reference model of TCP/IP network as opposed to OSI reference model as in Figure 2-2.

**Figure 2-2** TCP/IP and OSI reference model.

In the lowest layer of TCP/IP model, there are several possible networks which is discussed in the last section. On each upper layer, there is one or more different protocols which can be used.

### 2.5.1 Internet Protocol (IP)

IP (RFC 791) [12] is designed for use in interconnected systems of packet-switched computer communication networks. IP provides for transmitting blocks of data called datagrams or packets from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. IP version 4 (IPv4) uses 32 bit length addresses divided into four parts, for example 129.69.209.104. IP version 6 (IPv6), which is designed to overcome the limitation of IPv4, uses 128 bit length addresses.

### 2.5.2 Transmission Control Protocol (TCP)

TCP (RFC 793) [14] is intended for use as a reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.

Since TCP provides reliable connections, it is used in this thesis for controlling the streams, for example to request a stream or play a stream.

### 2.5.3 User Datagram Protocol (UDP)

UDP (RFC 768) [11] is defined to make available a datagram mode of packet-switched computer communication. UDP is a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. As opposed to TCP, the delivery and duplicate protection are not guaranteed in UDP, in other words it provides unreliable delivery of streams.

Since UDP provides unreliable connections, it is used in this thesis for delivering multimedia streams. In most cases, small packets loss in multimedia streams are acceptable from the user point-of-view. As an addition, TCP is not suitable for delivering multimedia streams because this protocol does not give time integrity. Since TCP is a reliable protocol, it will retransmit any loss packets. In multimedia applications, such conversational applications, re-transmitting packets loss will not make any difference from the recipient side.

### 2.5.4 Real-time Transport Protocol (RTP)

RTP (RFC 1889) [14] provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio or video over multicast or unicast network services. RTP typically runs on top of UDP, but it may be used with other suitable underlying network protocols. RTP adds type identification, sequence numbering, timestamping and delivery monitoring to the IP packets to provide end-to-end delivery services for real-time data.

### 2.5.5 Real-time Transport Streaming Protocol (RTSP)

RTSP (RFC 2326) [16] is an application-level protocol for control over the delivery of data with real-time properties. In other words, RTSP acts as a "network remote control" for multimedia servers. The streams controlled by RTSP usually use RTP, but the operation of RTSP does not depend on the transport protocol. Some methods available in RTSP include SETUP, PLAY, RECORD, PAUSE and TEARDOWN.

### 2.5.6 HyperText Transport Protocol (HTTP)

HTTP (RFC 2616) [15] is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP is widely used in World Wide Web for data transfer, and in fact many media providers store their media files on HTTP servers.

## 2.6 Quality of Service (QoS)

Currently IP protocol only allows end-to-end delivery service with “best-effort” delivery model. It means packets will be delivered to the destination as soon as possible without any commitment to bandwidth or latency. This is not adequate for distributed multimedia systems because the meaning of the packets depends on the time. Some protocols have been introduced to allow end-to-end delivery service on IP protocol, such as ST2 (*The Internet Streaming Protocol Version 2*) [13], Heidelberg Transport System [9] and Tenet [9].

According to Lu [9], distributed multimedia systems need end-to-end guarantees in order to achieve desired application quality. Based on this, the concept of Quality of Service (QoS) is introduced. QoS is normally specified by a set of parameters, for example bit rate, error rate, delay and delay jitter. One or more values might be associated with each QoS parameter. For example, an application may specify bit rate in a range of 100 – 150 kbps and delay bound = 100 ms.

Steinmetz and Nahrstedt [26] divides QoS parameters into three different categories, i.e.:

- *Application QoS*. Application QoS parameters describe requirements for application services, such as media quality and media relations.
- *System QoS*. System QoS parameters describe requirements on the communication services and operating systems, such as throughput, delay, response time, rate, etc.
- *Network QoS*. Network QoS parameters describe requirements on the network services, such as latency, bandwidth, delay jitter, etc.

The simplest QoS model is as follows, an application specifies its QoS requirements, which are submitted to the system. The system determines whether it is able to meet the requirements. If yes, it accepts the application and allocates the necessary resources so that the requirements is satisfied. If it has insufficient resources, the system may reject or negotiate the application by suggesting a lower QoS requirements.

## 2.7 Summary

This chapter gives an overview of some areas in distributed multimedia systems which is used in this thesis. The first discussion gives some important terminologies, followed by some applications of distributed multimedia systems as well as multimedia compression. The next discussion is about networks and networking protocol. The last discussion gives a brief overview of Quality of Service.

## Chapter 3

### Java-based Middleware

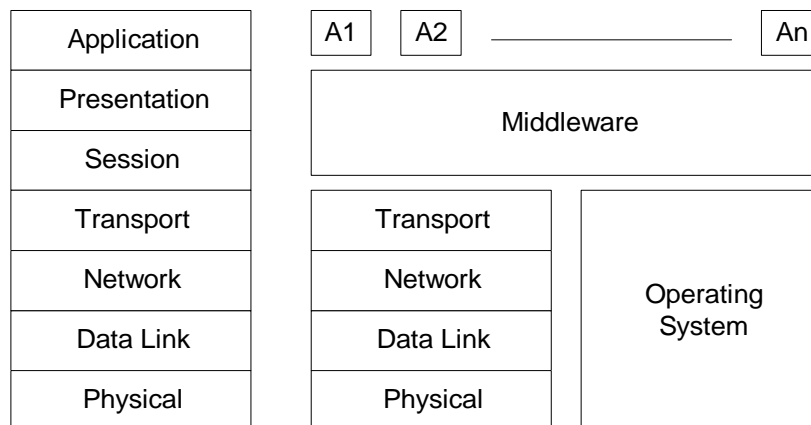
This chapter introduces the concept of middleware in distributed systems. It will only focus on Java-based middleware because the implementation of this thesis uses Java platform. There are two middleware models discussed here, Java RMI (*Remote Method Invocation*), which is Java-version of the RPC (*Remote Procedure Call*), and Jini, which is a service discovery protocol built on the top of RMI.

#### 3.1 Introduction to Middleware

The most common way to make communication between distributed applications is by using operations on sockets. The basic principle is explicitly exchanging messages using send and receive command of the socket mechanisms. The message itself can be encoded in a binary or text form. For example, a client may issue a REQUEST command to a server using TCP protocol. The server then sends back RESPOND command to the client using the same protocol.

In general, sockets are flexible and sufficient for most communications, but there is no distribution transparency, in this case location transparency. Furthermore the design of the socket protocols is cumbersome and error-prone.

Many distributed systems introduce an additional layer, called *middleware*, to overcome this problem. The purpose of middleware is to provide the distribution transparency and hide the heterogeneity of the underlying platforms [32]. The middleware layer sits in the middle between applications and the network operating systems, as illustrated in Figure 3-1.



**Figure 3-1** *Architecture of a distributed system as middleware (courtesy of Rothermel, 2001).*

### 3.1.1 Distribution Transparency

According to Rothermel [24], every distributed systems provides distribution transparency. There are several types of distribution transparency, i.e.:

- *Access transparency*, local and remote resources are accessed in the same way. This thesis supports access transparency because JMF, which is discussed in Appendix A, provides the same way to access media stream from local or remote location.
- *Location transparency*, location of objects is not known to the users. This thesis supports location transparency by using naming service of Jini which hides the actual location of a service.
- *Replication transparency*, the number of copies of an object is not known to the users. This thesis does not support replication transparency, but it might be added in the future. For example, the users may access a source media by using a familiar name, such as “CNN Live”, and this name points to more than one URL.
- *Migration transparency*, objects can migrate without affecting applications. This thesis supports migration transparency because a transcoder service might be moved to another computer. The users are still able to find it using service discovery protocol of Jini.
- *Fragmentation transparency*, objects are accessed without knowledge about any possible fragmentation. This thesis does not support fragmentation transparency.

### 3.1.2 Middleware Models

Most middleware is based on some models for describing distribution and communication. Tanenbaum outlines some of middleware models, i.e.

- *RPC (Remote Procedure Calls)*. This model hides network communication by allowing a process to call a procedure located on a



remote machine. It appears as if the procedure call is executed locally, the calling process is unaware of the fact that communication network takes place. The examples of RPC are Sun RPC and Java RMI.

- *Distributed Objects Invocations.* The idea of this model comes from RPC that if procedure calls could cross machine boundaries, it should also be possible to invoke objects located on remote machines. The essence of distributed objects is that an object implements an interface that hides all implementation details from the users. The examples of distributed object invocations are CORBA (*Common Object Request Broker Architecture*) and Microsoft's DCOM (*Distributed Component Object Model*).
- *Messaging.* The communication is done by passing messages between machines asynchronously. The example of messaging is SOAP (*Simple Object Access Protocol*) which develops an RPC-like model based on XML.

### 3.1.3 Middleware Services

Tanenbaum gives some examples of services common to middleware systems, i.e.:

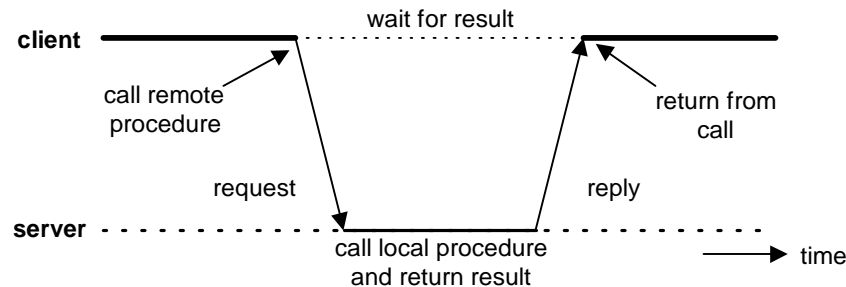
- *Communication Facilities.* The middleware provides communication facilities that hides the low-level message passing through the networks. This thesis uses Java RMI for communication facilities which is discussed later.
- *Naming.* Name services allow entities to be shared and looked up as in directories. The examples of name services are DNS (*Domain Name System*) and X.500. This thesis uses Jini technology which is based on a light version of X.500 name service, called LDAP (*Light Directory Access Protocol*).
- *Persistence storage.* Such service offers special facilities for storage, the simplest form is distributed file systems and the more advanced form has integrated database. This thesis does not use persistence storage.
- *Distributed Transactions.* Such service allows multiple read and write operations to occur atomically. This thesis does not use distributed transactions.

## 3.2 Java RMI

Java RMI is the implementation of RPC in Java platform. However, there is a significant difference between RMI and RPC, RMI deals with methods of distributed objects, instead of procedures. That's why Java RMI is also categorized as distributed object invocations by some people (Coulouris, [6]).

Java RMI (or in more generic term, RPC) itself is a mechanism that allows a machine to call another procedure located on a remote machine. When calling such procedure, the parameters are transparently shipped to the remote machine, and then after the procedure is executed, the result is shipped back to the caller. It

appears as if the procedure call was executed locally. Figure 3-2 shows a basic principle of RPC between a client and server program.



**Figure 3-2** *Principle of RPC between a client and server program (courtesy of Tanenbaum and van Steen, 2002).*

### 3.2.1 RPC Failure Semantics

Rothermel [24] outlines four types of RPC failure semantics, which describes what happens if a failure occurs, i.e.:

- *Maybe*, it means a request is executed by best-effort method.
- *At-least-once*, a request is executed at least once. Java RMI uses this type of RPC failure semantic.
- *At-most-once*, a request is executed at most once.
- *Exactly-once*, a request can only be executed exactly once.

## 3.3 Jini

The second Java-based middleware discussed in this chapter is Jini. As explained above, Jini is used in this thesis for service discovery protocol, that is to find a service of transcoder.

As described by Jini datasheet [30], Jini network technology provides a simple infrastructure for delivering services in a network. Jini technology offers “network plug and work” mechanism, where a service can be connected to a network, announce its presence, and the clients that want to use the service can discover and use it. Although at first, Jini was intended for device discovery, like printer, nowadays Jini is also intended for discovery of software services and even Web services.

Application	Service
Jini Technology	
Java Technology	
Operating System	
Network Transport	

**Figure 3-3** *The Jini architecture (courtesy of Sun Microsystems, 2001).*

One of the design goal of Jini is to provide a system that easily allows clients to look up new services as they become available [32]. Jini provides a specialized lookup service to gain this purpose. A service registers itself by providing a set of (attribute, value)-pairs that describe, for instance, what the service has to offer, and where it can be contacted. A client can look for a service by providing a template to the lookup service. The lookup service then returns information on matching services.

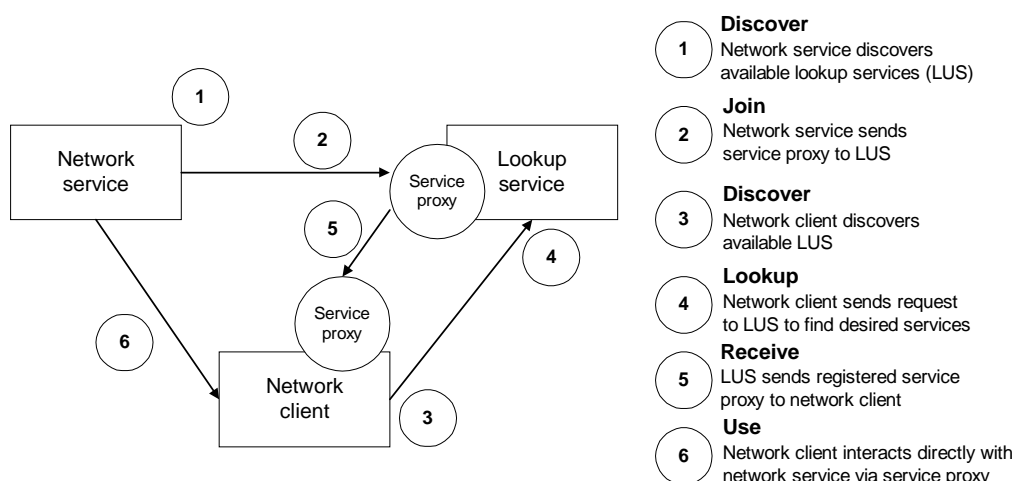
As shown in Figure 3-3, the Jini technology is built on Java technology and utilizing its object oriented features. It is entirely written in Java and it uses the mechanisms of RMI.

### 3.3.1 Architecture of Jini

The architecture of Jini consists of three main components, *service*, *lookup service* and *client*. A service, which is also called a *service provider*, provides a service to the network. A lookup service (or *service locator*), acts as a broker or locator between clients and services. A client basically is a component which makes use of a service.

The heart of Jini is a trio protocol, called *discovery*, *lookup* and *join*. Discovery occurs when a service or client is looking for a lookup service. Lookup occurs when a client needs to use a service. Join occurs when a service is plugged to the network.

When a new service provider is added to a network, it has to find the lookup service to register its service. After the lookup service has been discovered, a service provider registers its service object and its service attributes with the lookup service. The service object contains Java programming language interface for the service. For example, for printer, the interface may contains print method. The service attributes contains additional descriptive information, for example a printer may has a certain speed, either “fast”, “medium” or “slow”.



**Figure 3-4** A flow diagram of Jini technology (courtesy of Sun Microsystems, 2001).

Now the service is ready to be used. A client can ask a certain service by sending request to the lookup service. The client locates the lookup service using discovery protocol. The client locates a service by sending its type, that is interface written in Java programming language, and optionally with some descriptive attributes. The service object is then loaded into the client.

The final stage is to use the service. The client interacts with a service via a set of interfaces, which are implemented as RMI references to the remote object that implements the service. These interfaces define a set of methods which can be used to interface with the service.

### **3.4 Summary**

This chapter discusses about middleware in distributed systems, especially Java-based middleware. There are two main discussion in this chapter, RMI, which is Java version of RPC, and Jini, which is a service discovery protocol for Java.

## **Chapter 4**

# **Transcoding Infrastructure**

This chapter discusses the common infrastructure of transcoding systems. It starts with an example scenario where we find problems in distributed multimedia systems. The discussion is followed by some alternatives to solve the problem and the transcoding infrastructure is introduced here. At the end of this chapter, the requirements of the transcoding infrastructure is presented.

### **4.1 Example Scenario**

Before discussing the transcoding infrastructure, I will give an example scenario where the transcoding infrastructure could be fit into. Bill is a businessman working in a big company in Stuttgart. Now he is going to the airport because he has an important meeting in San Francisco tomorrow. While waiting in the airport, Bill opens his notebook and watches the latest business news from CNBC using WaveLAN in the airport. The news from CNBC, which has a high-quality resolution, should be transcoded to a lower resolution.

Suddenly he has a video-phone call from his partner in the United States, who is using a desktop PC to make a call. Bill wears his headset to do a video-phone using his mobile phone which is connected to a UMTS network. The quality of the video is not so good because the limitation of UMTS network. In this case, the video and audio stream from Bill's partner, who are using a good connection with a good device, should be transcoded so that it can be transmitted via UMTS network smoothly.

After Bill has arrived in San Francisco, he takes a taxi to the hotel. In the taxi, he finds a computer and monitor embedded in the car. While moving to the hotel, he is watching the latest politic news from CNN. The car itself is connected to the wireless network and it supports UMTS network. When watching the video, Bill has experience the quality of the video is changing several times because the car may change its network to different bandwidth. In this case, the stream should be adapted according to the car's network.

## 4.2 Solution

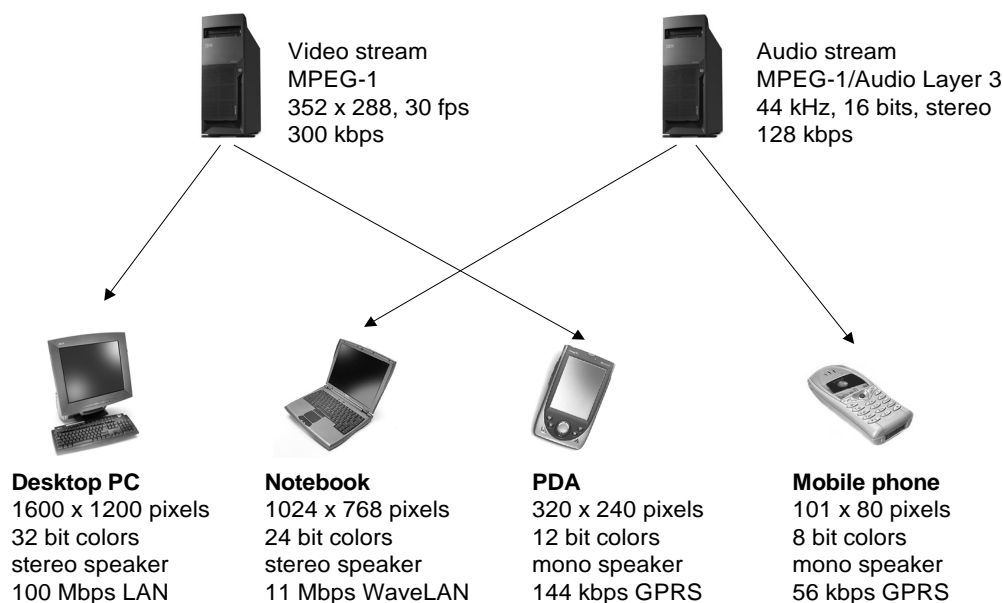
As explained in Chapter 1 and scenario example in the last section, nowadays there are many types of computer devices, such as desktop PCs, notebook PCs, PDAs, mobile phones, which have different capabilities, including computing power and display capabilities. Table 4-1 shows an example of the differences of capabilities between some typical devices.

**Table 4-1** *Different capabilities of some typical computer devices.*

Client Devices	Processor Speed	RAM	Screen Size (pixels)	Color Depth	Speaker
Desktop PC	2 GHz	512 MB	1600 x 1200	32 bit	Stereo
Notebook PC	1 GHz	256 MB	1024 x 768	24 bit	Stereo
PDA	200 MHz	32 MB	240 x 320	12 bit	Mono
Mobile phone	- *)	- *)	101 x 80	8 bit	Mono

\*) not available due to lack of documentation

All of those devices may have different connections to the Net, such as wired LAN or wireless LAN. Furthermore, the users of wireless devices, such as notebook PC or mobile phones, might move while they are connecting to the Net. They may sit on cars or trains and their networks are more likely to change from time to time. One time, they may be connected to a good network, and in another time, they may enter to a congested network.



**Figure 4-1** *Heterogeneity of client devices and network connections.*

The heterogeneity of devices and their connections leads to the problem in multimedia communications. As an example, a multimedia service offers a live

movie in MPEG format. A desktop PC connected to a 100 Mbps LAN may receive it without any problems, a notebook PC connected to a wireless LAN may have experience watching a non-smooth movie because of its not so good connection. A PDA perhaps cannot display the movie at all because the display is too small. The heterogeneity problem is illustrated in

Figure 4-1.

The mobility of clients also lead to the similar problem. The clients may change their connections to other networks which may be congested or have lower bandwidth. In other words, the clients will have difficulties getting QoS guarantee over wireless network because the bandwidth is highly variable [3]. Currently, there are some proposal to achieve QoS guarantees over wireless network, such as adaptive QoS. However, in general, the mobility problems is quite complex due to the nature of wireless network.

#### 4.2.1 Solving Heterogeneity and Mobility Problems

As discussed in the last section, there are two main problems, heterogeneity and mobility. Some people have been working to solve both problems [2][8][25]. However, as discussed in the last section, mobility problem is quite complex and still an open problem nowadays. On the other hand, heterogeneity problems has been solved by some approaches. In general, there are two different approaches, server-side approach and network-side approach.

The server-side approach basically provides several different media formats on the server. It can be implemented by providing several different files which can be selected by the clients. This is the most common approach used today. For example, CNN.com nowadays offers three different formats, QuickTime, Real Player and Windows Media Player, and two different network connections, Dial-up (28 kbps - 56 kbps) and Broadband (150+ kbps). When a user want to watch a streaming video from CNN, he should select the format supported by his hardware/software and the network connection he is using.

The COMCAR project developed a better solution for the server-side approach, called *adaptive mobile application* [20]. The adaptive mobile application, which is installed on the client devices, is able to select the most appropriate format. The end user does not need to select the format manually, instead a smart algorithm will select it. The decision itself is based on some parameters, such as computing power, network bandwidth, available memory, display capabilities, etc.

The server-side approach has an advantage of easy to implement because it does not require any additional infrastructures. It simply adds some new formats on the server and the client selects the most appropriate format for him. However, this approach also has a disadvantage because each time a new video is added, it has to be converted to some other formats and it may take a lot of storage capacity to save the same media in different formats.

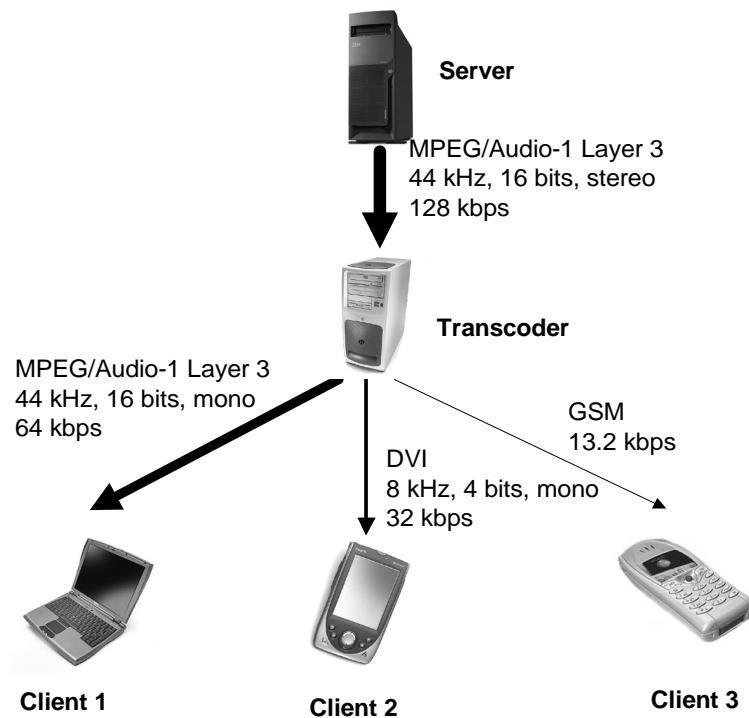
In the network-side approach, there is only one media format on the server and there are some computers on the network, called *transcoders*, that convert the streams into different formats on-the-fly.

The advantage of this approach is flexibility because the server now only store multimedia data in a single format. The disadvantage is the investment cost. The investment cost is quite high because we have to put some new computers. However, the transcoders and the server may run on the same machine to reduce the cost.

This thesis uses the network-side approach to solve the heterogeneity and mobility problems. The next section discusses the transcoder which is the main part of the network-side approach.

#### 4.2.2 Transcoders to Solve Heterogeneity and Mobility Problems

The main task of a transcoder is to transcode the media streams into the appropriate format for the client. This section gives two examples how the transcoders can solve the heterogeneity and mobility problems.

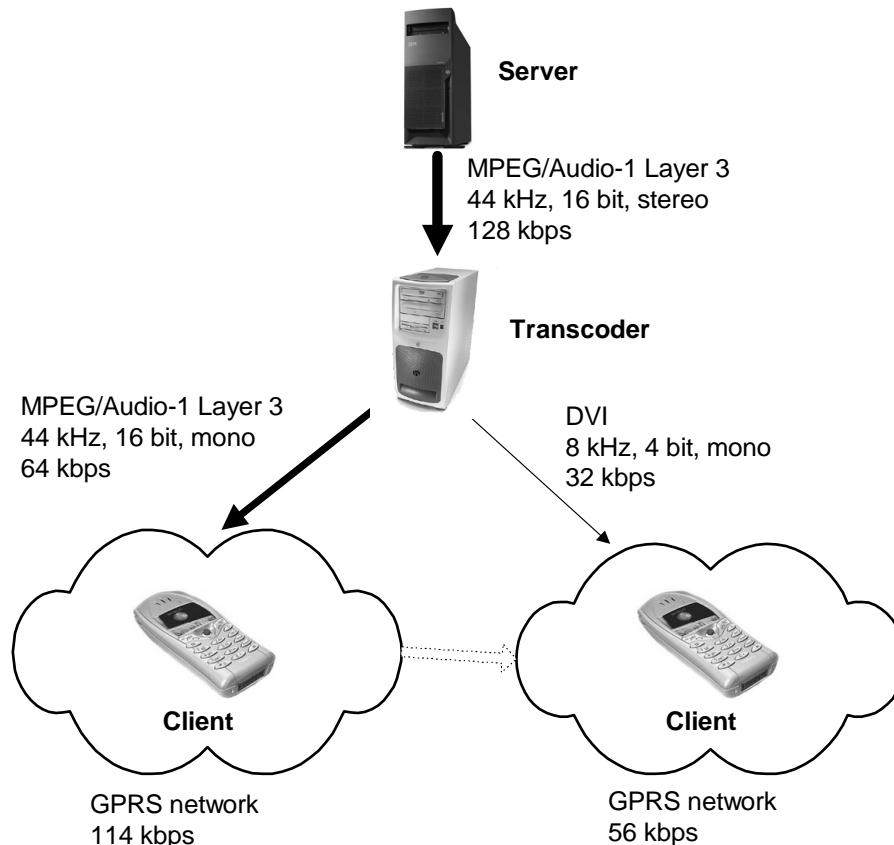


**Figure 4-2** *The architecture of a simple transcoding infrastructure.*

Figure 4-2 shows the architecture of a simple transcoding infrastructure. The media on the server which requires 128 kbps bandwidth is transcoded into three different formats for three different clients with three different connections. The transcoded streams have lower bandwidth than the original one, it requires 64 kbps, 32 kbps and 13.2 kbps respectively. Furthermore, the DVI and GSM require less computing power on the processor, compared to MPEG/Audio. The sound quality of the transcoded streams of course are worse than the original ones. However, the clients would prefer it rather than listening skipped music or news due to the lack of the bandwidths.



Another scenario, an end user is listening for an audio stream in a moving car. While moving, he may change the connection to another network which has lower bandwidth. In this case, the client can request the transcoder to deliver the stream in lower bit rate. Figure 4-3 shows one example of this scenario. The client is moving from a GPRS network with 114 kbps to a new network with 56 kbps. It means that it cannot receive the 64 kbps streams any more, so the transcoder should send another format of 32 kbps.

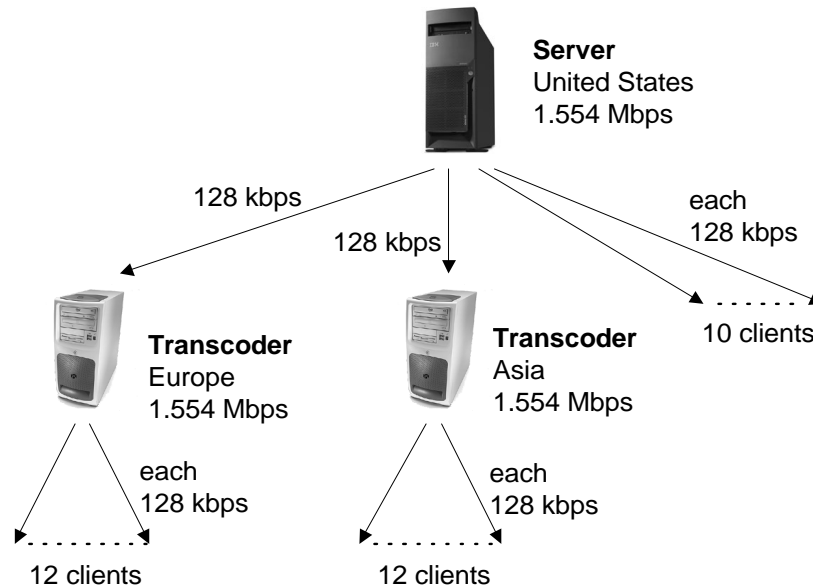


**Figure 4-3** Transcoders for mobile clients.

The introduction of transcoders in the network offers another advantage, that is the ability to share load between transcoders. The concept is very similar to the repeaters in TV broadcast. Unfortunately, this advantage can be gained only in certain situations. The first situation is that the media streams should be conversational or distributed streams, not on-demand streams. The second situation is the server, transcoder, and client are not located on the same LAN.

As an example, suppose a server in the United States, offering MPEG/Audio media 128 kbps stream, is connected to a network with a 1.554 Mbps network. The maximum number of individual users served by the server simultaneously is  $1554 / 128 = 12$  users. Now we introduce two transcoders, for example in Europe and Asia, each of them is connected to a 1.554 Mbps network. The users from these two continents can receive the streams from the transcoders, instead of the server. The situation is shown in Figure 4-4. The maximum

number of users in this infrastructure now is 34 users ( $= 10 + 12 + 12$ ), instead of 12 users.



**Figure 4-4** Another advantage of the transcoding infrastructure.

The transcoder may also perform another function as a cache for the most accessed media, so that the transcoder do not need to re-download the same media when another client ask for it. Unfortunately this scenario is only applied to the on-demand applications, not for conversational and distributed applications.

### 4.2.3 Lookup Service and Service Broker

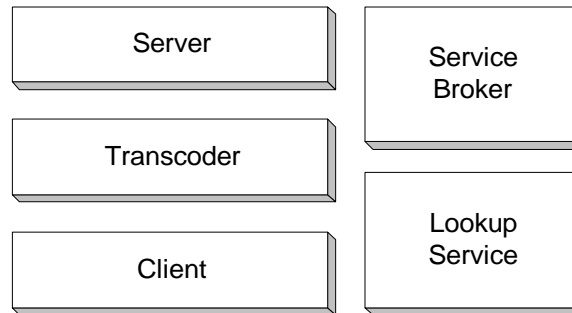
Basically there are three basic components in the transcoding infrastructure, server, client, and transcoder. However, these three components alone is not enough because there are some other issues which should be solved by other components.

The first issue is *service brokering*, how to select the transcoder for the client from a given source media. The solution used in this thesis for service brokering is by introducing a new component, called *service broker*. The service broker acts as a *magic-box* which uses an algorithm to select the most appropriate transcoder for the client.

The second issue is how the service broker knows what kind of transcoders are available in the network. A common method to solve this problem is by sending multicast messages to the network to find the transcoder. Unfortunately this method might not work on some networks because they do not allow multicast messages. Another method, which is relatively better, is by introducing a *directory service*. The directory service stores information about the transcoders, such as supported media formats, location, network connection, etc. The directory service suitable for this purpose is Lightweight Directory Access Protocol (LDAP), which is also a light variant of X-500 naming service. In the

next discussion, the directory service used in the transcoding infrastructure is called *lookup service*.

After the introduction to the two new components, the architecture consists of five elements, server, transcoder, client, service broker and lookup service as shown in Figure 4-5.



**Figure 4-5** *Architecture of the network service infrastructure for transcoding multimedia streams.*

## 4.3 Requirements

### 4.3.1 Server

A server is offering multimedia stream, either in conversational, retrieval or distributed applications. There are several protocols which can be used to deliver the stream, for example HTTP, RTP or even FTP.

In order to maintain compatibility with the existing services, the server is assumed to be a well-known streams, such as CNN or BBC Radio. This assumption makes the server does not need to be changed because it simply use the existing ones. However it makes some limitations to the infrastructure, for example service broker has no control to the server.

### 4.3.2 Transcoder

A transcoder is a service which transcodes a stream from one format to another format which is appropriate for the client. It should register its service to the lookup service with some attributes, such as its address, supported formats, location, etc. The detailed attributes is discussed in Chapter 6.

The transcoder is controllable by the client, it means the client can play, pause, stop, rewind, or fast forward the stream.

### 4.3.3 Client

A client is an end user which requests a stream and plays it. The client sends the request to the service broker along with its information, such as display capabilities and network connection. The service broker then looks for an appropriate transcoder in the lookup service.

#### **4.3.4 Service Broker**

The purpose of the service broker is to find the most appropriate transcoder for the client and then build service chain from the server to the client. It finds the transcoder using a directory which is stored in the lookup service. However, the service broker may also perform other tasks, such load balancing, or performing security tasks.

#### **4.3.5 Lookup Service**

The lookup service is a directory service which stores information about transcoders, for example its address and supported formats. Since the lookup service is very critical, it is recommended that a network has more than one lookup service. However, it depends on the number of clients and the transcoders. For small network, one lookup service may be enough.

### **4.4 Summary**

This chapter discusses the problems in distributed multimedia systems, heterogeneity of client devices and their networks as well as client mobility. There are two different approaches to solve these problems, server-side approach and network-side approach. The approach used in this thesis is network-side approach by having several transcoders on the network to convert streams from the server to the appropriate formats for the clients. The last section of this chapter covers the requirements for all components of the transcoding infrastructure, i.e. server, transcoder, service broker and client.

# Chapter 5

## Architecture Design

This chapter discusses the architecture design of the network service infrastructure for transcoding multimedia streams. There are two main discussions in this chapter, service brokering and service chaining. Server brokering is the algorithm to find the most appropriate transcoder for the client. Server chaining is the protocol to build path from the server to the client via transcoder. The last part of this chapter discusses N-level transcoding where the number of transcoders between server and client is more than one.

### 5.1 Service Brokering

The main task of the service broker is to find the appropriate transcoder for the client. The algorithm to find the transcoder presented here is based on the framework of D. Chen [3]. The basic idea is find *all* transcoders that is capable of serving the client and then choose the *best* one.

#### 5.1.1 Finding Source Format and Destination Format

The first thing to do is to find the source format, that is the media format from the server, and the destination format(s), that is media format(s) supported by the client. Finding the destination formats can be done easily because the client simply tell the service broker which decoders it supports. For example, the client simply tell the service broker, “I have MPEG and H.263 decoders”.

Finding the source format is not as easy as it sounds. The common method is by sending a request to the server. It simply asks the server the format of a given media. Unfortunately, not all servers support this feature, RTSP server is one example of a server that is able to handle such request. The service broker can send request to the RTSP server asking the media format it is sending.

Another method is by providing a meta-information about the format. For example, an end user might tell the service broker about the format of the media he is requesting. This is the easiest way to get the format, but most end users even do not know anything about media format.

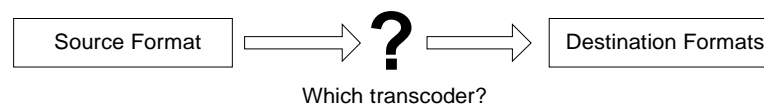
The last method is by providing a database in the service broker or in another computer that contains formats of some well-known address. For

example, the table may look like: “http://tuba:8080/media/starwars.mov” – MPEG-1 352 x 288 pixels 25 fps. The disadvantage of this approach is that the administrator of the infrastructure should provide the database.

As a summary, none of the methods above is perfect, but basically the service broker should know the source format of the source media.

### 5.1.2 Finding Transcoding Format

After the service broker knows the source and destination formats, it should find which transcoder is able to transcode the source media (see Figure 5-1). In some cases, the client does need transcoder at all because the source matches one of the destination formats.



**Figure 5-1** *How to find the transcoder?*

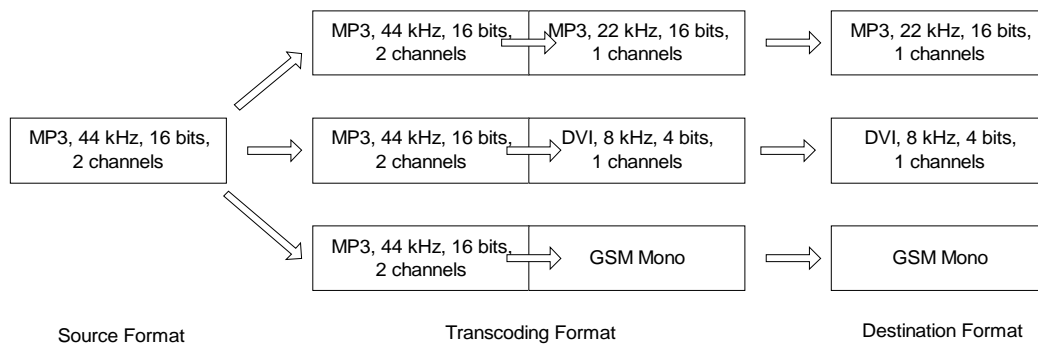
Now the real problem becomes clear, the service broker has a list of transcoders with predefined supported source formats and supported destination formats, and it should find one that fits the question mark in Figure 5-1. Unfortunately, the solution is not easy because the transcoder might have been overloaded or the transcoder does not have enough bandwidth.

In order to find the most appropriate transcoder, the service broker needs other system information from the client. In general, there are three different categories of system information that is needed by the service broker, i.e.:

- Network, such as available bandwidth, delay, jitter, etc.
- Hardware, such as processor speed, processor load, screen resolution, screen color depth, available memory, number of speakers, etc.
- Software, in this case the decoders, such as MPEG, DVI, H.263, etc.

For simplicity, this thesis uses only a few information. For example, in network category, this thesis uses available bandwidth only. For hardware category, this thesis uses processor speed, screen resolution and number of speakers. In the future work, any other information might be used to gain a better result.

The next step is to build a list of destination formats according to the system information given by the client. After that, the service broker builds a list of *transcoding formats* which matches the given source format and the destination formats. Figure 5-2 shows the examples of this list. Suppose that the source format is MP3, 44 kHz, 16 bit, and 2 channels and the client has MPEG, DVI and GSM decoder. Unfortunately, the client cannot receive MPEG stream directly because its bandwidth is not enough, it can only receive MP3 22 kHz, 16 bit and 1 channels. The client, although has DVI decoder, cannot receive all formats of DVI due to bandwidth limitation. It can only receives DVI 8 kHz, 4 bit and 1 channels.



**Figure 5-2** An example of a list of transcoding formats and destination formats.

The service broker has decided that there are three possible formats which can be received the client. Now the service broker should build the transcoding formats that transcodes the source format to each destination format. In the example above, the service broker build three transcoding formats.

There is something missing here, how does the service broker know whether the client's bandwidth is enough or not. Firstly the service broker is given the estimated available bandwidth of the client. There are several ways to detect the available bandwidth in a network, for example using a tool from Jacobson called *PathChar* [17]. Secondly, the service broker is supposed to know the bit rate of each format. For example, the bit rate of MP3, 44 kHz, 16 bit, and 2 channels is around 128 kbps. Basically, the service broker has an algorithm or a table to calculate the bit rate of each format. This thesis simply uses a table similar to Table 2-2, but the best method is actually to use an algorithm which is capable of calculating bit rate for every given media format. Unfortunately, I do not find an algorithm like this, so I simply use a table. However, this method is actually also used in QoS Mapping to map a given resource to the desired format.

### 5.1.3 Assigning Priority to Transcoding Format

The service broker now has to assign priority to each transcoding format. It is needed because one format may have a better quality than the others. For example, an end user would prefer to listen MP3 stream, rather than GSM stream because MP3 stream has much better quality. In this case, MP3 format would have a higher priority.

The priority decision in this thesis also based on a given table. The table itself should be given by a person who is responsible for the network. Each item in this table is a format and its priority, the first item in the table has the highest priority and the last item has the lowest priority. Table 5-1 shows the priority table of audio formats. In reality, the table might be much longer than this. Using this table, the service broker firstly consider MP3, 44 kHz, 16 bit, stereo as the first priority. If the client is not able to receive this format, the service broker consider the second one, MP3, 22.05 kHz, 16 bit, stereo, and so on.

**Table 5-1** *Priority table of audio formats in the service broker.*

Priority	Codec	Sampling rates	Bits/sample	Mono/Stereo	Bit rates (kbps)	Computation
1	MP3	44100	16	Stereo	128	High
2	MP3	22050	16	Stereo	64	High
3	MP3	44100	16	Mono	64	High
4	DVI	22050	4	Mono	64	Low
5	$\mu$ -Law	8000	8	Mono	64	Low
6	DVI	11025	4	Mono	45	Low
7	MP3	22050	16	Mono	32	High
8	DVI	8000	4	Mono	32	Low
9	GSM	8000	8	Mono	13.2	Low

In another case, and end user might only want to receive MP3 streams because he has a powerful processor and wants to listen a “high-quality” audio. In this case, although he has DVI decoder for example, he tells the service broker that his machine only supports MP3 decoder. Therefore the service broker will look at MP3 formats and ignore other formats.

The priority table of video formats is quite similar to Table 5-1, as shown in Table 5-2.

**Table 5-2** *Priority table of video formats in the service broker.*

Priority	Codec	Resolution (W x H pixels)	Frame per second	Bit rates (kbps)	Computation
1	MPEG	352 x 288	30	300	High
3	H.263	352 x 288	30	150	Low
2	MPEG	352 x 288	15	150	High
4	H.263	352 x 288	15	75	Low
5	MPEG	176 x 144	30	150	High
6	H.263	176 x 144	30	75	Low
7	MPEG	176 x 144	15	75	High
8	H.263	176 x 144	15	37.5	Low

The problem becomes quite complicated when the stream contains both video and audio, how to build the transcoding formats. The service broker can build them using Table 5-1 and Table 5-2, for example the service broker select the highest priority for video, MPEG, 352 x 288, 30 fps and then select all audio formats from the highest priority to the lowest one. In other words, for this video format, there are combinations of MPEG 352 x 288, 30 fps and MP3 44 kHz, 16



bit, stereo; MPEG 352 x 288, 30 fps and MP3 22 kHz, 16 bit, stereo; MPEG 352 x 288, 30 fps and MP3 44 kHz, 16 bit, mono, and so on.

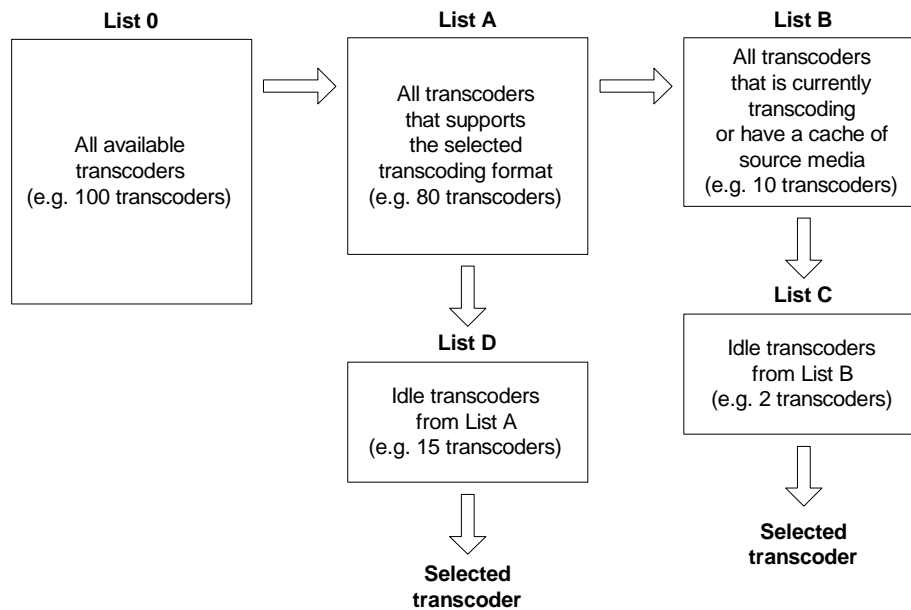
Unfortunately this approach is not reasonable because the service broker will have too many transcoding formats. Suppose that, the list of audio formats contains 10 formats and the list of video formats contains 10 formats, the combination of video and audio formats will contain 100 formats. For this reason, this thesis uses a table of audio and video formats as shown in Table 5-3. Although this solution is not perfect because not all possible formats are considered, but it avoids the increasing number of transcoding formats.

**Table 5-3** *Priority table of audio and video formats in the service broker.*

Priority	Format	Bit rates (kbps)	Computation
1	MPEG 352 x 288, 30 fps MP3 44 kHz, 16 bit, stereo	428	Very high
2	H.263 352 x 288, 30 fps MP3 44 kHz, 16 bit, stereo	278	Low
3	MPEG 352 x 288, 30 fps DVI 22 kHz, 4 bit, mono	364	High
4	H.263 352 x 288, 30 fps DVI 22 kHz, 4 bit, mono	214	Very low
5	MPEG 176 x 144, 30 fps MP3 44 kHz, 16 bit, stereo	278	Very high
6	H.263 176 x 144, 30 fps MP3 22 kHz, 16 bit, stereo	140	Low
7	MPEG 176 x 144, 30 fps DVI 11 kHz, 4 bit, mono	214	High
8	H.263 176 x 144, 30 fps DVI 11 kHz, 4 bit, mono	107	Very low

#### 5.1.4 Cascade Filtering

At this step, the service broker has list of transcoding formats with their priorities. For each transcoding format, started with the highest priority, the service broker must perform *cascade filtering* to all transcoders. The goal is of course the find the “best” transcoder for the client. Figure 5-3 shows the cascade filtering in more detail.



**Figure 5-3** Cascade filtering to the list of transcoders.

In the beginning, there is List 0 containing all transcoders in the network. In this example, suppose that there are 100 transcoders in List 0. Using the selected transcoding format, the service broker select all transcoders which can transcode from the source format to the selected destination format. They are stored in a list, called *List A*.

From List A, the service broker performs a filter again to find all transcoders which is currently transcoding the source media. The purpose of this filter is to save bandwidth from the server to the client. For example, if the user is asking for CNN video and currently there is a transcoder that is transcoding the video, that transcoder is a good candidate to be the selected transcoder. However, this purpose can be achieved in conversational and distributed applications, not in retrieval applications. For retrieval applications, the query might be different, for example, which transcoders have the cache of the requested stream. The result of this filter is stored in a list, called *List B*.

From List B, the service broker finds all transcoders which are capable of serving a new client. This new list, called List C, can be built by considering some QoS parameters which is described in the next section. The filtering itself is done by comparing each required QoS parameter with each available QoS parameter. For example, a transcoding format requires QoS parameters of 128 kbps bandwidth, 200 MHz processor and 1 MB available memory. The service broker then should only filter transcoder which has at least 128 kbps available bandwidth, 200 MHz and 1 MB available memory.

From List C, the service broker simply select one transcoder. The most common method to do it is by randomly selecting one transcoder from the list. Another common method is *round-robin*, but it is quite difficult to be applied here because there may be more than one service broker in the network.

In some cases, List C contains no transcoder, it means there is no idle transcoder right now. What should be done by the service broker is to use List A and find all transcoders which is capable of serving a new client. From this list, called List D, the service broker simply select one transcoder randomly or by other methods.

For optimization purpose, List B and List C can be built using a single query. It means the service broker asks the transcoder whether it is currently transcoding the source media **and** is able to serve a new client.

### 5.1.5 QoS Parameters

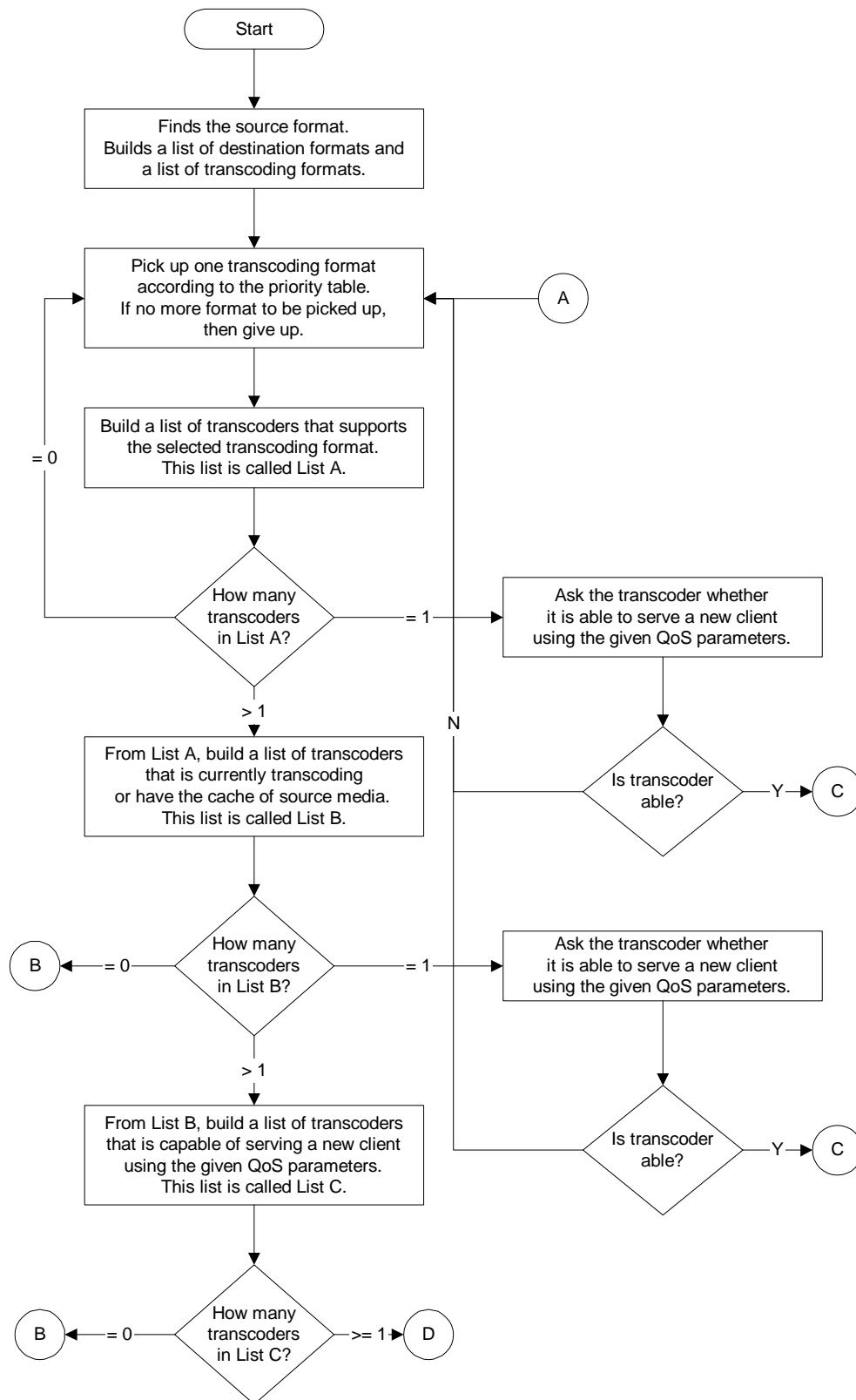
There is an unanswered question in cascade filtering algorithm, how does the transcoder knows whether it is capable of serving a new client or not. The purpose of this query is to get a QoS guarantee so that the client receive a relatively good quality of stream. Because it deals with QoS guarantee, this query involves several QoS parameters from the client as well as the transcoder.

As explained in Chapter 2, there are three QoS parameters, i.e. application, system and network QoS parameters. The application QoS parameters supported in this thesis are given by the users manually. For example, if the user wants to listen high quality audio, he should enable MP3 decoder and disable other decoders only so that the service broker will consider MP3 formats only. Another scenario, when a user need CPU power to do other tasks, it may disable MPEG decoder so that the service broker select formats which needs less CPU power. The system QoS parameters which are used by the service broker include system information from the operating system, such as processor load, available memory, etc. The network QoS parameters which are used by the service broker includes network characteristics, such as bandwidth, latency, jitter, etc.

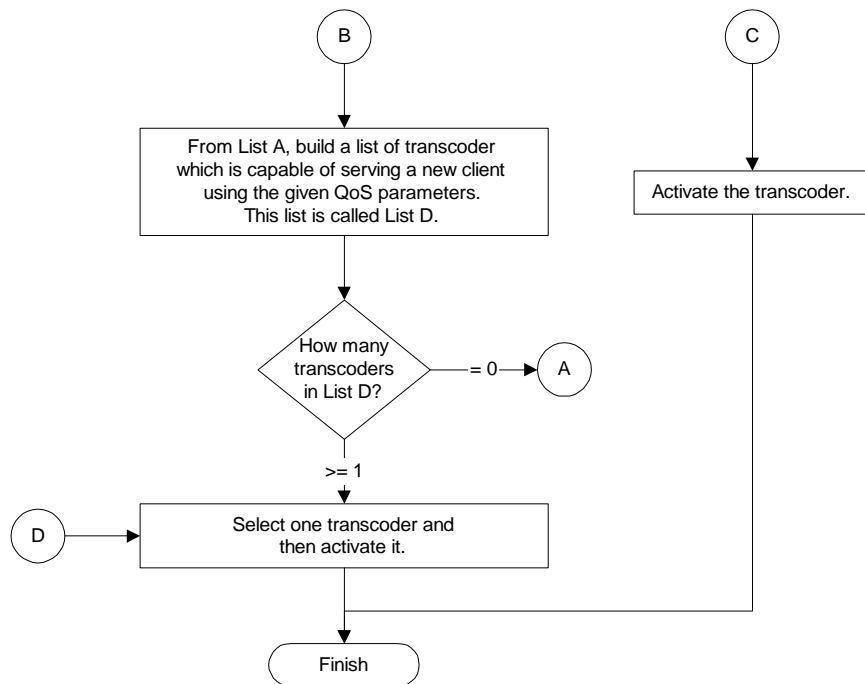
When the service broker receive request from a client for a stream, the client actually also gives some QoS parameters. Since each transcoder also has some QoS parameters, the service broker simply matches the parameters from the client and the transcoder. If the QoS parameters of a transcoder satisfy the requested QoS parameters from the client, it means the transcoder is capable of serving the client.

### 5.1.6 Flow Chart

Figure 5-4 shows the overall flow chart of service brokering. In fact it summarizes the service brokering steps which I explain above.



(continued in the next page)



**Figure 5-4** Flowchart of service brokering.

## 5.2 Service Chaining

As explained in the beginning of this chapter, service chaining is the protocol to build path from the server to the client via transcoder. The basic idea is the service broker searches a directory in the lookup service to select a transcoder and then gives this information back to the client. The client then could listen to the media stream from the selected transcoder. In general, there are three protocols in service chaining i.e.:

- Finding lookup service.
- Service registration.
- Requesting transcoder service.

### 5.2.1 Finding Lookup Service

When a component, either transcoder, service broker or client, is attached on a network, it has to find the lookup service. In general, there are three different methods which can be used to find the lookup service on a network, i.e.:

- Multicast request
- Multicast announcement
- Unicast discovery

*Multicast request* is done by sending a multicast message to a well-known address on a network and waits until the lookup service respond to it. Multicast request uses a UDP protocol while the respond from the lookup service uses a TCP protocol. Since UDP packet may lost, the first multicast request may not

reach the lookup service. The most common method to solve this problem is by sending several multicast requests, for example sending 7 multicast requests every 5 seconds. The multicast request has a disadvantage because some networks do not allow multicast messages or no lookup service is available in the multicast range.

*Multicast announcement* is done by sending multicast messages from the lookup service to a well-known address. The client that listens to this multicast message may add the lookup service to its list so that it can be used in the future. The multicast announcement is sent using UDP protocol periodically, for example every 120 seconds. The disadvantage of multicast announcement is similar with the multicast request. In addition the client may need the lookup service before it receives the announcement, in other words, a new client should wait for the announcement message.

*Unicast discovery* is done by entering the address of the lookup service manually. It can be entered using a dialog box, like setting a proxy in a Web browser, or using a configuration file. This method works on all cases, unlike the other two methods. The disadvantage is that the client must know the address of the lookup service, if the client is moving to another network, it must know the address of the new lookup service.

The best method to find lookup service is by combining the three methods above. The client is given well-known addresses of some lookup services. If they are unreachable, the client can send multicast message to find other lookup services. In the mean time, the client should listen for multicast announcement in case there is a new lookup service installed on the network.

Since lookup service is very important, usually there are more than one lookup services on a network to prevent single point of failure. Whenever a lookup service crashes, the client can use another lookup service.

## 5.2.2 Service Registration

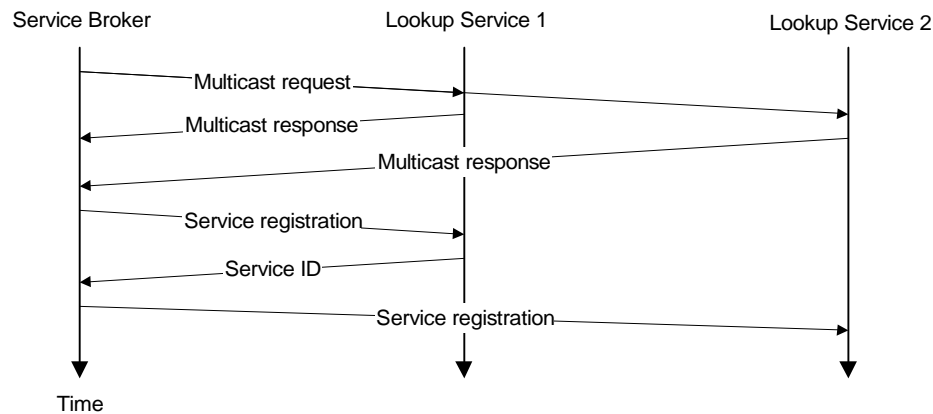
After the lookup service is known, the transcoder and the service broker have to register with it. The registration of service broker is quite simple because it does not require any attributes. It is done by sending a TCP packet to a lookup service. If the lookup service is successfully added the service to its directory, it will return a unique *service ID*. The generation of a unique service ID is discussed later. If the registration fails, the service broker should find another lookup service and register with it.

The service ID is then used by the service broker to register with other lookup services. When a new lookup service is added to the network, the service broker should use the same service ID to register with it.

### 5.2.2.1 Registration Protocol for Service Broker

Figure 5-5 shows the example of service broker registration using multicast request. There are two lookup services and one service broker here. Firstly, the service broker sends multicast request to find lookup services. The two lookup services respond it with multicast respond. In this case, the service

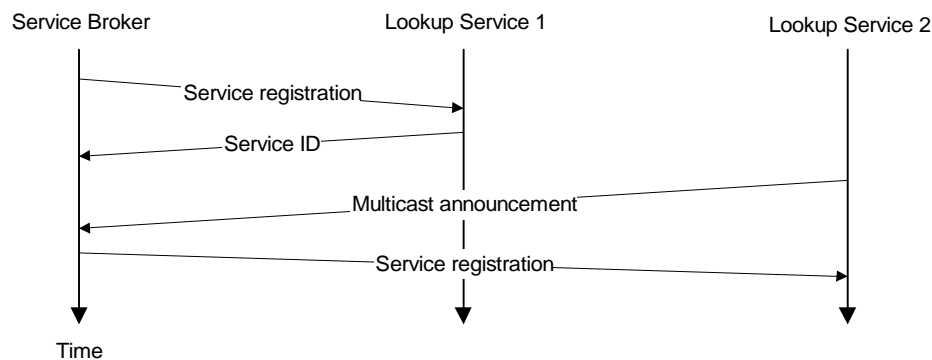
broker simply selects the first respond from lookup service 1 and register its service with it. The lookup service 1 adds the service broker to its database and then returns a service ID. The service ID is used to register with another lookup service, that is lookup service 2.



**Figure 5-5** Registration of service broker using multicast request.

Suppose that the lookup service 1 crashes after sending multicast response. The service registration will fail because the lookup service is no longer available. The service broker then use another lookup service to register its service.

Another scenario, suppose that lookup service 1 crashes after sending the service ID. In this case the service broker has got a service ID so that the service broker can use it to register with another lookup service.



**Figure 5-6** Registration of service broker using multicast announcement and unicast discovery.

Figure 5-6 shows an example of registration of service broker using multicast announcement and unicast discovery. Firstly, there is only lookup service 1 and the service broker finds it using unicast discovery which is given by the user. The service broker registers with lookup service 1 and gets a service ID. After some time, a new lookup service, that is lookup service 2, is added to the network. It sends multicast announcement so that every hosts in the network

know there is a new lookup service. The service broker then use the same service ID to register its service with lookup service 2.

The service broker is registered with a lookup service for a limited time only. In other words, the service broker leases a service ID from the lookup service. After the leasing time is expired (or almost expired), the service broker should renew its registration or the lookup service will assume the service is no longer available.

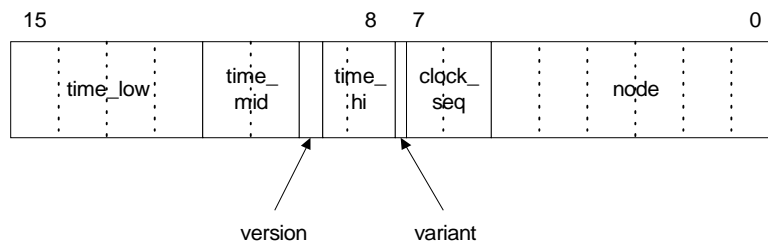
### 5.2.2.2 Registration Protocol for Transcoder

The registration protocol basically is very similar to the registration of service broker. The only difference is that the transcoder registration needs some additional attributes, for instance supported source media formats, supported destination formats, network bandwidth and location. The other processes is the same with the registration of service broker. A service of transcoder is also given a unique service ID which is leased for a limited time. The transcoder should renew it when the leasing time is almost expired.

### 5.2.2.3 Service ID Generator

A unique service ID is important to identify a service, either transcoder or service broker. It also prevents a duplicate registration among lookup services. There are many ways to generate a unique service ID, such as using counter, timestamp or random number.

In this thesis, the service ID is a 128 bits (or 16 bytes) number which is a combination of *timestamp*, *random number* and *host address*. The service ID uses these combinations to guarantee that the possibility of two services have the same ID is almost zero. The service ID uses combination of timestamp and random number, instead of counter because sometimes we need to know when a service is registered or when a service is going to be expired. Figure 5-7 shows the diagram of the service ID.



**Figure 5-7** A time diagram of a service ID.

The *time\_low*, *time\_mid* and *time\_hi* field are set to the least, middle and most significant bits respectively of 60 bit timestamp measured in 100 nanosecond units since midnight, October 15, 1582 UTC. The version field indicates the version, whether 0x1 or 0x4. The variant field must be set 0x02. The *clock\_seq* field is set a 14 bits random number. The node is set to the hardware address of the client.



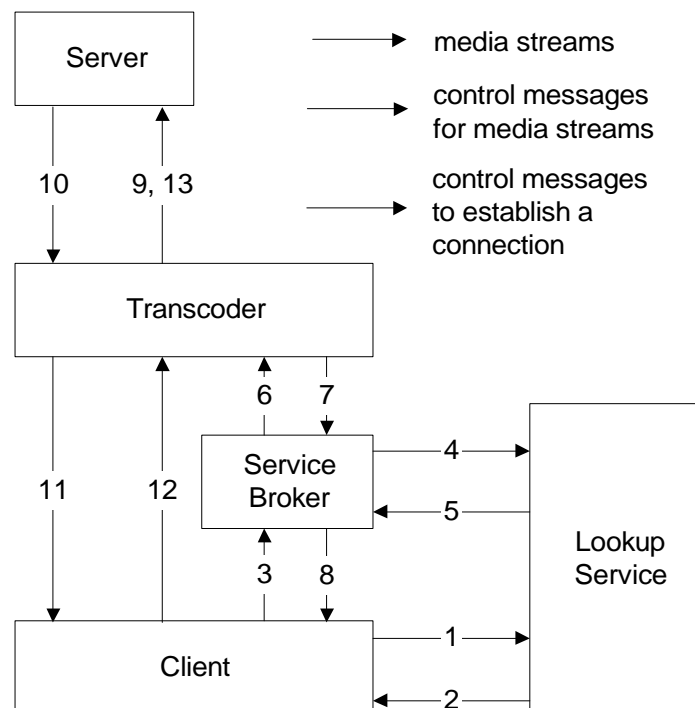
### 5.2.3 Requesting Transcoder Service

At this step, the service broker has registered its service to the lookup service and all transcoders have registered their services to the lookup service as well. When a client request for a media stream, the transcoding infrastructure should be able to find the appropriate transcoder, using service brokering algorithm, and then request the transcoder to start transcoding the stream. Basically there are two approaches which can be used to request a transcoder service, i.e.:

- Client-initiated request.
- Server-initiated request.

#### 5.2.3.1 Client-initiated Request

Client-initiated service request means that the client initiates a request to the service broker to find the transcoder. Figure 5-8 shows both media streams and control messages between components of the transcoding infrastructure. The number on each arrow indicates the step of service chaining protocol, it is discussed later.



**Figure 5-8** Stream and control flow of the client-initiated request.

The media stream flows from the server to the transcoder, where it is transcoded to another format, and then to the client. The control messages are divided into two types. The first type is control messages to establish a connection from the server to the client via transcoder. The second type is control messages to control the media, such as play, pause, rewind and fast forward.

Since it is client-initiated request, the client should ask the service broker to find the most appropriate transcoder. The service broker itself can be found by sending a request to the lookup service. After the client receives the service broker address, it sends information about requested media stream as well as client's capabilities, such as processor type, monitor resolution and network connection, to the service broker. The service brokering algorithm is performed in this step.

After the transcoder has been found, the service broker then sends a request to the transcoder to send the transcoded stream to the client. The transcoder itself request the original media stream from the server and then creates a transcoder session to transcode it. Along with it, the transcoder sends information, which contains IP address and port number, back to the service broker. The service broker sends this information to the client so that the client can receive the stream.

The explanation of each arrow in Figure 5-8 is as follows:

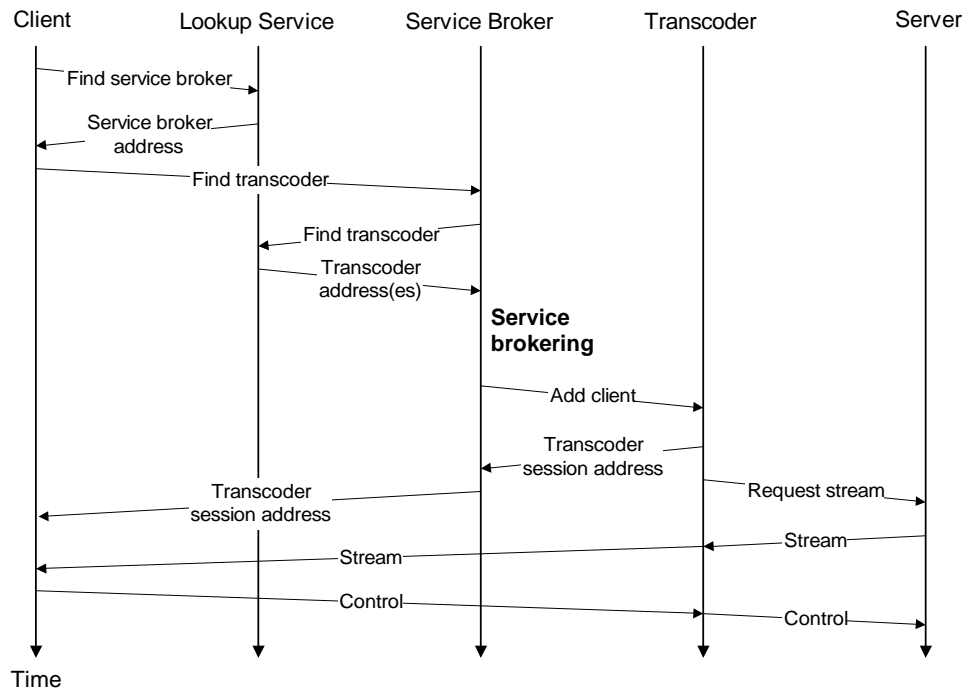
1. The client asks the lookup service for a service broker.
2. The service broker address is returned to the client.
3. The client asks the service broker to find one or more transcoder.
4. The service broker asks the lookup service a list of transcoders which may be appropriate for the client.
5. The lookup service returns a list of transcoders. The service broker then select one which will serve the client using service brokering algorithm.
6. The service broker sends a request to the selected transcoder to send a media stream.
7. The transcoder returns transcoder session address to the service broker.
8. The service broker returns transcoder session address to the client.
9. The transcoder asks the server to send a media stream to it.
10. The server starts sending the stream.
11. The transcoder transcodes the stream and then transmits it to the client.
12. The client control the transcoder to play, stop, pause, rewind or fast forward the stream.
13. The transcoder forward the control from the client to the server.

There is a note in step 12 and 13, some streams, such as conversational or distributed applications, do not allow controls, such as pause or fast forward. For example a live football video cannot be fast forwarded because we do not know the future. In this case, the client is not be able to control the stream. It may send a control message to the transcoder but the transcoder will do nothing.

The service broker address might be cached in the client in order to improve performance. Next time, when a client requests another stream, step 1 and 2 can be eliminated, and the client can send the request directly to the service broker in step 3. The same thing also happens in the service broker, it may caches

the list of transcoders. When another client request the same stream and the same format, it looks in its cache first.

The complete time diagram for a client-initiated request for a media stream is shown in Figure 5-6.

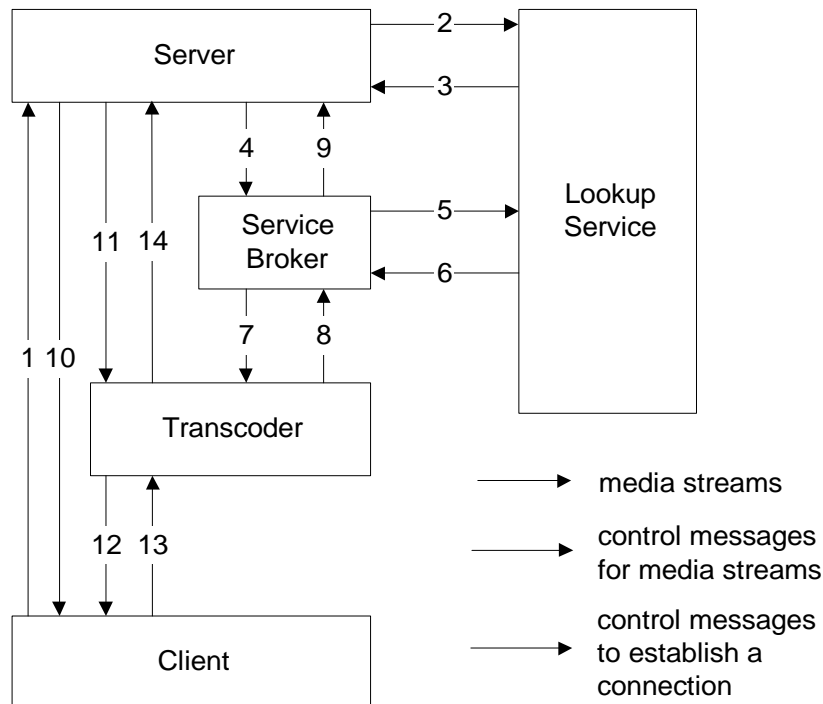


**Figure 5-9** Time diagram of client-initiated request for multimedia streams.

When a client does not want to receive the stream any longer, it should send a message to the transcoder. The transcoder then send stop message to the server to stop the transmission. If the client does not send this message, the transcoder will always send the stream even if the client is not listening. It will create garbage of streams. In some situation, garbage of streams may exist in the network, for example the client crashes before sending stop message to the transcoder. The elimination of this garbage needs another protocol and is discussed later.

#### 5.2.4 Server-initiated Request

Service-initiated request means that request to the service broker is initiated by the server. Figure 5-8 shows media streams and control messages between components of the service-initiated request.



**Figure 5-10** *Architecture of the server-initiated service chaining.*

The protocol is quite different from the client-initiated request because here the client does not need to know about the service broker. It simply sends a request to the server and the server returns back the transcoder address in which the client should listen for the media stream.

Firstly, the client send a request to the server for a media stream. The client has to give information about its capabilities, such as processor load, available memory, network bandwidth, etc. The server then should find the service broker in order to find the appropriate transcoder. If the server does not know the address of the service broker yet, it has to send request to the lookup service.

After the server has the address of the service broker, it sends message to the service broker. The service broker, using the directory in the lookup service, asks a list of transcoders and perform service brokering on this list.

If the service broker find one appropriate transcoder, it send request to the transcoder to create a new transcoder session. The transcoder session address contains the address and port number on the receiver side, to receive stream from the server, and on the sender side, to send the transcoded stream to the client. The transcoder session address is sent back to the server. The server uses the address on the receiver side to start sending the stream and sends the address on the sender side to the client.

The client, after receiving the transcoder session address, opens connection to the given address and port number and listens for the incoming stream. After this step, the client should be able to receive the transcoded stream. Like the

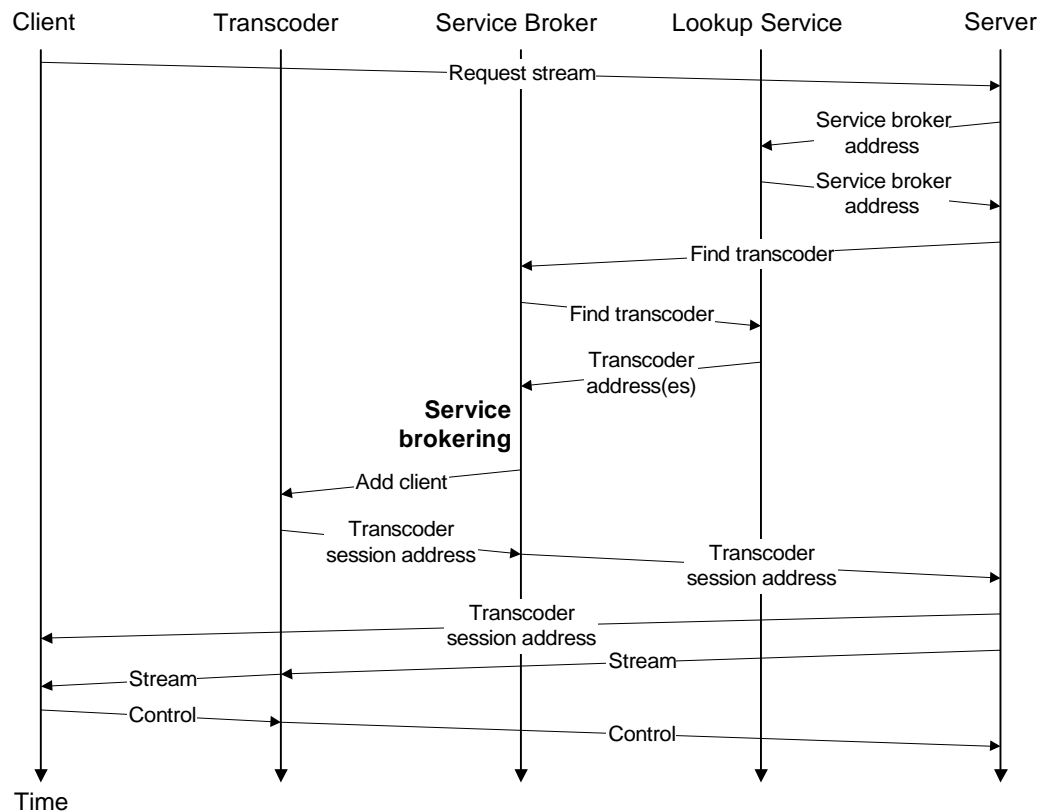
client-initiated service, the client can control the stream, but it depends on the type of the stream.

When a client does not want to receive the stream any longer, it should send a message to the server. The server then sends the stop message to the transcoder. Alternatively, the client can send stop message to the transcoder, and then the transcoder sends the stop message to the server.

The explanation of each arrow in Figure 5-10 is as follows:

1. The client asks the server for a media stream.
2. The server asks the lookup service for a service broker address.
3. The service broker address is returned to the server.
4. The server asks the service broker to find a transcoder.
5. The service broker asks the lookup service a list of transcoders which may be appropriate for the client.
6. The lookup service returns a list of transcoders. The service broker then select one which will serve the client using service brokering algorithm.
7. The service broker sends a request to the selected transcoder to send a media stream.
8. The transcoder returns transcoder session address to the service broker.
9. The service broker returns transcoder session address to the server.
10. The server sends the transcoder session address to the client.
11. The server starts sending the stream.
12. The transcoder transcodes the stream and then transmits it to the client.
13. The client control the transcoder to play, stop, pause, rewind or fast forward the stream.
14. The transcoder forward the control from the client to the server.

The complete time diagram for a server-initiated request for a media stream is shown in Figure 5-6.



**Figure 5-11** Time diagram of server-initiated request for multimedia streams.

#### 5.2.4.1 Comparison of Client-initiated and Server-initiated Request

The client-initiated and server initiated request have advantages and disadvantages. The advantage of client-initiated request is that the server does not need to be replaced. It can be any type of well-known servers, such as HTTP server, RTP server or RTSP server. In server-initiated request, the server must be a “new” server because it has to know the availability of the service broker.

The advantage of server-initiated request is that request from the client is quite simple, thus the client becomes thin. The client simply request to the server and then the server returns back the address of the transcoder to which the client should receive the stream. The client even does not need to know about service broker nor lookup service. In the next discussion, I will focus only on client-initiated request.

#### 5.2.5 Streams Garbage

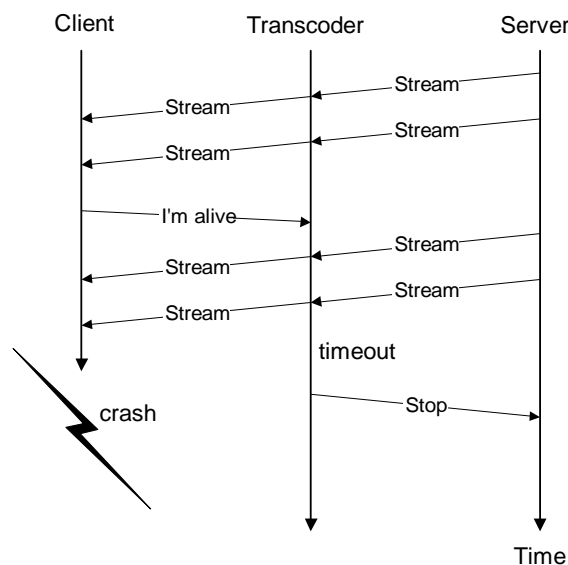
In reality, the client may crash while receiving a stream from the transcoder. If this happens, the transcoder will always transcoding the stream because it does not receive stop message from the client. When a client comes up from the crash and asks for the same stream, the service broker may select another transcoder. This scenario creates streams garbage which sends unnecessary packets to the network.

There are two approaches which can be used to eliminate garbage of stream, i.e.:

- The transcoder always sends a ping message to each client every interval of time, for example every 30 seconds. If a client does not respond to the ping message for a given interval, for example 120 seconds, the transcoder simply stops the stream.
- The client always sends I'm alive message to the transcoder every interval of time, for example every 30 seconds. If the transcoder does not receive I'm alive message from any clients in a given interval, for example 120 seconds, it simply stops the stream.

The architecture of this thesis uses the second approach because it requires one message only, rather than two messages.

Figure 5-12 shows the time diagram of elimination of streams garbage. When a client is receiving a stream, it sends I'm alive message to the transcoder periodically. When the client crashes, the transcoder does not receive I'm alive message in a period of time, so that it must stop the stream.



**Figure 5-12** Time diagram of elimination of streams garbage.

### 5.2.6 Transcoder Handover

Transcoder handover allows a client to switch to another transcoder whenever it needs a different format. For example, a client, which is moving from a 144 kbps network to a 64 kbps, may need to change from 128 kbps MP3 format to 64 kbps MP3 format.

The protocol for transcoder handover is quite similar to the setup of transcoder. The only difference is that the client should stop the stream from the current transcoder and then send another request to the service broker. The rest of the protocol does not need to be changed. Furthermore, in on-demand stream, the new request to the service broker might contain the position of the current stream

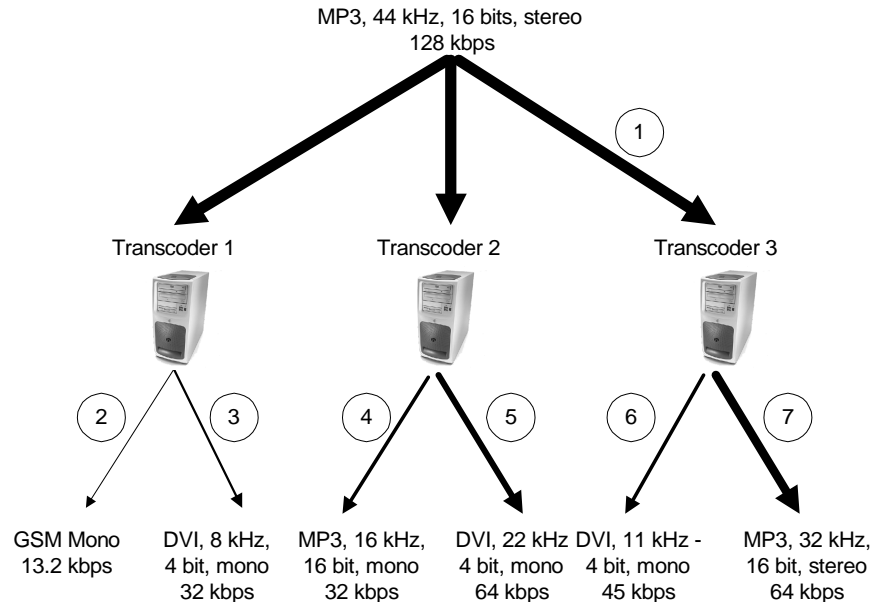
so that the client do not need to play the stream from the beginning. The client will have experience that the stream is being played continuously but with different format.

The only problem with this protocol is that the transcoder setup might need some time within a few seconds. In other words, the client will notice a paused stream when the new transcoder is being initialized. A solution to this problem is by waiting the new stream before stopping the current stream. After the new stream from the new transcoder is arrived, the client sends a stop command to the old transcoder.

### 5.3 Transcoder Configuration

There are two common methods to put the transcoders, flat and hierarchical. The flat method is very simple, each transcoder is given a specific task, to transcode some arbitrary formats to other formats. Coulouris, et. al. [6] proposed hierarchical transcoding infrastructure as another alternative. In this infrastructure original stream is downgraded on each level of transcoding.

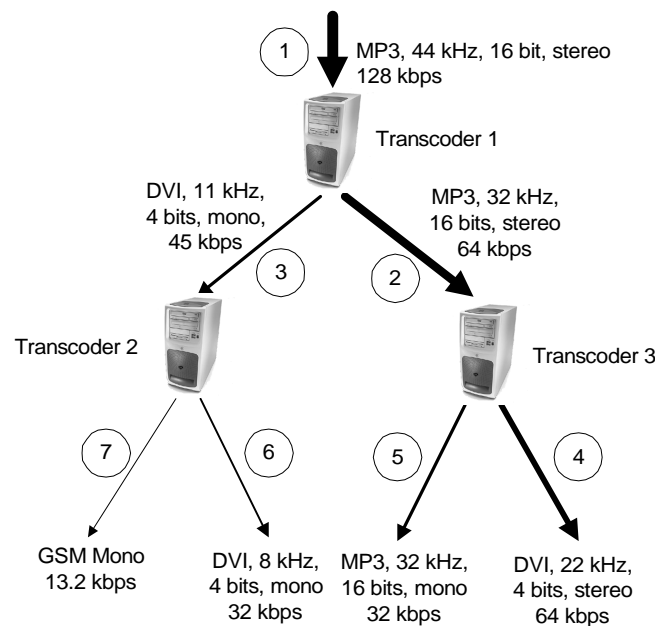
As a simple example, suppose that an administrator is designing a network service infrastructure for transcoding MP3, 44 kHz, 16 bit, stereo to six different formats, i.e. MP3, 32 kHz, 16 bit, stereo; MP3, 32 kHz, 16 bit, mono; DVI, 22 kHz, 4 bit, mono; DVI, 11 kHz, 4 bit, mono; DVI, 8 kHz, 4 bit, mono and GSM-Mono. Using the flat infrastructure, the configuration may look like Figure 5-13.



**Figure 5-13** Example of flat transcoding infrastructure.

There are three transcoders in this example. All of them support MP3, 44 kHz, 16 bit, stereo as their source formats. The same three transcoders above can be configured using hierarchical infrastructure as shown in Figure 5-14.





**Figure 5-14** Example of hierarchical transcoding infrastructure.

Each configuration has advantages and disadvantages. The advantage of hierarchical infrastructure is that it saves a lot of bandwidth. As an example, there are six users who request six different formats provided by the infrastructures above. In flat infrastructure, there is a  $3 \times 128 \text{ kbps} = 384 \text{ kbps}$  traffic from the server to this network. In hierarchical infrastructure, there is a 128 kbps traffic only.

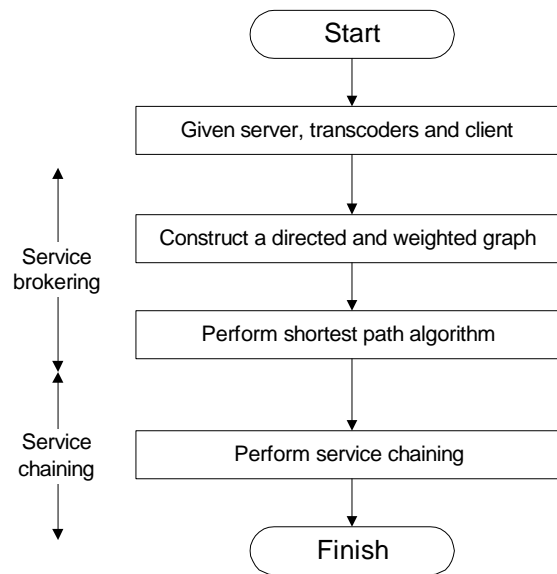
However, the hierarchical infrastructure has a disadvantage because it introduces a single point of failure. If Transcoder 3 in Figure 5-14 crashes, the whole user is not able to receive the stream at all. In the flat structure, only users who request GSM and DVI, 8 kHz, 4 bit, mono are not able to receive the stream.

This thesis uses the configuration of flat infrastructure because the implementation of this thesis only allows one-level transcoding.

## 5.4 N-Level Transcoding

The discussion above assumes there is only one transcoder between server and client. In reality, we might need more than one transcoder because there is no transcoder that is supporting the transcoding format. In another scenario, we might need hierarchical infrastructure as shown in Figure 5-14.

The solution for N-level transcoding is not simple and I do not implement it in this thesis. I only discuss how to construct N-level transcoding theoretically. The main idea is by creating a directed and weighted graph for the transcoders and using the shortest path algorithm to select the chain. The detailed steps are shown in Figure 5-15. Basically the concept is similar to 1-level transcoding except that now the algorithm involves a directed and weighted graph.



**Figure 5-15** *Service brokering and service chaining in N-level transcoding.*

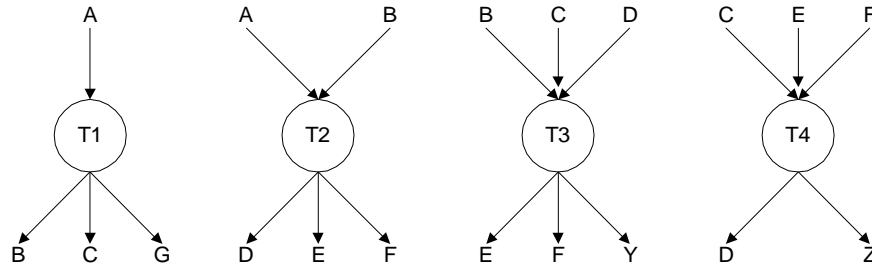
In the first step, the service broker is given a server, one or more transcoders, and a client. The purpose of service brokering algorithm is to find a path from the server to the client via transcoders. In order to achieve this purpose, the service broker then constructs a directed and weighted graph. Using this graph, the path from the server to the client itself can be determined by performing a shortest path algorithm.

The vertices, edges, and cost of the directed and weighted graph can be explained as follows:

- Vertices are the server, the transcoders and the client.
- The edges represent the network connecting two vertices which the destination format of one vertex matches the source format of another vertex. In this thesis, it is assumed that every transcoder is connected to each other.
- The costs of the edges are the QoS parameters in the transcoder and in the network connecting two transcoders.

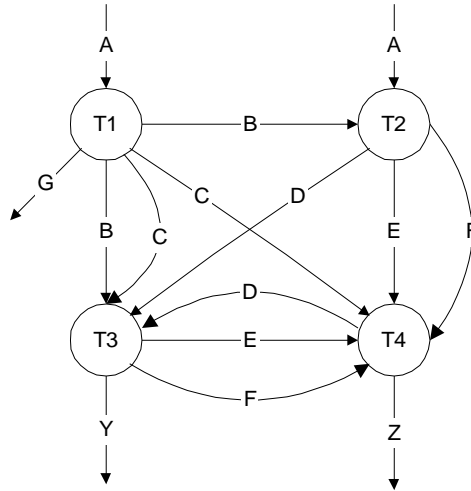
#### 5.4.1 Constructing Directed Graph

The first thing to do in N-level transcoding is to construct a directed graph for the transcoders. To make the explanation easier, let us assume there are four transcoders in the network as shown in Figure 5-16. The first transcoder, T1, supports one source format, A, and three destination formats, B, C and G. The second transcoder, T2, supports two source formats, A and B, and three destination formats, D, E and F. The third transcoder, T3, supports three source formats, B, C and D, and three destination formats E, F and Y. The fourth transcoder, T4, supports three source formats, C, E and F, and two destination formats D and Z.



**Figure 5-16** Example of four transcoders in a network.

From these four transcoders, the directed graph can be constructed. Firstly, we start from the first transcoder which supports destination format B, C, and G. We can draw edges to T2 and T3 because these two transcoders support source formats B. We can also draw other edges to T3 and T4 because these two transcoders support source formats C. Since no transcoder supports format G, we do not need to connect G to any other vertices. Next, we move to T2 and look that T2 supports three destination formats, D, E, and F. We can draw an edge to T3 because this transcoder support source format D. We can also draw edge to T4 because this transcoder support source formats E and F. These steps is done repeatedly until all transcoders are included in the graph. Figure 5-19 shows the directed graph for these transcoders.



**Figure 5-17** Directed graph for the transcoders.

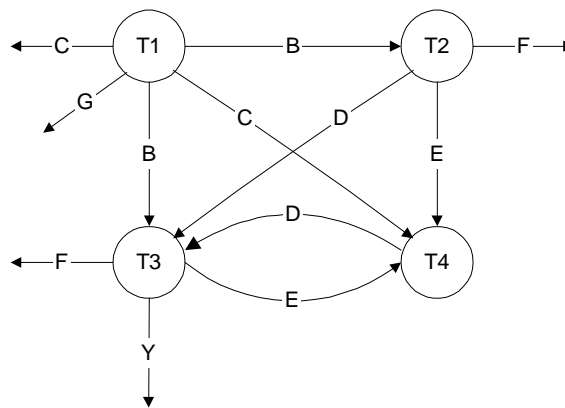
The graph in Figure 5-17 relatively static because the configuration of transcoders is not likely to be changed from time to time. This might be used for optimization purpose because each time a client requests for a stream, the graph does need to be reconstructed. It needs to be reconstructed, for example when a new transcoder is added to the network or a transcoder is removed from the network.

#### 5.4.2 Optimizing Directed Graph

The directed graph in Figure 5-17 is quite complicated although there are only four transcoders. The graph would be more complicated as the number of

transcoder increases. Fortunately, the directed graph can be optimized by reducing the number of edges from one transcoder to another transcoder. For example, there are two edges from T1 to T3, two edges from T2 to T4 and two edges from T3 to T4. We do not need more than one edge to connect two same transcoders, so that we “remove” it from the graph. However, we do not really remove these edges because the format might be requested by a client in a later time. For example, if we remove edge F from T2 – T4 and T3 – T4, when a client requests format F, the graph will not give any result.

The optimization itself can be based on the computing power of the transcoding process. For example, if the computing power of transcoding A to C is higher than transcoding A to B, we can remove edge C from T1 to T3. If the optimization is performed, the directed graph would be like Figure 5-18.

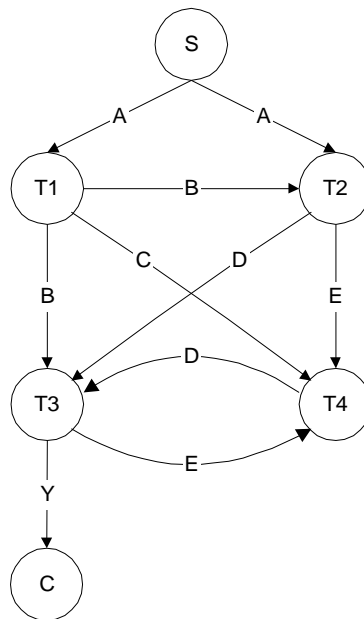


**Figure 5-18** Directed graph after optimization.

### 5.4.3 Adding Client and Server

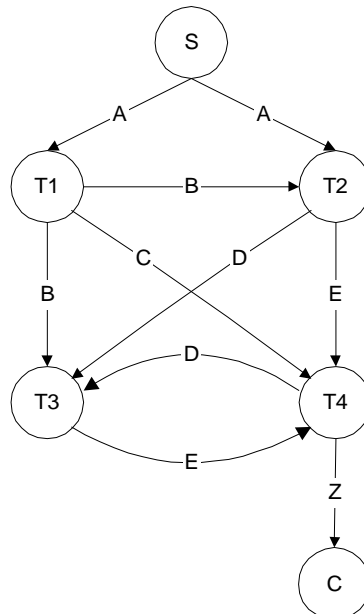
Now, a client is requesting a media from the server, for example in format A. The client itself supports for example two different formats, Y and Z. The task of the service broker is to find the path from the server to the client via the directed graph in Figure 5-19.

Like 1-level transcoding, the service broker should firstly build the table of transcoding formats and then give priority to each format. In this case, there are two transcoding formats only, i.e.  $A \rightarrow Y$  and  $A \rightarrow Z$ . From the priority table, for example  $A \rightarrow Y$  has higher priority than  $A \rightarrow Z$ . The service broker then consider  $A \rightarrow Y$  and build a new directed graph as shown in Figure 5-19. There are two new vertices in this graph, one vertex represents the client and another vertex represents the server.



**Figure 5-19** Directed graph for the transcoders, the server and the client.

Using an algorithm, which is discussed later, the service broker then finds the best path from the server to the client. However, in some cases, the service broker might not be able to do that because there is no end-to-end QoS guarantee in all possible paths. In this situation, the service broker can build another directed graph using the second transcoding format,  $A \rightarrow Z$ . Figure 5-20 shows the directed graph if the transcoding format  $A \rightarrow Z$  is selected.

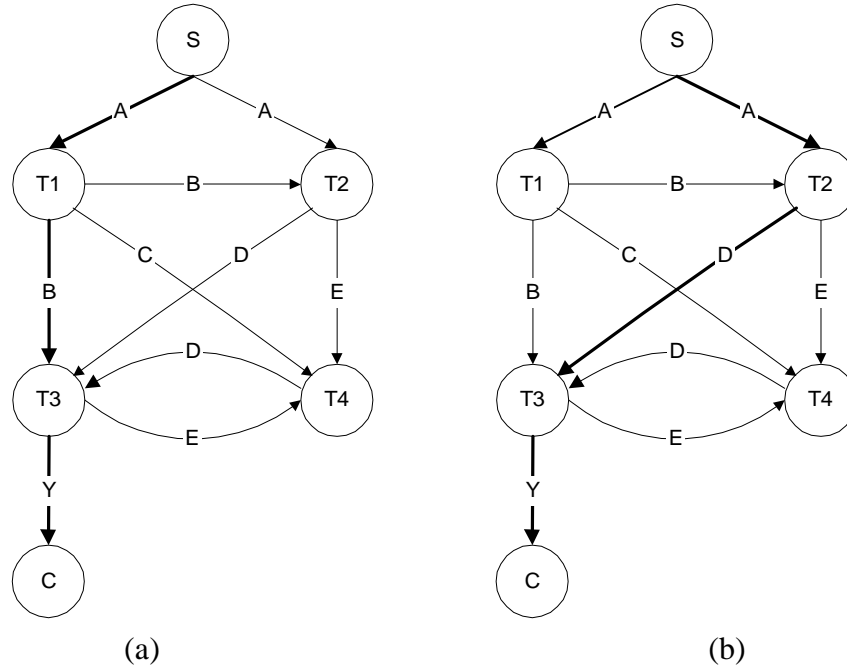


**Figure 5-20** Another directed graph for the transcoders, the server and the client.

The graph in Figure 5-19 and Figure 5-20 looks very similar, but it might give a different solution to the client. For example, if T3 is overloaded, the graph

in Figure 5-19 would not give any solutions, but the graph in Figure 5-20 might give a solution when T4 is idle.

From the graph in Figure 5-19, the chain can be built from the server to the client via T1 and T3 (see Figure 5-21a) or via T2 and T3 (see Figure 5-21b). The chain can even be built from all transcoders, via T1 - T2 - T4 - T3. The algorithm to determine the best path will be discussed later.



**Figure 5-21** Two possible chains from the server to the client.

#### 5.4.4 Constructing Weighted Graph

Like 1-level transcoding, Quality of Service is an important factor to select the transcoders. In N-level transcoding, the QoS parameters can be included as the weight of the graph edges. Since the number of QoS parameters is more than one but each edge of the graph only allows a single value, the QoS parameters can be represented as a single value using percentage.

As an example, we consider four QoS parameters, i.e. current processor load, required memory, required computing power and required bandwidth. If we want to calculate the value of edge between T1 and T3, the processor load is calculated on T1 because this transcoder will transcode format A to format B. The required memory is calculated on T1 based on the memory required to transcode format A to format B. The required computing power is calculated based on how much computing power is needed to transcode format A to format B. The required bandwidth is calculated on the network between T1 and T3 to send format B. As mentioned above, the estimated bandwidth between two nodes might be calculated using a tool such as *PathChar* [17].

In general, the formula to calculate the edge values can be calculated as follows:

$$Value = \alpha.(current\ processor\ load) + \beta.(required\ memory) + \chi.(required\ computing\ power) + \delta.(required\ bandwidth)$$

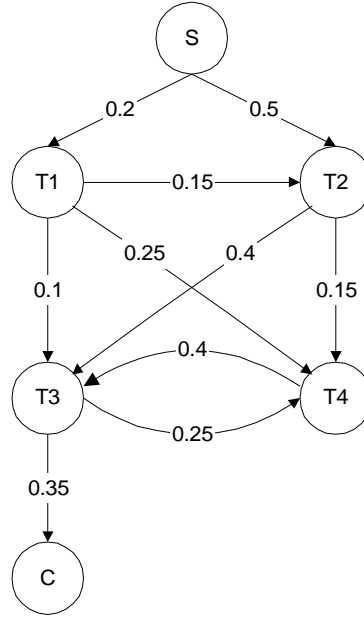
where

$$\alpha + \beta + \chi + \delta = 1$$

The value of  $\alpha$ ,  $\beta$ ,  $\chi$  and  $\delta$  can be given equally, that is 0.25, or non-equally. For example, if the required bandwidth is the most important factor,  $\delta$  can be given 0.4 while  $\alpha$ ,  $\beta$  and  $\chi$  are 0.2 respectively.

Since the value of each edge changes from time to time, the service broker should periodically send a request to each transcoder to update the edge's values. Another way to do this is by allowing the transcoders to broadcast their current system information periodically.

Figure 5-22 shows the example of the weighted and directed graph for the transcoders above.



**Figure 5-22** *Weighted and directed graph for the transcoders.*

#### 5.4.5 Shortest Path Algorithm

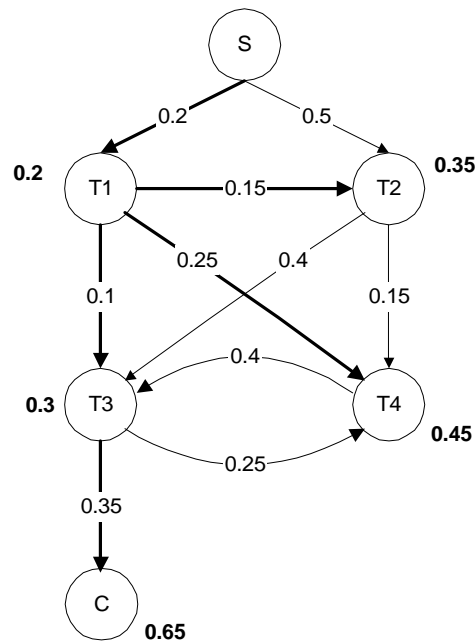
After the directed and weighted graph has been constructed, the next step is to perform shortest path algorithm. There are some algorithms to find the shortest path of a directed and weighted graph, such as bread-first search [1], Dijkstra [1], and Floyd [1]. This thesis uses Dijkstra algorithm because it is simple and relatively fast. Aho, et. al. [1] discusses Dijkstra algorithm in more detailed.

Since Dijkstra algorithm can only be performed on a static graph, the value of each weight during service brokering algorithm is assumed to be constant.

Table 5-4 and Figure 5-23 shows Dijkstra algorithm when it is performed on the directed and weighted graph in Figure 5-22.

**Table 5-4** Finding the shortest path using Dijkstra algorithm.

	S	w	D(T1)	D(T2)	D(T3)	D(T4)	D(C)
0	{S}	-	0.2	0.5	$\infty$	$\infty$	$\infty$
1	{S, T1}	T1	0.2	0.35	0.3	0.45	$\infty$
2	{S, T1, T3}	T3	0.2	0.35	0.3	0.45	0.65
3	{S, T1, T3, T2}	T2	0.2	0.35	0.3	0.45	0.65
4	{S, T1, T3, T2, T4}	T4	0.2	0.35	0.3	0.45	0.65
5	{S, T1, T3, T2, T4, C}	C	0.2	0.35	0.3	0.45	0.65



**Figure 5-23** Finding the shortest path using Dijkstra algorithm.

From Table 5-4, the shortest path from the server to the client is 0.65 and it can be built via T1 and T3. Like 1-level transcoding, the final thing to do before building service chain is to make sure that the selected path has end-to-end QoS guarantee. In some cases, the shortest path might not provide end-to-end QoS guarantee, so the service broker should not build the chain. Once again, if the selected path cannot give QoS guarantee, the service broker should use another transcoding format which has lower priority.

#### 5.4.6 Flow Chart

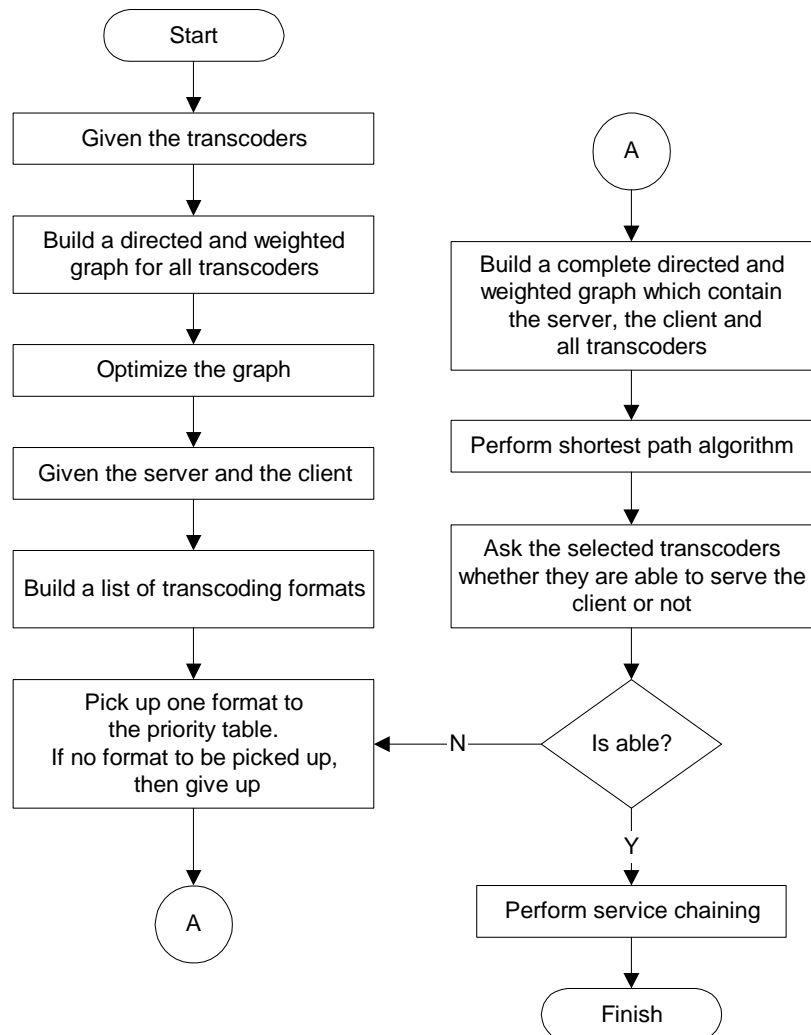
Figure 5-24 shows the complete flow chart of service brokering and service chaining of N-level transcoding. Firstly the service broker has a list of



transcoders. From this list, the service broker is able to build a directed and weighted graph of all transcoders. The graph should also be optimized if necessary.

When a client request for a stream from a given server, the service broker is able to build a transcoding format and give priority to each format (see section 5.1.2 and 5.1.3 for detailed explanation). From this list, the service broker select the highest priority and build a complete directed and weighted graph. This graph contains the server, the client, and all transcoders.

After performing the shortest path algorithm, the service broker send a request to all selected transcoders whether they are able to serve a client or not. If all transcoders answer the request with “yes”, then the service broker build a chain from the server to the client via them. If one ore more transcoder answer with “no”, the service broker pick the second highest priority of transcoding format and then build another graph. This stages are repeated until no transcoding is available to be picked up.



**Figure 5-24** Flow chart of service brokering and service chaining of N-level transcoding.

#### 5.4.7 Implementation Problems

The solution of N-level transcoding described above is not easy to be implemented. Constructing a directed graph may be not too difficult, the service broker could simply send request to the lookup service to return all available transcoders. From this list, the service broker then should be able to construct a directed graph for the transcoders. After this, the service broker should request the lookup service periodically to check whether there is a new transcoder added or a transcoder crashes.

The most difficult task is to give weights to each edge of the graph. The system information of each transcoder is likely to change from time to time. If the service broker is always sending request periodically to all transcoders to ask their current status, the load of the network will increase significantly.

In N-level transcoding, we also have the same problems with 1-level transcoding, that is some QoS parameters are very difficult to find, such as the available bandwidth from one host to another host, for example from T1 to T2.

### 5.5 Summary

This chapter discusses two main issues, service brokering and service chaining. Service brokering is the algorithm to find the most appropriate transcoder for the client. Service chaining is the protocol to build path from the server to the client via transcoder. At the end of this chapter, two configurations of transcoders are discussed and one proposal for N-level transcoding are also discussed.

# Chapter 6

## Implementation

This chapter discusses the implementation of the prototype of network service infrastructure for transcoding multimedia streams. In general, it is divided into two main parts. The first part is the platform in which the prototype is implemented. The second part, which is the main part of this chapter, discusses how the infrastructure is implemented.

### 6.1 Platform

There are three basic decisions that should be made in this implementation, i.e. programming language, communication protocol and service discovery protocol.

#### 6.1.1 Programming Language

The first decision is in which language the prototype should be implemented. There are two main alternatives that is suitable for my purpose, Java and C++. Java is a platform-independent language and it is used in many mobile devices, such as mobile phones. Java also provides a platform independent multimedia library, called JMF (*Java Media Framework*). JMF itself is not part of J2SE, but is available as an extension to J2SE. On the other hand, C++ was an industry standard language a few years ago and it has a good performance. Multimedia library in C++ is platform-independent, for example Windows operating system provides MCI (*Media Control Interface*).

In this thesis, I use Java language because it is platform-independence and it is used in the COMCAR project. Besides that, nowadays there is a growth of computers and devices supporting Java. For example, at the time of writing this thesis, more than 10 millions Java-enabled mobile phones have been released in the market.

The implementation of this thesis is based on *Java 2 Standard Edition* (J2SE) version 1.3.1. The implementation was tested in Sun Solaris machine with Sun OS 5.8 and Intel machine with Windows XP operating system. However, it should be portable to other operating systems, such as Linux as well. Since Java is used, the multimedia library used here is JMF (*Java Media Framework*). The

detailed explanation of JMF can be found in Appendix A. The implementation of this thesis is based on JMF version 2.1.1a.

### 6.1.2 Communication Protocol

The second decision is which communication protocol should be used. There are several alternatives of communication protocol in Java, i.e. sockets, RMI, IIOP (*Internet Inter-ORB Protocol*) and SOAP (*Simple Object Access Protocol*). The implementation of this thesis is based on RMI.

Sockets model is not used in this thesis because as explained in Chapter 3, the implementation would be cumbersome and error-prone. In fact, RTSP which is used quite extensively in this thesis is based purely on sockets model.

IIOP [23], which is a part of CORBA, is a protocol from OMG (*Object Management Group*) which allows computer applications to work together over networks. IIOP allows program from any vendor on almost any computers, operating systems, programming languages and networks can interoperate each other.

CORBA and IIOP is not used in this thesis because it is very complex and has too many overheads. In other words, this thesis does not need CORBA and IIOP because the prototype implemented here is quite simple. Besides that, this thesis is implemented in one language, Java.

SOAP [34] provides a light-weight protocol and it is used widely nowadays. SOAP is an XML-based protocol for information exchange in a decentralized and distributed environment. Unlike RMI and CORBA, SOAP is based on asynchronous communication, it means SOAP messages are fundamentally one-way transmission from a sender to a receiver.

SOAP is not used in this thesis for practical reasons only. The first reason is because service brokering and service chaining should be done synchronously, not asynchronously such as provided by SOAP. Although, SOAP messages can be designed to allow synchronous communication, but it requires more works. The second reason is convenience. As an example, if we use RMI, we can register a transcoder using a simple command, `registerService(formats)`, where `formats` has type of `Format[]`. The `Format` type itself is a class in JMF that represents a format of media data. If we use SOAP, we have to design an XML Schema of the `Format` type and then construct an XML Document.

### 6.1.3 Service Discovery Protocol

The third decision is which service discovery protocol should be used in this thesis. There are two main alternatives here, i.e. Jini and UPnP (*Universal Plug and Play*). Jini is based on Java language, which is appropriate for this thesis, while UPnP is proposed by Microsoft and it has become an open standard technology for connecting devices, PCs and services. Although now there is a Java implementation of UPnP, this thesis is based on Jini.

In general, the basic concept of UPnP [22] is quite similar to Jini technology. Unlike Jini, which is based on Java and RMI, UPnP is based on

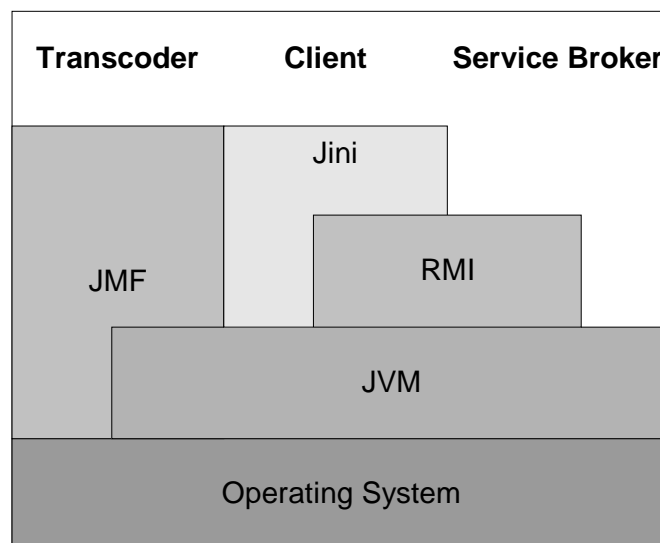
HTTP and XML. The service announcement and service discovery are expressed in XML and are communicated via HTTP. For example, a device or service sends announcement message called ANNOUNCE when it is attached to the network. Another differences between Jini and UPnP is the lookup service. Unlike Jini, in UPnP, a lookup service, which is called a directory service, is optional. A network can have a directory service or not.

This thesis uses Jini because it is based and written entirely in Java and RMI. Besides that, this thesis uses RMI for communication protocol which is also used in Jini.

#### 6.1.4 Architecture

Figure 6-1 shows the architecture of the transcoder, client and service broker from Java perspective. It is shown here that these components use JMF for multimedia processing and RMI for communications and Jini for service discovery.

The server is not discussed in detail because it only hosts multimedia files or transmit media streams via HTTP server, RTP server or RTSP server. Currently there are many HTTP servers available, such as Apache and Microsoft IIS. There are also RTSP servers, such as QuickTime Streaming Server RealSystem Server.



**Figure 6-1** *Architecture of transcoder, client and service broker from Java perspective.*

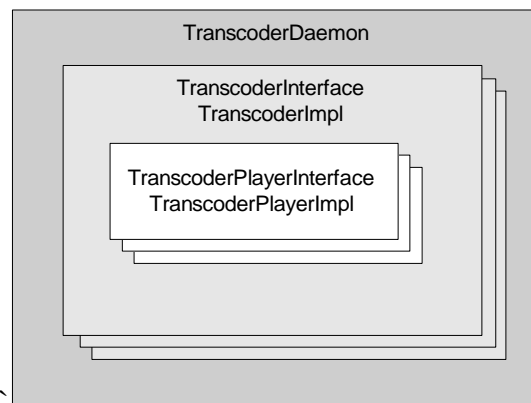
## 6.2 Transcoder

A transcoder basically has two main functions. The first one, as a Jini service, the transcoder must register its service to the lookup service and maintain the registration, for example re-register it when the lease period has been expired. The second one, a transcoder transcodes streams from the server to the client. It means that it has to act as a client for the server and as a server for the client.

As a Jini service, a transcoder has an interface `TranscoderInterface` and its implementation class `TranscoderImpl`. The main task of this class is to receive request from the service broker.

A transcoder may serve several clients, it means that `TranscoderImpl` class may create several different sessions. Each session serves one client, and the client may control the transcoder, for example to play, pause, stop, rewind and fast forward, using RMI. This session is implemented in `TranscoderPlayerInterface` and its implementation class `TranscoderPlayerImpl`.

The registration of transcoder to the service broker and its maintenance is implemented in `TranscoderDaemon` class. Figure 6-2 shows the architecture of a transcoder. It is shown here that one `TranscoderDaemon` may contains several `TranscoderImpl` because one computer may contains several transcoders. Each transcoder itself may serve several clients, thus it may contain several `TranscoderPlayerImpl`.



**Figure 6-2** *Architecture of the transcoder.*

### 6.2.1 `TranscoderInterface` and `TranscoderImpl`

As explained in the last section, `TranscoderInterface` and `TranscoderImpl` receives request from the service broker to serve a client. The simplest form of the interface is as follows:

```
public interface TranscoderInterface extends Remote {
    public TranscoderIdentifier addClient(
        String clientAddress,
        SourceMedia sourceMedia,
        AudioFormat audioFormat,
        VideoFormat videoFormat) throws RemoteException;
}
```

The interface contains one method, `addClient`, which receives four parameters and returns `TranscoderIdentifier`. The `clientAddress` is the address of the client that receives the stream, either IP address, such as

129.69.209.104, or computer name, such as tuba. The `sourceMedia`, which is discussed later, contains information about the source media. The `audioFormat` and `videoFormat` are the formats to which the transcoder should transcode, in other words, these are the video format and/or audio format received by the client.

#### 6.2.1.1 TranscoderIdentifier

The `TranscoderIdentifier` stores an identifier of a transcoder session. It is used by the client to receive the stream from the transcoder as well as to control the transcoder via RMI. There are two basic information in this class, the session address of the transcoder and the address of the RMI object.

```
public class TranscoderIdentifier implements Serializable {  
    public SessionAddress sessionAddress;  
    public String remoteObject;  
}
```

The `sessionAddress` contains the host name, port number and Time to Live (TTL) of the transcoder. The session address should be returned to the client so that the client knows to which address and port it should listen to.

The `remoteObject` is the address of the RMI object of the transcoder. The common syntax of `remoteObject` is `rmi://hostname/transcoding/TranscoderRemote/uniqueID`. In order to guarantee that each transcoder session has a unique address, this thesis uses combination of timestamp and random number. The timestamp used here is the elapsed milliseconds since January 1, 1970 00:00:00 GMT. The random number has a range of 10000 – 50000. An example of the address of an RMI object is shown as follows:

```
rmi://horn/transcoding/TranscoderRemote/1015767147778-15091
```

The probability that two sessions have the same addresses is almost zero because it is not likely that two clients request at the same time (in one millisecond range) and the random number generator at that time produces the same value.

#### 6.2.1.2 SourceMedia

The `SourceMedia` contains information about the source media, such as its address and its format. In general, there are three types of source media which is supported by `SourceMedia`, i.e.:

- URL, including HTTP, FTP and RTSP. For example, `http://tuba/media/starwars.mov`.
- RTP session, indicated by a source address and port number. For example, a multicast stream might have an IP address 224.0.0.1 and port number 22222.

- Familiar name, which can be resolved to a URL or RTP session. For example “Star Wars” may be resolved to <http://tuba/media/starwars.mov>. In this thesis, the real URL or RTP session can be resolved from a lookup-table, which is given by the administrator of the service broker.

Since `SourceMedia` supports three different types of source media, this class has three constructor for each type.

```
public class SourceMedia implements Serializable {  
    public SourceMedia(URL url);  
    public SourceMedia(String sourceAddress, int sourcePort);  
    public SourceMedia(String source);  
}
```

The `SourceMedia` also stores the format of the media, both audio and video format. This format is needed in the service brokering process to find the transcoding format (see section 5.1.1).

### 6.2.2 TranscoderPlayerInterface and TranscoderPlayerImpl

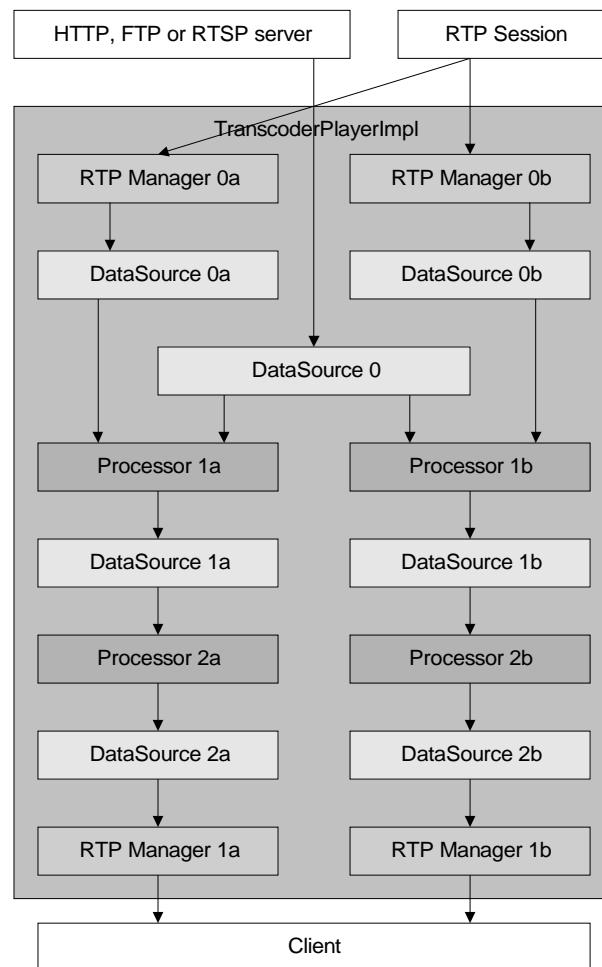
A `TranscoderPlayerInterface` and `TranscoderPlayerImpl` maintains a transcoder session, it means one instance of `TranscoderPlayerImpl` serves one client. For simplicity reason, this thesis assume that source media contains maximum two tracks, one for audio and the other one for video. In other words, the implementation of this thesis can only handle media that contains audio only, video only or audio and video.

Figure 6-3 shows stream flow from the server to the client via `TranscoderPlayerImpl`. There are two possible servers in this case, one possibility is HTTP, FTP and RTSP server, and another possibility is RTP server. For HTTP, FTP and RTSP server, the media can be read by a single `DataSource`. Each track of the `DataSource` is read by a `Processor` to be transcoded into different formats. As explained before, this thesis assume that a media contains maximum of two tracks, so there are two `Processors`, i.e. `Processor 1a` and `Processor 1b`, one for transcoding audio stream and the other one for transcoding video stream.

RTP server needs different handling, because it should be received by an `RTPManager`. Since audio and video track of a media might be sent using two different ports, there are two `RTPManagers`. The `DataSources` of both `RTPManager`, i.e. `DataSource 0a` and `DataSource 0b`, then are read by `Processor 1a` and `Processor 1b` to be transcoded to other formats.

The task of `Processor 2a` and `Processor 2b` is to send the output of `Processor 1a` and `Processor 1b` to the client. It can be done by using one `RTPManager` for each track.





**Figure 6-3** Stream flow in the transcoder player.

`TranscoderPlayerImpl` should provide some methods for the client to control the media. The basic interface for `TranscoderPlayerImpl` can be as simple as follows:

```

public interface TranscoderPlayerInterface extends Remote {
    public void play() throws RemoteException;
    public void stop() throws RemoteException;
    public void pause() throws RemoteException;
    public void rewind() throws RemoteException;
    public void ff() throws RemoteException;
}

```

The client can remotely call one of the methods above by using RMI. The address of RMI object of the `TranscoderPlayerImpl` is stored in `TranscoderIdentifier.remoteObject` which is returned by `TranscoderInterface.addClient` method. The code below shows how to call `stop` method to stop media stream using RMI.

```
try {
    TranscoderPlayerInterface transcoder = (TranscoderPlayerInterface)
        Naming.lookup(remoteObject);
    transcoder.stop();
} catch (Exception ex) {
    System.err.println(ex);
}
```

### 6.2.3 TranscoderDaemon

The instance of TranscoderDaemon in one computer can only be one. The main task of TranscoderDaemon is to register service of the transcoders to the lookup service. A TranscoderDaemon in a computer may also maintain several instance of TranscoderImpls because one computer may have several transcoders in it. The code below shows the registration of an instance of transcoder to the lookup service.

```
TranscoderImpl transcoder = new TranscoderImpl(
    transcoderName, // name of the transcoder
    sourceFormats, // supported source formats
    destFormats); // supported destination formats

JoinManager joinManager = new JoinManager(
    transcoder, // new instance of the transcoder
    attributes, // attributes of the transcoder
    transcoder, // listener to the service ID
    lookupManager, // the lookup discovery manager
    new LeaseRenewalManager());
```

The registration of a transcoder needs some attributes that is used by the lookup service to find a service. The attributes might be name, location, manufacturer, supported formats, etc. Figure 6-4 shows an example of attributes of a transcoder. The transcoder which has a name of “tuba.informatik.uni-stuttgart.de-1” is located on Britwiesenstrasse 20-22 on the first floor and in room 0.113. There are also some attributes about the service, manufacturer, serial number, vendor and version. The transcoder supports two source formats and three destination formats.

TranscoderImpl	
Name.name = "tuba.informatik.uni-stuttgart.de-1"	
Location.building = "Brietwiesenstrasse 20-22"	
Location.floor = "1"	
Location.room = "0.113".	
ServiceInfo.manufacturer = "COMCAR Project"	
ServiceInfo.serialNumber = "123-456-789"	
ServiceInfo.vendor = "University of Stuttgart"	
ServiceInfo.version = "1.0"	
SourceFormatEntry.sourceFormat = new AudioFormat(AudioFormat.MPEG_RTP, 44100, 16, 2));	
SourceFormatEntry.sourceFormat = new AudioFormat(AudioFormat.MPEG_RTP, 44100, 16, 1));	
DestFormatEntry.destFormat = new AudioFormat(AudioFormat.DVI_RTP, 22050, 4, 1));	
DestFormatEntry.destFormat = new AudioFormat(AudioFormat.DVI_RTP, 11025, 4, 1));	
DestFormatEntry.destFormat = new AudioFormat(AudioFormat.DVI_RTP, 8000, 4, 1));	

**Figure 6-4** *Attributes of a transcoder.*

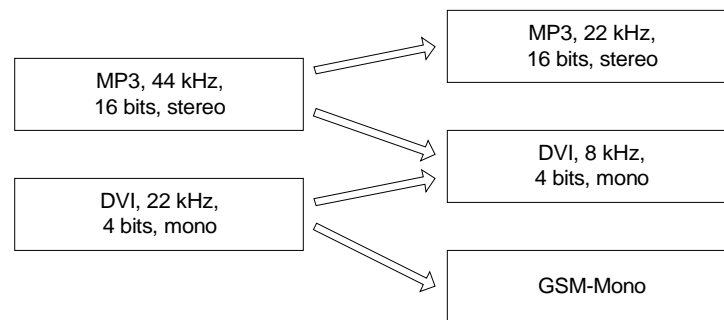
In general, the transcoder may have as many attributes as it can, but the mandatory attributes are `SourceFormatEntry` and `DestFormatEntry`. Both classes are derived from `AbstractEntry` interface because it is required by Jini.

```
public class SourceFormatEntry extends AbstractEntry {
    public Format sourceFormat;
}

public class DestFormatEntry extends AbstractEntry {
    public Format destFormat;
}
```

In the design of this attributes, I assume that all formats of `SourceFormatEntry` must be able to be transcoded to the formats of `DestFormatEntry`. For example, transcoder in Figure 6-4 is able to transcode MP3, 44 kHz, 16 bit, stereo to three formats, DVI, 22 kHz, 4 bit, mono; DVI, 11 kHz, 4 bit, mono; and DVI 8 kHz, 4 bit, mono. It is able to transcode MP3 44 kHz, 16 bit, mono to three formats as well, DVI, 22 kHz, 4 bit, mono; DVI, 11 kHz, 4 bit, mono; and DVI 8 kHz, 4 bit, mono.

In some cases, a transcoder might support several source formats but not all of them can be transcoded to all destination formats. Figure 6-5 shows one example of this scenario. In this case, MP3, 44 kHz, 16 bit, stereo cannot be transcoded to GSM Mono and DVI, 22 kHz, 4 bit, mono cannot be transcoded to MP3, 22 kHz, 16 bit, stereo.



**Figure 6-5** *Not all source formats can be transcoded to the destination formats.*

The solution to the case above is simply by creating two instances of `TranscoderImpl`. It is allowed in this design because one `TranscoderDaemon` may maintain more than one instance of `TranscoderImpl`. Each instance of `TranscoderImpl` supports one source format.

## 6.3 Service Broker

Basically there are two main functions of a service broker. The first one is very similar to the first functions of the transcoder, to register and maintain its service to the lookup service. The second one is to find the appropriate transcoder for the client and to build path from the server to the client.

### 6.3.1 ServiceBrokerDaemon

Like the transcoder, the service broker also has a daemon, called `ServiceBrokerDaemon`. On one computer, there should be only one instance of `ServiceBrokerDaemon`. The main task of this class is to register the service to the lookup service. Unlike the transcoder, an instance of `ServiceBrokerDaemon` can only have one instance of `ServiceBrokerImpl` (see Figure 6-6). The code below shows the registration of the service broker to the lookup service.

```

serviceBroker = new ServiceBrokerImpl(lookupManager);

JoinManager joinManager = new JoinManager(
    serviceBroker, // new instance of the service broker
    null,          // no attributes for this service broker
    serviceBroker, // listener to the service broker
    lookupManager, // the lookup discovery manager
    new LeaseRenewalManager());
  
```



**Figure 6-6** *Architecture of the service broker.*

### 6.3.2 ServiceBrokerInterface and ServiceBrokerImpl

The main task of the `ServiceBrokerInterface` and `ServiceBrokerImpl` is to receive a request from the client and then find the appropriate transcoder. The simplest interface for this purpose is shown as follows:

```

public interface ServiceBrokerInterface extends Remote {
    public TranscoderIdentifier findTranscoder(
        String clientAddress,
        SourceMedia sourceMedia,
        ClientPreferences clientPreferences,
        SystemInfo systemInfo) throws RemoteException;
}

```

The interface contains only one method, called `findTranscoder`, which receives four parameters and returns `TranscoderIdentifier`. The `clientAddress` is the address of the client who request the stream. The `sourceMedia` is the address of the media stream requested by the client. The `clientPreferences` and `systemInfo` are explained below.

### 6.3.3 ClientPreferences

The `ClientPreferences` class implemented in this thesis contains all formats which are supported by the client. However, in the next future, it might contain other attributes, such as location, so that the client can ask “Give me transcoder in building A”. The code below shows the declaration of `ClientPreferences` class.

```

public class ClientPreferences implements Serializable {
    String[] supportedFormat;
}

```

### 6.3.4 SystemInfo

The `SystemInfo` class contains current system information which reflects QoS parameters of the client, such as processor load, available bandwidth, etc. It is used by the service broker to determine whether the client is able to receive a certain format or not. For example, a client with low processor speed and high

processor load might not be able to receive MPEG streams. The code below shows the declaration of `SystemInfo` class.

```
public class SystemInfo implements Serializable {
    int processorSpeed      = 0; // in MHz
    int processorLoad       = 0; // in %
    int totalMemory         = 0; // in MB
    int availableMemory     = 0; // in MB
    int screenWidth         = 0; // in pixels
    int screenHeight        = 0; // in pixels
    int colorDepth          = 0; // in bits
    int numberOfSpeaker     = 0; // mono/stereo/surround
    int estimatedBandwidth  = 0; // in bps
    int availableBandwidth  = 0; // in bps
}
```

In this thesis, most QoS parameters in `ClientPreferences` do not reflect “real” situation because they are given through a user interface. In reality, QoS parameter should be given real values by using some measurements. Actually one task in the COMCAR project deals with how to get QoS parameters.

## 6.4 Client

The most important class in the client side is `ClientImpl`. This class has two main functions, firstly to find the transcoder via service broker, and secondly to create a player in a window and then to play the stream.

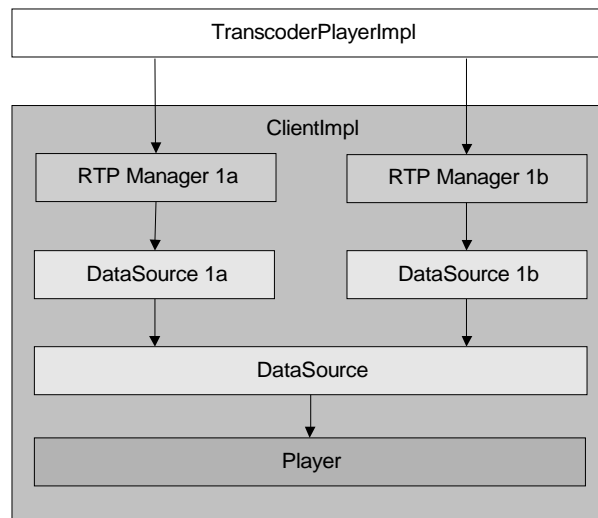
When a client enters a URL to play the stream from the given URL, a `findTranscoder` method is executed.

```
public boolean findTranscoder(SourceMedia sourceMedia,
                             ClientPreferences clientPreferences,
                             SystemInfo systemInfo,
                             Container container,
                             long startTime) {
```

The first three parameters is similar to the first three parameters of `ServiceBrokerInterface.findTranscoder` method. The `container` parameter is the AWT container in which the video and control buttons should be displayed. The `startTime` is the time, in milliseconds, when the media should be played. For example, if the client wants to play the media from the beginning, this parameter is simply given value 0. This parameter is important in transcoder handover because the new transcoder will usually not play the stream from the beginning. This parameter only affects in retrieval applications, not in conversational and distributed applications.

The task of `findTranscoder` method is to find the service broker and then ask the service broker to find a transcoder. After that, this method creates a

new session to receive the stream. The stream flows from the transcoder to the client is shown in Figure 6-7.

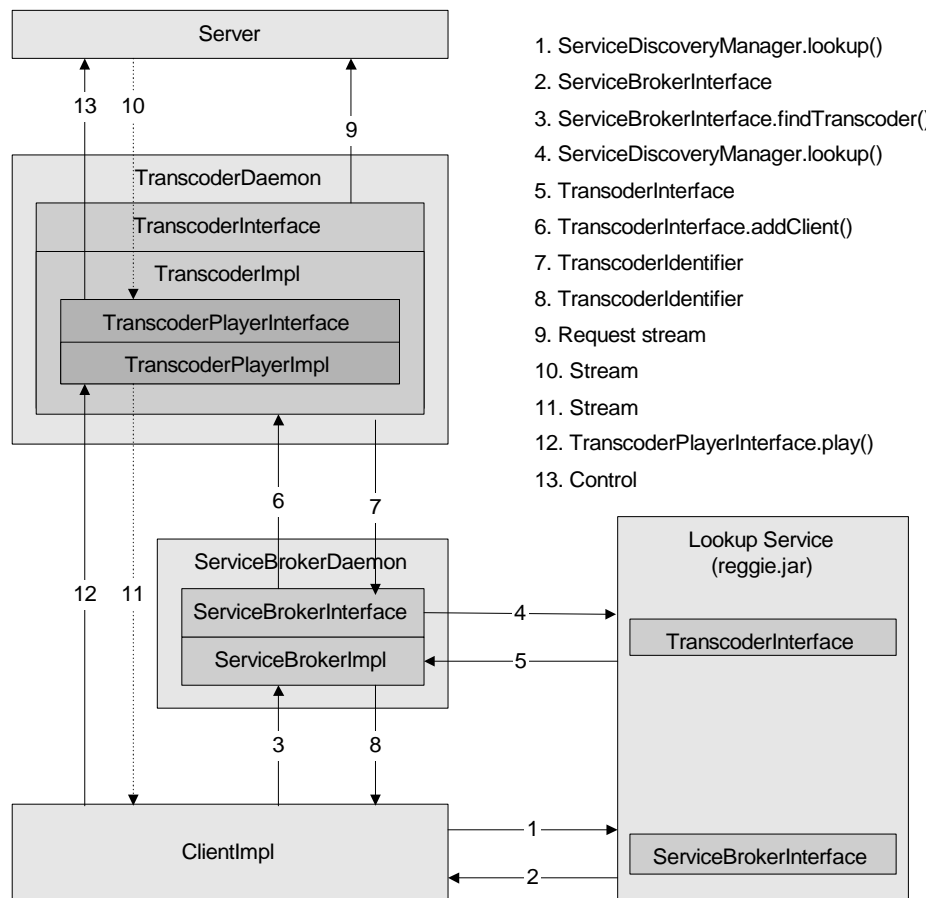


**Figure 6-7** *Stream flow in the client.*

As in the transcoder, there are two RTPManagers in the client, one for audio stream and the other one for video stream. These two clients should be merged into one single DataSource so that the stream can be played without any synchronization problems. According to Sun Microsystems [27], JMF uses the audio stream to synchronize with the video stream in a merged data source.

## 6.5 Establishing Connection

This section summarizes the discussion of the implementation part, it shows briefly how the connection between the client and the server is established. Figure 6-8 is the modified version of Figure 5-8, it shows the protocols to establish a connection from Java perspective. This figure has been simplified because in reality, there is a Web server which hosts the stub files of the transcoder and the service broker. The client may download the stub files of the service broker and the server broker may download the stub files of the transcoder via this Web server.



**Figure 6-8** *Protocols to establish a connection from Java perspective.*

## 6.6 Summary

This chapter discusses the implementation of the prototype of network service infrastructure for transcoding multimedia streams. It starts with the selection of programming languages, communication protocols and service discovery protocols from some available alternatives. The rest of this chapter discusses the technical parts of the implementation.



# Chapter 7

## Integration and Testing

This chapter explains the integration of all components implemented in Chapter 6. The integration includes the installation of each component into several computers. The next step is to test the infrastructure with different source media.

### 7.1 Integration

In the integration of the transcoding infrastructure, I used several computers in the Computer Science department, University of Stuttgart.

#### 7.1.1 Hardware

The hardware used here were three machines of Sun Ultra Sparc II 450 MHz, 512 MB RAM with Sun OS 5.8 operating system and one notebook of Intel Pentium III 600 MHz, 128 MB with Windows XP operating systems.

#### 7.1.2 Software

The list below shows the software used to integrate and test the implementation of this thesis:

- J2SE (*Java 2 Standard Edition*) version 1.3.1.
- JMF (*Java Media Framework*) version 2.1.1a.
- Jini Technology Starter Kit version 1.2.
- Apache Web server 1.3.

The first three software should be installed on all machines, while the last software was only installed on one machine which acts as a Web server.

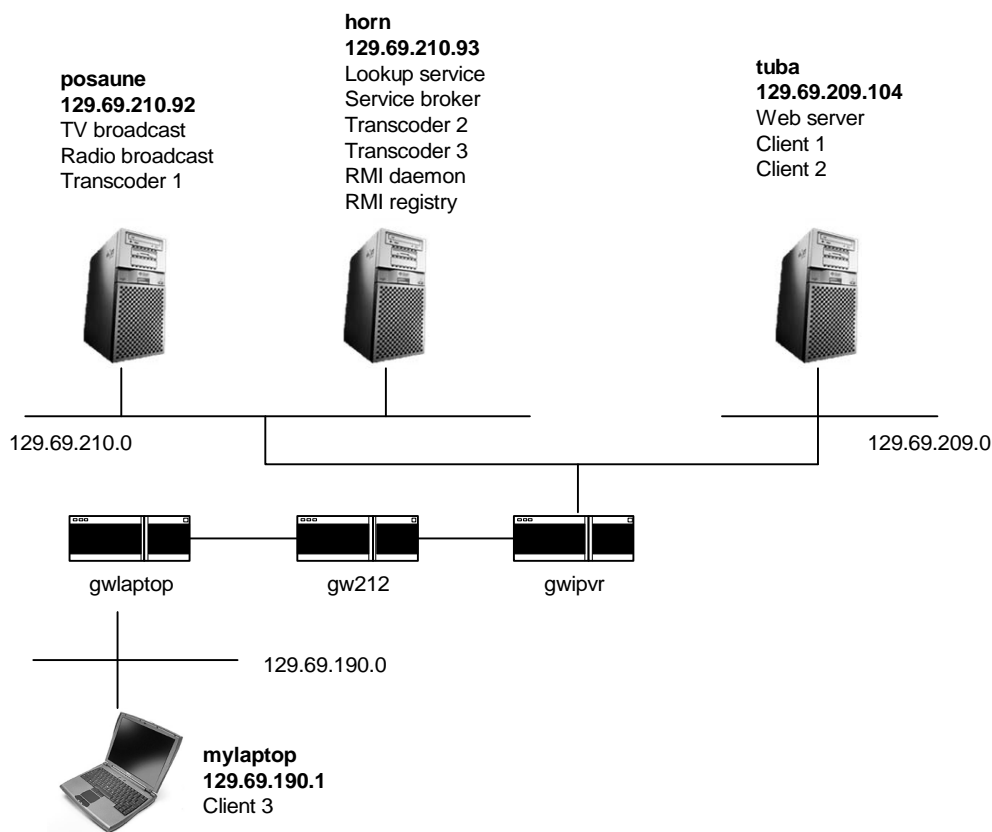
#### 7.1.3 Service

There are two main services I used here, i.e. retrieval and distribution services. The retrieval services are on-demand audio and video streams from a Web server. The distribution services were audio and video broadcast which was sent to a multicast addresses. The audio used here had format of MP3 44 kHz, 16

bit, stereo; while the video had format of MPEG, 352 x 240, 30 fps and MP3 44 kHz, 16 bit, stereo.

### 7.1.4 Installation

As explained in the last chapter, there were five main components of the transcoding infrastructure, i.e. server, transcoder, client, lookup service and service broker. Beside those components, we need a Web server to store the stub files and the media files. We also need RMI daemon and RMI registry to support Jini and Java RMI. The last components I added in the infrastructure is radio broadcast and TV broadcast that delivers audio and video streams to the network. Figure 7-1 shows the installation of these components into the “real” computers.



**Figure 7-1** *Installing the components of the transcoding service into computers.*

#### 7.1.4.1 Web Server

The Web server was installed on the tuba machine on port 8080, so the complete URL of the Web server is <http://tuba.informatik.uni-stuttgart.de:8080/> or <http://129.69.209.104:8080/>. For example to request Star Wars movie from the Web server, one could use the URL of <http://tuba.informatik.uni-stuttgart.de:8080/media/starwars.mov>.

#### 7.1.4.2 Radio and TV Broadcast

The radio broadcast and TV broadcast were sent from posaune machine to the multicast addresses. They were transmitted using JMStudio application, which is provided by JMF. The radio broadcast was sent to the multicast address of 239.0.0.5/22010, while the TV broadcast was sent to the multicast address of 239.0.0.6/22020.

#### 7.1.4.3 RMI Daemon and RMI Registry

RMI daemon and RMI registry are needed for the Jini and Java RMI respectively. Both of them are provided by JDK in two files, rmid and rmiregistry. The simplest way to run RMI daemon and RMI registry is by typing the following command.

```
unsetenv CLASSPATH
rmid -J-Dsun.rmi.activation.execPolicy=none &
rmiregistry &
```

#### 7.1.4.4 Lookup service

The lookup service used in this thesis is the lookup service from Sun Microsystems. The file for lookup service is provided in Jini called reggie.jar. The following command is an example of how to run the lookup service.

```
java -jar
-Djava.security.policy=file:/scratch/jinil_2/policy/policy.all
/scratch/jinil_2/lib/reggie.jar
http://tuba.informatik.uni-stuttgart.de:8080/jini/reggie-dl.jar
/scratch/jinil_2/policy/policy.all
/home/pranatay/tmp/reggie_log public
```

#### 7.1.4.5 Service broker

The main class of the service broker is called ServiceBroker. The service broker can be run in GUI mode or in text mode only. By default, the service broker is run in GUI mode. It can be run in text mode by adding -nogui parameters, such as the example below.

```
java -classpath ".:/scratch/JMF2.1.1/lib/jmf.jar:
/scratch/jinil_2/lib/jini-core.jar:
/scratch/jinil_2/lib/jini-ext.jar:
/home/pranatay/Transcoding/classes/"
-Djava.security.policy=file:/scratch/jinil_2/policy/policy.all
-Djava.rmi.server.codebase=http://129.69.209.104:8080/classes/
com.antonypranata.transcoding.ServiceBroker -nogui
```

#### 7.1.4.6 Transcoder

There are three transcoders in this infrastructure, each of them serves different formats. Table 7-1 shows the list of all transcoders with their supported formats.

**Table 7-1** *List of transcoders in the integration part.*

Transcoder	Supported Source Format	Supported Destination Format
Transcoder 1	MPEG/Audio (all sampling rates)	MPEG/Audio (all sampling rates)
	MPEG/Video (all sizes)	H.263 (all sizes)
Transcoder 2	MPEG/Audio (all sampling rates)	DVI (all sampling rates)
	MPEG/Video (all sizes)	MPEG/Video (all sizes)
Transcoder 3	MPEG/Audio (all sampling rates)	$\mu$ -Law
	MPEG/Audio	GSM Mono

The main class of the transcoder is `Transcoder`. Like the service broker, the transcoder can be run in GUI mode or text mode only. The following command run the transcoder in GUI mode.

```
java -classpath ".:/scratch/JMF2.1.1/lib/jmf.jar:
    /scratch/jini1_2/lib/jini-core.jar:
    /scratch/jini1_2/lib/jini-ext.jar:
    /home/pranata/Transcoding/classes/"
-Djava.security.policy=file:/scratch/jini1_2/policy/policy.all
-Djava.rmi.server.codebase=http://129.69.209.104:8080/classes/
com.antonypranata.transcoding.Transcoder
```

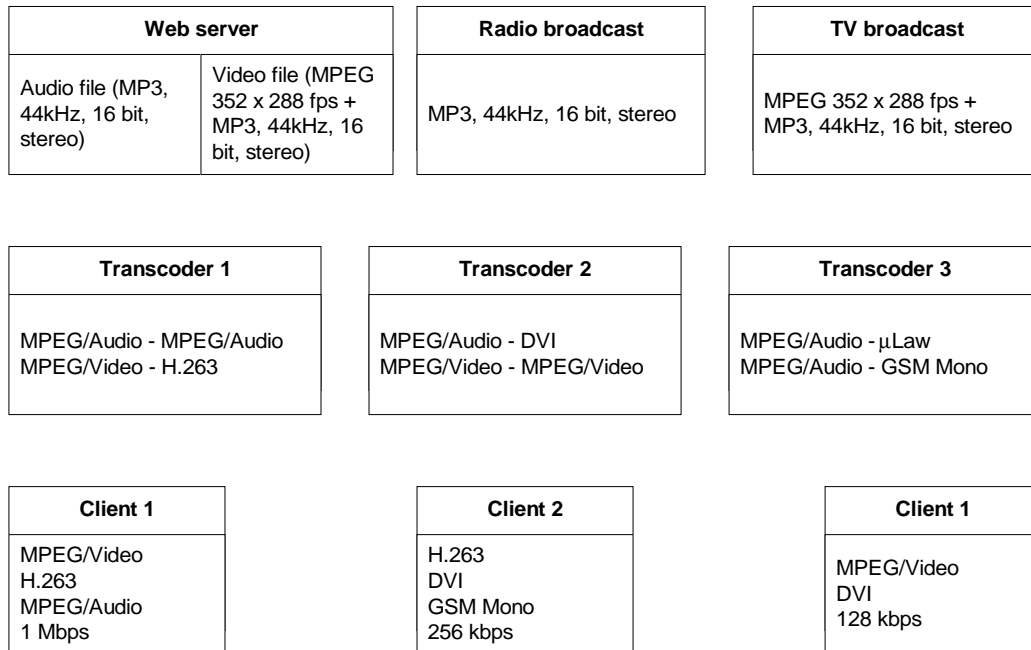
#### 7.1.4.7 Client

There are three clients used in this integration part, each of them has different QoS parameters, i.e:

- Client 1, supports MPEG/Video (all sizes), H.263 (all sizes), and MPEG/Audio (all sampling rates); the available bandwidth was given 1 Mbps.
- Client 2, supports H.263 (all sizes), DVI (all sizes), and GSM Mono; the available bandwidth was given 256 kbps.
- Client 3, supports MPEG/Video (all sizes) and DVI (all sampling rates); the available bandwidth was given 128 kbps.

## 7.2 Testing

In this step, we had already had media files stored in a Web server in the tuba machine as well as TV and radio broadcast from the posaune machine. The situation can be illustrated in Figure 7-2.



**Figure 7-2** *The configuration of testing purpose.*

In the first test, client 1 requested media stream from the address <http://tuba:8080/media/richard.mp3>. The service broker, after performing service brokering algorithm, selected transcoder 1 and the format MP3, 44 kHz, 16 bit, stereo. The service broker selected MP3, 44 kHz, 16 bit, stereo because this format has the highest priority according to the priority table (see Table 5-1). Besides that, transcoder 1 did not serve any clients yet at that time so it has enough resources to serve the client.

The required time to setup the connection was 6 seconds. It took quite long because this was the first request so it means either the client and the service broker did not have any information about the lookup service yet.

When the available bandwidth of client 1 was decreased to 100 kbps, the service broker still selected transcoder 1 but with the format MP3, 44 kHz, 16 bit mono.

The required time to setup the connection was around 1 second. In this request, it seems that the client had already had the cache of the service broker so that it did not need to resend request to the lookup service. The same thing happens in the service broker, it had already had the cache of the transcoder so that it did not need to resend request to the lookup service. That is why the setup connection was about 1 second only.

When the available bandwidth of client 1 was decreased again to 50 kbps, the service broker still selected transcoder 1 but with the format MP3, 22 kHz, 16 bit, mono. The required time to setup the connection was around 1 second.

In the second test, client 2 requested stream from TV broadcast from the address 239.0.0.6/22020. The service broker then selected transcoder 2 with the format H.263 176 x 144 pixels for video stream and DVI 11 kbps, 4 bit, mono for audio stream. This format was selected because it has the highest priority when it is compared to the client's preferences (see also priority table in Table 5-3).

The time to setup the connection was 2 seconds. The time required to setup this connection was less than the first test because the service broker had already had the cache of transcoders in the network so that it did not need to resend request to the lookup service.

When the available bandwidth of client 2 was decreased to 100 kbps, the service broker selected transcoder 2 with the format H.263 128 x 96 pixels for video stream and DVI 8 kbps, 4 bit, mono for audio stream. The time to setup the connection was 1 second.

Next the available bandwidth of client 2 was decreased to 30 kbps, the service broker reported that it could not find the transcoder. This is because no video stream can be delivered in 30 kbps bandwidth (see Table 5-3).

With unchanged bandwidth, client 2 then requested stream from radio broadcast. The service broker selected transcoder 3 with the format GSM Mono.

In the last test, client 3 request video stream from the address <http://tuba:8080/media/starwars.mp3>. The service broker selected transcoder 2 and the format MPEG 128 x 96 pixels for video stream and DVI 8 kHz, 4 bit, mono for audio stream.

### 7.3 Summary

This chapter discusses the integration of the implementation program in Chapter 6. The integration uses computers in the lab of Computer Science department, University of Stuttgart. This chapter also describes some tests performed on the infrastructure integrated in the integration part.

# Chapter 8

## Related Works

This chapter discusses some related works as well as the contribution of this thesis to the community of distributed multimedia systems.

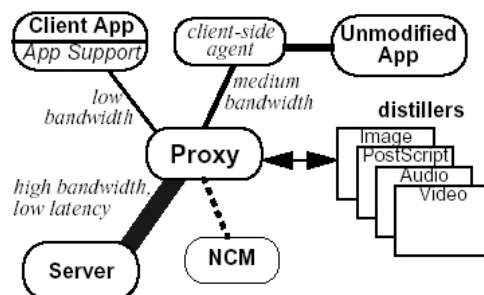
### 8.1 Proxy-based Transcoding

Fox, et. al. [8] has proposed a proxy-based transcoding infrastructure using a principal which they call *datatype-specific lossy compression* or *on-demand distillation*. The purpose is to increase Quality of Service for the client and to reduce end-to-end latency perceived by the client. The distillation or *refinement* uses an intelligent decision to throw away information based on the *semantic type* of the data. For example, distillation of video might include reduction of color information, high-frequency components, pixel resolution, and/or frame rate.

The infrastructure proposed by Fox, et. al. is able to adapt web pages and their contents, including images and video streams. They developed a distiller, which contains image distiller, rich-text distiller, and video distiller, on a proxy server. The proxy itself might be put in the Internet Service Provider connection point or wireless basestation.

#### 8.1.1 Architecture

The architecture of proxy-based transcoding infrastructure proposed by Fox, et. al. is shown in Figure 8-1.



**Figure 8-1** Basic architecture of proxy-based transcoding (courtesy of A. Fox, et.al., 1996).

The components of the infrastructure are *proxy*, one or more *datatype-specific distiller*, an optional *network connection monitor*, and the *application support library*.

#### 8.1.1.1 Proxy Control Point

A client communicates exclusively with the proxy, a controller process located logically between the client and the server. The task of the proxy is to retrieve content from the server on behalf of the client and then determine which distillation engine must be employed. When the proxy calls a distiller, it passes information such as the hardware characteristics of the client, acceptable encoding, and available network bandwidth.

#### 8.1.1.2 Datatype-Specific Distillers

The distillers are processes that are controlled by proxies and perform distillation on behalf of one or more clients. The distiller performs distillation to the data, either text, images or videos, along three important dimensions, i.e.:

- Network variations, include bandwidth, latency and error behavior of the network.
- Hardware variations, include screen size and resolution, gray or grayscale bit depth, memory and CPU power.
- Software variations, include the application-level encoding that a client can handle, for example MPEG or H.263.

#### 8.1.1.3 Network Connection Monitor

A Network Connection Monitor (NCM) which monitors end-to-end bandwidth and connectivity to the proxy's client. NCM uses three methods of determining the characteristics of the client's network connection, i.e.:

- *User advice*. The user notifies the proxy via a user interface his expected bandwidth.
- *Network profile*. NCM uses the average characteristics of the network.
- *Automatic*. NCM creates a process to track the values of effective bandwidth, roundtrip latency, and probability of packet error.

#### 8.1.1.4 Client-side Architecture

The architecture supports both modified and unmodified client applications. The modified applications make use of *application support library* that provides an API with suitable abstractions for manipulating data and interacting with the proxy. Unmodified legacy applications can take advantages of the architecture with the help of a *client-side agent*. The client-side agent is a process that runs locally on the client device.

### 8.1.2 Contribution of This Thesis

A proxy-based transcoding is good enough to solve heterogeneity problem. However it has two main disadvantages, i.e.:



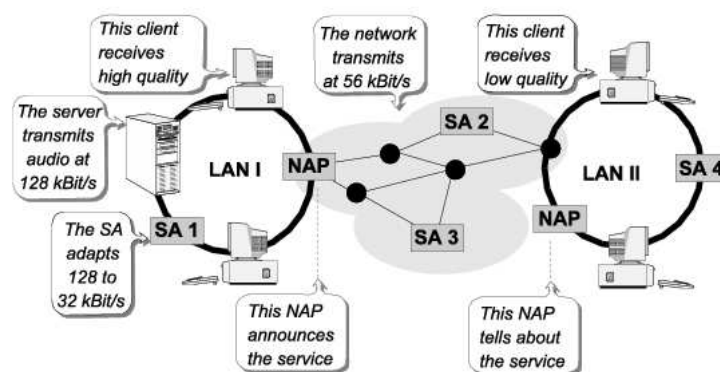
- *Scalability.* The client of proxy-based transcoding depends on one proxy to transcode the stream. If the number of users is growing, the proxy might become overloaded. A. Fox et. al. also mentioned this problem and they have simulated an image-distiller on a single 80-MHz HP PA-RISC workstation. The reasonable number of users for this simulation is around 20 users. If the number of users is more than 24, the system is even unusable.
- *Single Point of Failure.* The proxy-based transcoding has a single point of failure because if the proxy crashes, all clients are not be able to receive anything.

The architecture of this thesis eliminates two main disadvantages of the proxy-based transcoding. Firstly, it solves scalability problem by introducing several transcoders on the network. Secondly, it eliminates single-point-of-failure because each component might be duplicated, including the lookup service and the service broker.

## 8.2 KISS Project

K. Jonas, et. al. on the KISS project [18] [19] proposed communication structure for streaming services in a heterogeneous network that allows transparent integration of network service applications. Network services may be concatenated so that the content streams may experience several transitions on their way through the network, in order to achieve a requested QoS.

K. Jonas, et. al. designed a network with several Service Applications (SA) and Network Access Points (NAP). Figure 8-2 shows the architecture of network infrastructure in KISS project.



**Figure 8-2** Architecture of network infrastructure of KISS project (courtesy of K. Jonas, et. al., 1998).

The mediator between end-system applications and the service network is the NAP. The SA connects to the NAP and offer its service. Client applications connect to the NAP and request services. The NAP handles services announcement and delivery. If the service requirement does not match the

announced service, the NAP tries to find a service application in the network which adapts the offer to the requirement.

The server and the client only know NAP, they do not know SA at all. The server announce its service via NAP and the client request for a service via NAP. Each time a client requests for a stream, it sends the request to the NAP and then NAP finds for an appropriate SA using multicast messages. For example a server provides a service (a live audio stream) with a data rate of 128 kbps. The server announces its service via NAP N1. The announcement itself might look like "CNN Live 4 Mbps, MPEG-2". A client connected to NAP N2 requests a service with a maximum data rate of 64 kbps because of its limited bandwidth. The request itself might look like "CNN Live 64 kbps H.263". The NAP N2 now sends a multicast message into the network asking for the transcoder which is able to transcode 4 Mbps to 64 kbps stream.

The advantage of the KISS infrastructure is that none of the user applications are involved in any of the networking/multicasting/transcoding issues. This approach allows simple end-system applications to obtain added service transparently from SAs installed somewhere in the network and without knowledge of SA existence. Another advantage is that it provides a method for increasing the variety of network services on demand. New services can be installed and used without any end-system modification.

### 8.2.1 Contribution of This Thesis

The infrastructure proposed in this thesis is quite similar to the KISS project from K. Jonas, et. al. The SA in the KISS project is equivalent to the transcoder in my thesis. The NAP in the KISS project is very similar to the service broker in my thesis. However, there are some differences between both architectures, i.e.:

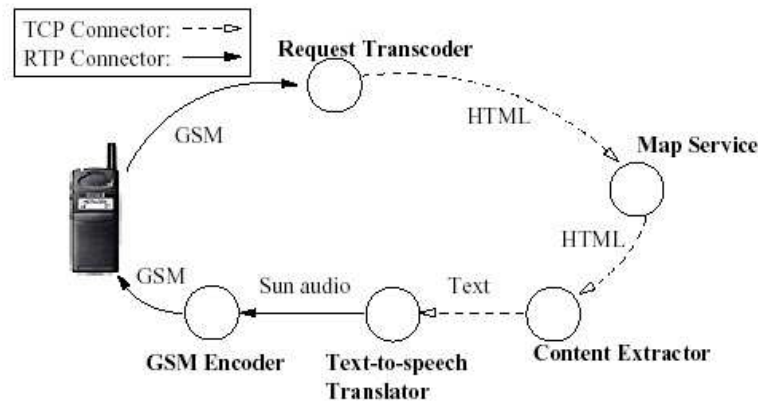
- The KISS project uses multicast messages to find SA dynamically, while my thesis uses a directory service of transcoder, called lookup service. The approach of this thesis eliminates multicast messages but requires the transcoder to register with the lookup service.
- This thesis uses a priority-based service brokering, it means the service broker has a list of possible destination formats and searches the transcoder starting from the highest priority to the lowest one. The KISS project uses only one destination format and increases the TTL of the multicast messages when NAP cannot find the appropriate SA.

## 8.3 ICEBERG Project

The main goal of the ICEBERG project [21] [33] is to develop an Internet-based integration of telephony and data services spanning diverse access networks. The ICEBERG project defines service portability, that is the ability to access services using any devices, anywhere, continuously with mobility support and dynamic adaptation to resource variations. A middleware service

infrastructure, called APC (*Automatic Path Creation*), allows services to be accessed transparently from any device and any network.

Figure 8-3 shows one scenario of the use of the APC.



**Figure 8-3** One scenario of APC in the ICEBERG project (courtesy of Mao and Ratz, 2000).

In this scenario, a user is retrieving map information using a GSM phone. The APC establishes the path by converting HTML format from the map service to the GSM format by going through content extraction, speech synthesizer and GSM encoding operators. APC is completely transparent to the user because it only interacts with the Network Service Provider.

The path construction consists of four steps, i.e.:

- *Logical Path Creation.* A logical path consists of an ordered sequence of operators joined by connectors. The logical path is determined using shortest path search.
- *Physical Path Creation.* A physical path is a logical path, along with a choice of actual nodes (physical machines) on which to run the operators.
- *Path Instantiation, Execution, Maintenance and Querying.* This steps set up the path so that the data flow can be started.
- *Path Tear-Down.* When a path is no longer needed, the user informs APC to stops the data flow, removes connectors, and frees other relevant resources.

The ICEBERG is a very good and ambitious project and the transcoding infrastructure, which they call APC, is only a small part of the overall goal of the project. Actually, the architecture proposed in this thesis uses some principles of the ICEBERG project. For example, the logical path construction of the APC is quite similar to the service brokering algorithm, while physical path creation is quite similar to the service chaining protocol.

### 8.3.1 Contribution of This Thesis

Since the ICEBERG project concentrates on building any-to-any communication, they do not pay attention to the Quality of Service issue. This

result of this thesis, although is not good as the ICEBERG project, but at least it gives a contribution in the QoS issue.

## **8.4 Summary**

This chapter gives an overview and comparison of some related works. There are three related works discussed in this chapter, i.e. proxy-based transcoding, KISS project and ICEBERG project. At the end of each discussion, the contribution of this thesis is also discussed.

## **Chapter 9**

### **Summary and Future Works**

#### **9.1 Summary**

The growth of Internet mobile devices leads to two basic problems in distributed multimedia systems. The first problem is heterogeneity of client devices which have different capabilities along many axes, including network connections. The client may be connected to the Internet via Wireless LAN, such as WaveLAN, or Wireless WAN, such as third generation mobile networks. The second problem is mobility which allows a mobile client to move from one network to another network which might have different bandwidth.

This thesis solve heterogeneity and mobility problems by transcoding media streams to the appropriate format for the client via transcoder. The goal of this thesis is to implement a prototype of network service infrastructure for transcoding multimedia streams. The prototype also includes service brokering, that is the protocol to find the appropriate transcoder, as well as service chaining, that is the algorithm to build service chain from the server to the client via transcoder. The prototype is developed in Java platform using RMI, Jini and JMF technology.

The prototype has been tested in the lab of Computer Science department, University of Stuttgart. It run well without any problems. However, this prototype still needs some other tests in real-world situation.

#### **9.2 Future Works**

The prototype implemented in this thesis still needs many improvement, i.e.

- Most of QoS parameters in this prototype is constant values, they do not represent the actual condition of the system and network. In the future, some measurements of QoS parameters, such as bandwidth, latency and jitter, should be integrated in the prototype.
- This thesis only support 1-level transcoding, it means there is only one transcoder between server and client. It would be better if the

infrastructure support N-level transcoding as well so that we can build more sophisticated infrastructure.

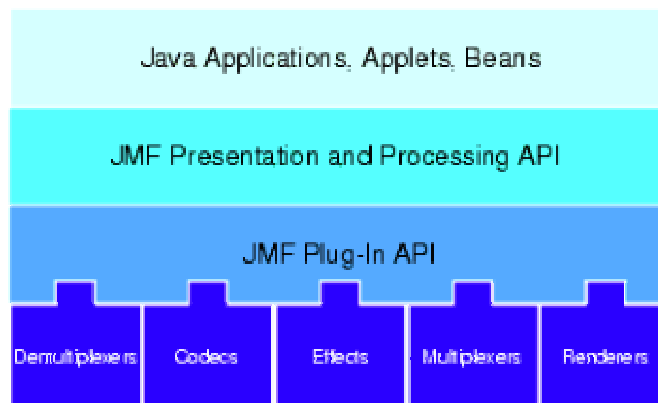
- The prototype only supports Campus LAN with limited number of users. There are many things should be considered so that this prototype can be implemented in the Internet. The most important is scalability.

# Appendix A

## Java Media Framework

This appendix discusses briefly JMF (*Java Media Framework*), a multimedia library for Java from Sun Microsystems, Inc. For more detailed about JMF, see the *Java Media Framework Programmer's Guide* [27].

JMF provides a unified architecture and messaging protocol for managing the acquisition, processing and delivery of time-dependent media data (JMF). JMF is designed to support many well-known formats of media, such as MPEG, H.263, DVI, GSM, and many more. This thesis uses JMF to send and receive media stream to and from each components in the transcoding infrastructure. Figure A-1 shows the high-level architecture of JMF.

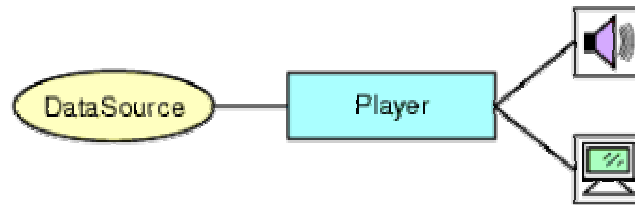


**Figure A-1** *High-level architecture of JMF (courtesy of Sun Microsystems, Inc., 1999).*

JMF uses the basic model very similar to devices such as tape decks or VCRs. When we play a movie using VCS, we provide the media stream to the VCR by inserting a video tape. A data source in JMF acts like the video tape, it encapsulates the media stream. Playing and capturing audio and video with JMF requires the appropriate input and output devices, such as microphones, cameras, speakers and monitors.

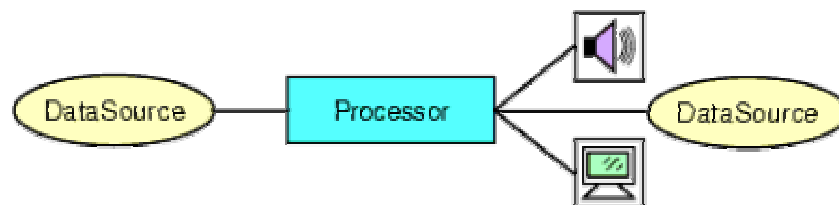
The basic class needed in JMF is `Player` class, which processes an input stream of media data and renders it at a precise time. Figure A-2 shows the JMF

Player model. A `DataSource` class here is used to deliver the input stream to the Player.



**Figure A-2** *JMF Player model (courtesy of Sun Microsystems, Inc., 1999).*

Another basic class is `Processor`, which is a specialized type of `Player` that provides control over what processing is performed on the input stream. In addition, a `Processor` can output media data through a `DataSource` so that it can be presented by another `Player` or `Processor`, further manipulated by another `Processor`, or delivered to some other destination, such as a file. Figure A-3 shows the JMF `Processor` model.

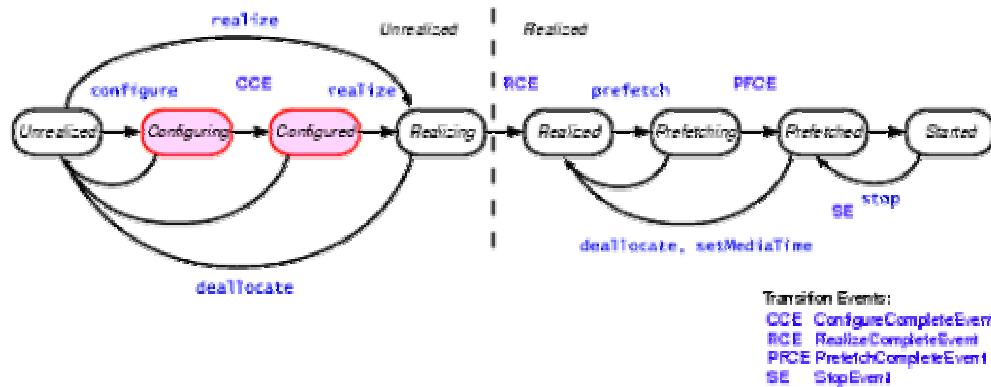


**Figure A-3** *JMF Processor model (courtesy of Sun Microsystems, Inc., 1999)*

Figure A-4 shows the stages of `Processor`. As shown in this picture, the processing of media data is split into several stages:

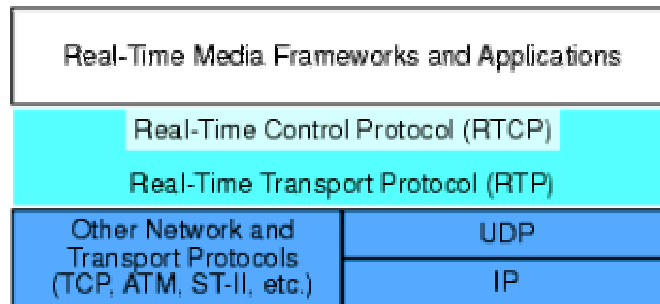
- *Demultiplexing* is the process of parsing the input stream, and it is extracted if the input stream contains multiple tracks.
- *Pre-processing* is the process of applying effect algorithms to the extracted tracks.
- *Transcoding* is the process of converting each track from one format to another. It is actually the main topic of this thesis.
- *Post-processing* is the process of applying effect algorithms to decoded tracks.
- *Multiplexing* is the process of interleaving the transcoded media tracks into a single output stream.
- *Rendering* is the process of presenting the media to the user.





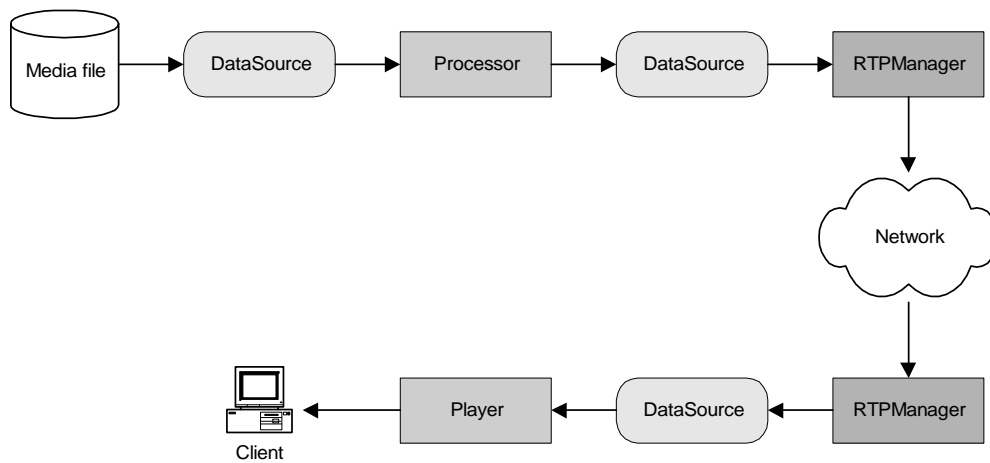
**Figure A-4** Stages in Processor (courtesy of Sun Microsystems, Inc., 1999)

What is used extensively in this thesis from JMF is the JMF RTP API. It allows the playback and transmission of RTP streams. Figure A-5 shows the high-level architecture of JMF RTP API.



**Figure A-5** High-level architecture of JMF RTP API (courtesy of Sun Microsystems, Inc., 1999).

The Player and Processor model of JMF RTP API is very similar to Figure A-3 and Figure A-2, but JMF RTP API introduces a new class, called RTPManager. RTPManager is the starting point for creating, maintaining and closing an RTP session. The tasks of RTPManager includes to keep track of the session participant and the streams that are being transmitted, to maintain the state of the session as viewed from the local participant, to handle the RTCP control channel and support RTCP for both senders and receivers. Figure A-6 shows the model of JMF RTP API from the sender and receiver view.



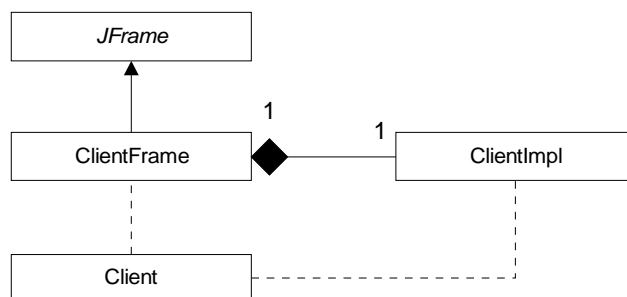
**Figure A-6** *The model of JMF RTP API from sender and receiver perspective (courtesy of Sun Microsystems, Inc., 1999).*

## Appendix B

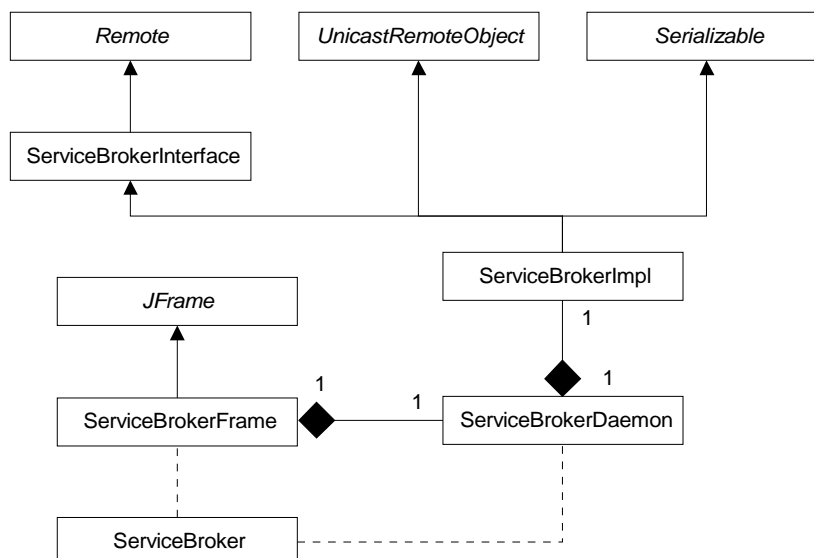
### Class Hierarchy

This appendix presents hierarchy of the classes developed in this thesis.

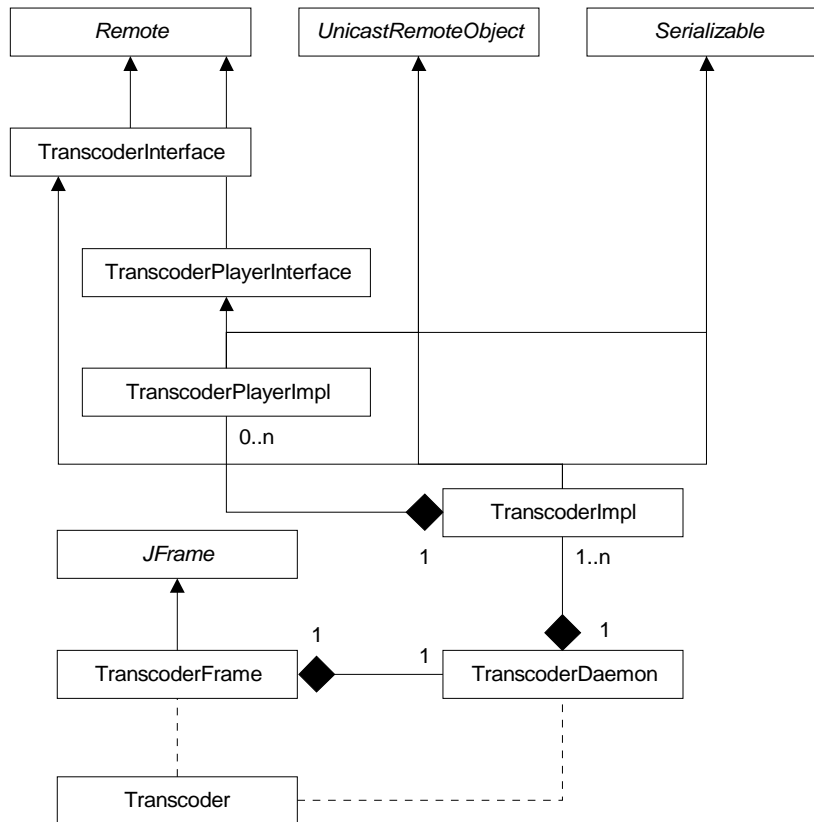
#### Client



#### Service Broker



## Transcoder



## References

- [1] A. V. Aho, J. E. Hopcroft and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, USA, 1987.
- [2] H. Bharadvaj, A. Joshi and S. Auephanwiriyakul. *An Active Transcoding Proxy to Support Mobile Web Access*. Proc. IEEE Symposium on Reliable Distributed Systems, 1998.
- [3] D. Chalmers and M. Sloman. *Survey of Quality of Service in Mobile Computing Environments*. Department of Computing, Imperial College, London, 1999.
- [4] D. Chen, R. Colwell, H. Gelman, P. K. Chrysanthis and D. Mossé. *A Framework for Experimenting with QoS for Multimedia Services*. University of Pittsburgh, Pittsburgh, PA, USA, 1996.
- [5] COMCAR Project. *COMCAR – Communication and Mobility by Cellular Advanced Radio*. <http://www.comcar.de/overview.pdf>, 1999.
- [6] G. Coulouris, J. Dollimore and T. Kindberg. *Distributed Systems Concepts and Design*. Pearson Education Limited, Essex, UK, 2001.
- [7] S. E. Czerwinski, et. al. *An Architecture for a Secure Service Discovery Service*. Mobicom'99, Seattle, WA, USA, 1999.
- [8] A. Fox, S. D. Gribble, E. A. Brewer, E. Amir. *Adapting to Network and Client Variability via On-Demand Dynamic Distillation*. Proc. Seventh International Conference on ASPLOS, 1996.
- [9] Guojun Lu. *Communication and Computing for Distributed Multimedia Systems*. Artech House Inc., Norwood, MA, USA, 1996.
- [10] IETF. *DoD Standard Transmission Control Protocol*. RFC 761, <http://www.ietf.org/rfc/rfc0761.txt>, January 1980.
- [11] IETF. *User Datagram Protocol*. RFC 768, <http://www.ietf.org/rfc/rfc0768.txt>, August 1980.
- [12] IETF. *Internet Protocol Darpa Internet Diagram Protocol Specification*. RFC 791, <http://www.ietf.org/rfc/rfc0791.txt>, September 1981.
- [13] IETF. *Internet Stream Protocol Version 2 (ST2)*. RFC 1819, <http://www.ietf.org/rfc/rfc1819.txt>, August 1995.

- [14] IETF. *RTP: A Transport Protocol for Real-Time Applications*. RFC 1889, <http://www.ietf.org/rfc/rfc1889.txt>, January 1996.
- [15] IETF. *HyperText Transport Protocol – HTTP/1.1*. RFC 2068, <http://www.ietf.org/rfc/rfc2068.txt>, January 1997.
- [16] IETF. *Real-Time Streaming Protocol*. RFC 2326, <http://www.ietf.org/rfc/rfc2326.txt>, April 1998.
- [17] Van Jacobson. *PathChar - A Tool to Infer Characteristics of Internet Paths*. Network Research Group, Lawrence Berkeley National Laboratory, Berkeley, CA, 1997.
- [18] K. Jonas. *Forget the Net! Architecture, Specification and Implementation of a Communication System for Real-time Streaming in Heterogeneous Environments*. 1997 Pacific Workshop on Distributed Multimedia Systems, Vancouver, Canada, July 1997.
- [19] K. Jonas, M. Kretschmer and J. J. Mödeker. *Get a KISS - Communication Infrastructure for Streaming Services in a Heterogeneous Environment*. ACM Multimedia 98, Bristol, UK, September 1998.
- [20] E. Kovacs, R. Keller, T. Lohmar and A. Held. *Adaptive Mobile Applications over Cellular Advanced Radio*. Personal Indoor and Mobile Radio Communications (PIMRC). London, UK, September 2000.
- [21] Z. M. Mao and R. Katz. *Achieving Service Portability in ICEBERG*. CS Division, EECS Department, University of California at Berkeley, California, USA, 2000.
- [22] Microsoft Corporation. *Understanding Universal Plug and Play White Paper*. [http://www.upnp.org/download/UPNP\\_UnderstandingUPNP.doc](http://www.upnp.org/download/UPNP_UnderstandingUPNP.doc), 2000.
- [23] Object Management Group (OMG). *CORBA Basics*. <http://www.omg.org/gettingstarted/corbafaq.htm>, 2001.
- [24] K. Rothermel. *Lecture Note: Introduction to Distributed Systems*. Institute of Parallel and Distributed High-Performance Systems, University of Stuttgart, Germany, 2001.
- [25] John R. Smith, Rakesh Mohan and Chung Sheng Li. *Transcoding Internet Content for Heterogeneous Client Devices*. Proc. IEEE International Conference On Circuits and Systems (ISCAS), May 1998.
- [26] R. Steinmetz and Klara Nahrstedt. *Multimedia: Computing, Communications and Applications*. Prentice Hall Inc., NJ, USA, 1995.
- [27] Sun Microsystems Inc. *Java Media Framework Programmer's Guide*. <http://java.sun.com/products/java-media/jmf/2.1.1/guide/>, 1999.
- [28] Sun Microsystems, Inc. *Java Remote Method Invocation Specification*. <http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmiTOC.html>, 1999.

- [29] Sun Microsystems Inc. *Jini Specifications v1.2*.  
<http://www.sun.com/jini/specs/>, 2001.
- [30] Sun Microsystems Inc. *Jini Network Technology Datasheet*.  
<http://wwwswest.sun.com/jini/whitepapers/>, 2001.
- [31] A. S. Tanenbaum. *Computer Networks 3<sup>rd</sup> Edition*. Prentice Hall Inc., NJ, USA, 1996.
- [32] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall Inc., NJ, USA, 2002.
- [33] H. J. Wang, et. al. *ICEBERG: An Internet-core Network Architecture for Integrated Communications*. IEEE Personal Communications (2000): Special Issue on IP-based Mobile Telecommunication Networks, 2000.
- [34] World Wide Web Consortium (W3C). *Simple Object Access Protocol (SOAP) 1.1*. <http://www.w3.org/TR/SOAP/>, 2000.
- [35] L. C. Wolf, C. Griwodz and R. Steinmetz. *Multimedia Communication*. Proceedings of the IEEE, Vol. 85, No. 12, December 1997, pp. 1915 – 1933.