An Investigation into Tools and Protocols For Commercial Audio Web-site Creation

S'busiso Simon Ndinga

An Investigation into Tools and Protocols For Commercial Audio Web-site Creation

Submitted in fulfillment of the requirements of

Master of Science

By

S'busiso Simon Ndinga

Computer Science Department
Rhodes University
Grahamstown 6140
South Africa

January 2000

Abstract

This thesis presents a feasibility study of a Web-based digital music library and purchasing system. It investigates the current status of the enabling technologies for developing such a system. An analysis of various Internet audio codecs, streaming audio protocols, Internet credit card payment security methods, and ways for accessing remote Web databases is presented. The objective of the analysis is to determine the viability and the economic benefits of using these technologies when developing systems that facilitate music distribution over the Internet. A prototype of a distributed digital music library and purchasing system named WAPS (for Web-based Audio Purchasing System) was developed and implemented in the Java programming language. In this thesis both the physical and the logical component elements of WAPS are explored in depth so as to provide an insight into the inherent problems of creating such a system, as well as the overriding benefits derived from the creation of such a system.

Acknowledgements

I would like to thank Prof. Richard Foss, my supervisor, for his continued effort and support over the duration of my degree. Many thanks also go to the members of the Rhodes University Computer Science Department for their support.

I would also like to thank Prof. Radlof, for her support during the statistical analysis of the subjective audio listening tests. Thanks to Prof. Pat Terry, Vinothan Naidoo and Siviwe Kwatsha for proofreading the drafts of this thesis.

Special thanks goes to my mother, my sister Duduzile, and my brothers, for their ceaseless encouragement throughout my masters study.

Contents

Chapter 1

Introduction

The growth of the WWW has provided new opportunities for distributing and selling recorded music over the Internet. Nowadays consumers have the ability to tune-in to their favorite sites, and to listen to music or purchase it while online. Online stores can be referred to as the new or current generation way of doing business. In the context of business transactions, online stores use the Internet to facilitate the sale and purchase of goods and services. The Web-based Audio Purchasing System (WAPS) developed during this project is a typical example of an online store that provides the sale of African music on the Internet. WAPS is a Web-based application that lets users search for music of their choice from a music database, provides short audio excerpts from its collection on demand, and allows for music to be ordered securely and delivered in digital form over the Internet.

The development of this system has been supported by the International Library of African Music (ILAM), an institute of Rhodes University. ILAM have a large archive of African music that they have collected over many years. This music has been recorded on old 78 rpm LP's (10 and 12 inches), ¼ Magnetic tapes (of various sizes, including 'pancakes'), Shellac discs, Acetates, Cassettes, and recently CD's. ILAM has been distributing this music to their customers on audio tapes using the postal services. Due to the high demand for this music from their customers, ILAM has had to consider other delivery mechanisms, of which they chose the Internet. Thus, a Web-based audio delivery system was needed. This then lead to the development of WAPS. The major aim of this project has been to investigate the current technological requirements for setting-up an ideal Web-based audio purchasing environment.

1.1 Design Challenges

Putting together an enterprise-level application for an Internet music store involves design and integration of various related technologies that play specific roles in a distributed computing environment. For example, providing widely distributed client

access to a large music database and sale of musical assets poses several challenges that are inadequately addressed by several electronic commerce applications. Firstly, a distributed topology is a prerequisite for building applications of this sort, since the Internet is inherently distributed in nature. Thus, a distributed architecture is the foundation for these applications. Secondly, designing such applications involves choosing between technologies based on feasibility, applicability, availability, cost, and many other factors. However, the challenge is to investigate relevant technologies and select competing, yet complementary technologies that can be integrated efficiently and effectively to produce a robust system. Thirdly, online commerce applications generally need to cater to both the product distributor's (merchant's) needs as well as the user's needs. This is one important factor that in most instances is ignored. For example a typical e-commerce application should be easy to use for both users and merchants, easy to maintain and upgrade, fast, and should consume few machine resources.

The following is a summary of the design challenges that are to be faced when creating commercial audio web sites:

- *Provide flexible online access to a large music database.*

  Providing networked access to a large music library presents several challenges. First, one has to deal with size, since the demands of digitized audio and the high bandwidth requirements of audio playback require high capacity storage devices. Also, the limited inter-network bandwidth and the desire for low latency access suggest the use of distributed stores.

- *Support ad-hoc queries for allowing users to find music material based on bibliographic information.*

  Indexing and retrieval of multimedia data poses many different demanding requirements for navigation on database systems. For example, multimedia databases need to support searches by media content, such as audio clips, in addition to processing the traditional text-based queries. Hence, the system should not be limited to one media retrieval method, but should be able to support both current and future technology. There is a lot of research which is taking place into

the field of content- based audio retrieval systems, although no firm standards have emerged as yet.

- *Provide support for the standard and the most popular audio codecs.*

Web shoppers are accustomed to sampling items they want to buy in the form of images. Similarly users of the system will prefer to pre-listen to a shorter version of a particular music item before purchasing it. Hence, the system needs to provide support for most of the standard and the popular audio file formats.

- *Provide a way to deliver audio according to the users preference.*

The system has to allow users to download music material online. However, many users of the Internet are still connected to the net with a bit rate of 33.6 kbps, and hence other options for delivery (such as e-mail attachments) will need to be provided.

- *Provide a secure online payment mechanism through the use of the standard and the most popular Internet security protocols.*

Internet payment security is still a major bottleneck that slows down the growth of Internet commerce. Internet shoppers need a payment system that is easy to use, fast and trustworthy.

- *Support the standard multimedia transport protocols so as to allow users to preview short audio excerpts on demand.*

Multimedia transport protocols allow audio to be streamed over the Internet in real time. Streaming audio is an ideal way to preview music on the Internet, with no download time or disk space issues to worry about.

The six central goals presented above have provided a platform for the analysis of various audio compression schemes suitable for the Internet, audio and data transport protocols, and secure purchase protocols. Realizing the above requirements, an efficient and flexible architecture for WAPS had to be developed.

1.2 What is New in this Project?

This project aims to provide a complete picture of secure music distribution over the Internet, not just specific pieces (e.g. electronic payments), specific scenarios (e.g. delivering high quality audio online), or specific products and protocols. The relationship of the selected enabling technologies is explained in Chapters 2, 3, and 5 through the three logical components of WAPS, outlined in section 1.3.3 below. WAPS presents an open structure for electronic commerce. This includes a technical architecture also outlined in section 1.3.3 as well as a legal aspect described in Chapter 6. The technical structure enables the integration of various tools and protocols that provide the necessary services. Thus, WAPS is not restricted to specific proprietary technology or specific protocols. The prototype implementation uses existing technology standards and widely used technologies, such as data transport protocols used to access databases remotely, streaming audio technology, credit card payments, public-key certification, and cryptographic services.

1.3 A Model for an Online Music Store

There are a number of steps that need to be taken when setting up an online store for the sale of digital music on the Internet. Firstly, the music to be sold has to be captured from its original source, digitized, and cleaned. Secondly, a short audio clip of each musical item to be sold has to be created. This is to allow users to preview music before purchasing it. Thirdly, each musical item has to be catalogued along with its audio clip, using industry-standard SQL databases such as those from Microsoft, Informix, and Oracle. Finally, the online store, which takes the form of a distributed application, has to be built for the purchase and sale of the musical items.

1.3.1 Digitizing Audio

There are two major operations that need to be effected when digitizing audio: firstly, the audio should be captured from its original source at such a level so that clipping does not occur. Secondly, clicks, hisses, ticks, and pops of modest amplitude have to be removed. There is a wide variety of audio digitizing software packages on the market that can be used to fulfill the above operations. Some of these packages are platform-specific, while others are cross-platform. Listed below are a few of these software packages: CubaseVST, WaveLab, CoolEdit, GoldWave, Sound Forge, and

many others. All these packages offer excellent tools for capturing, re-sampling, mixing, splitting tracks, and similar manipulations. For this project, WaveLab was chosen for digitizing ILAM's music material since it integrated with a number of powerful sound processing plug-ins.

1.3.1.1 Digitizing with WaveLab

WaveLab is an all-in-one sound editing and recording software package. It allows for recording and editing, correcting mistakes and making detailed analysies giving quality results in 24-bit, 96 kHz resolution. For filtering out noise, WaveLab uses a Plug-In called DeNoiser. The DeNoiser is a real-time broadband noise removal tool. It is based on spectral subtraction, whereby each frequency spectrum that has amplitude below the estimated noise floor is reduced in intensity by the use of a spectral expander. This results in noise reduction that does not affect the signal phase. Figure 1 below illustrates the signal flow in the DeNoiser Plug-In.

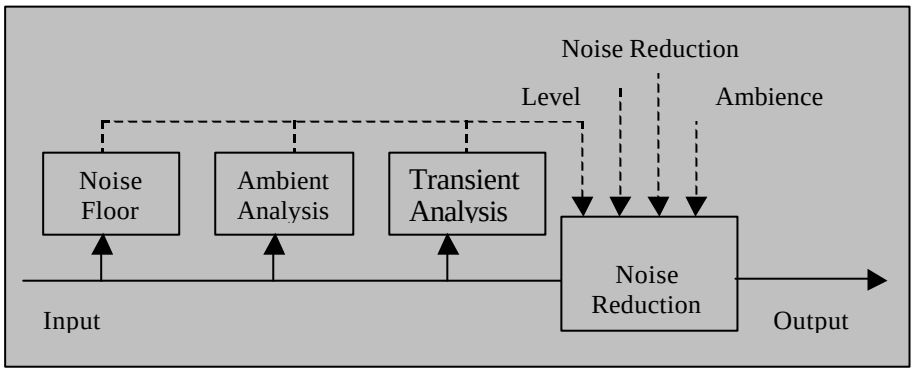

Figure 1: Signal flow in DeNoiser

When the noise level is constant or modulates slowly, the signal is continuously analyzed by the first module in the chain to estimate the noise floor at any given time. However, if the noise level varies rapidly, the Ambient and Transient analysis help adjust the response of the noise reduction unit. This allows transient material to maintain its liveliness and natural ambience.

1.3.2 Encoding and Cataloguing

The process of encoding and cataloguing of the music items on sale was done by ILAM. The encoders used and the cataloguing process are described in section 4.4 of Chapter 4.

1.3.3 The System Overview

The model chosen for WAPS is shown in Figure 2. It integrates four Internet related technologies, which are: multimedia databases, suitable audio codecs, data and audio transport protocols and Internet security protocols. For any musical store based on the web, the above technologies need to be addressed during the design and implementation of such a system.

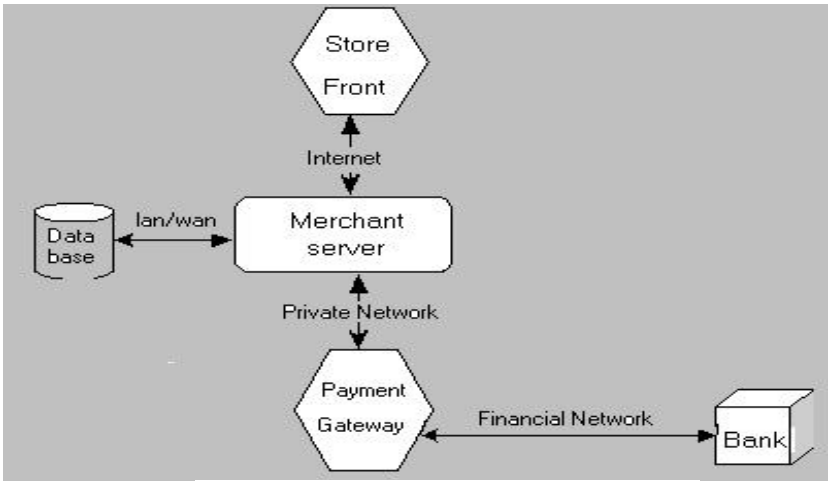

Figure 2: A model for a web music store

In designing the architecture of WAPS, the primary concern was its modularity. A modular system structure enables quick and easy modification and improvements of some parts without affecting others. Based exclusively on the core modules, it has to be possible to add new modules, utilizing the existing system mechanisms. WAPS modularity covers both the physical system components and the logical layered structure. Physically, WAPS consists of three primary components: Audio Storage Server, Store Front, and the Merchant Server.

1.3.3.1 The Audio Storage Server

The Audio Storage Server stores bibliographic information regarding a particular music item for searching and retrieval purposes. Since one of the system's objectives is to provide access to a large volume of audio information, the Audio Storage Server has to be capable of handling numerous requests simultaneously. It also has to be capable of supporting any other multimedia storage system with an ODBC structure. The choice of the Audio Storage System used in WAPS is described in section 2.3 of Chapter 3.

1.3.3.2 The Store Front

This forms the client side of WAPS. It is a graphical user interface that resides on any Java-capable browser on the user's machine. The Store Front is composed of three components, which are the information retrieval engine, the audio player, and the shopping cart.

- **The Information Retrieval Engine**

  The Information Retrieval Engine is used primarily for conducting structured and free text searching, menu browsing, and displaying of results.

- **The Audio Player**

  This is a remote continuous audio playback application that has to be used in conjunction with a web browser to access remotely stored continuous audio data. It provides support for most of the popular audio codecs as well as transmission protocols that are suitable for the Internet. It integrates with various audio servers to facilitate real-time streaming of audio data.

- **The Shopping Cart**

  The shopping cart is needed to temporarily hold all the purchased items, while the user is busy shopping. It is also used to allow secure transfer of customer payment details from the Store Front to the Merchant Server.

1.3.3.3 The Merchant Server

The Merchant Server forms the server and the processing end of WAPS. It resides on the merchant's computer. The merchant server is composed of three components, which are the data access server, the audio server, and the payment server.

- **The Data Access Server**

  The Data Access Server processes audio queries from the remote client (Store Front) and communicates with the Audio Storage Server to enable users of WAPS to extract audio information of interest.

- **The Audio Server**

  The Audio Server uses suitable multimedia transport protocols to transmit on-demand locally stored short audio excerpts to the audio player. Before the Audio Server transmits audio data to the player, an appropriate transmission protocol is negotiated between the Audio Server and the player, depending on the current traffic on the network.

- **The Payment Server**

  This forms an interface between the shopping cart and the payment gateway of the Acquiring bank (Merchant's bank). It supports two third-party payment systems that enable credit card authorization and settlement services. These are the Virtual Vendor (VV) [10] system, an e-mail-based system; and the BankGate [11] system. The BankGate system offers immediate authorization, while the VV system does not.

Figure 3 illustrates the physical structure of the system's components.

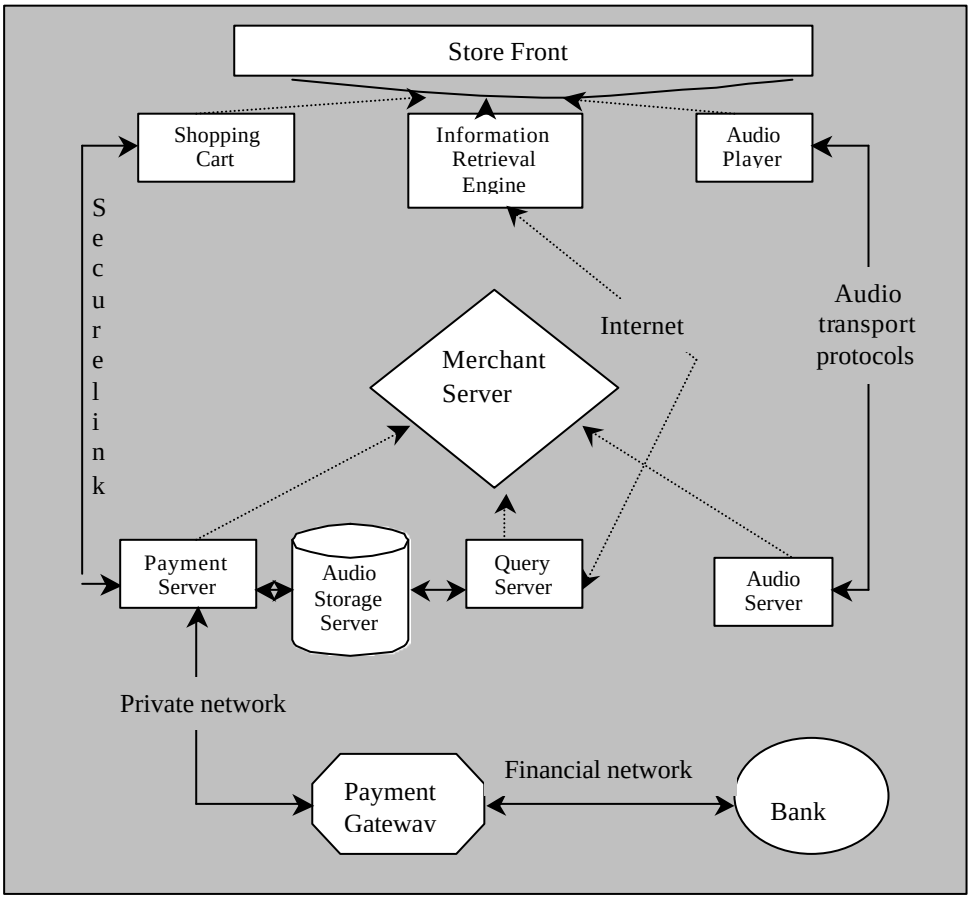

Figure 3: The WAPS physical component structure

Although the system is split into physical components, it is also divided into three logical communication components responsible for providing various functionality. Usually, a physical component implements one logical layer. However, one-to-many relations between a component and layers are possible. The three logical components implemented by WAPS are:

- **Data Communication Component**

  The data communication component provides the means for transporting information from the Audio Storage Server to the Store Front. The implementation details for this component are described in Chapter 2.

- **Audio Communication Component**

  This provides the means for the real-time transmission of audio clips from the audio server to the audio player. The audio codecs and the transport protocols that are utilized in this component and the details of how these interact are described in Chapters 3 and 4.

- **Payment Communication Component**

  This component details how WAPS secures the user's payment details when making online transactions. The implementation details of the payment communication component are described in Chapter 5.

1.4 Thesis Overview

The remainder of this thesis is structured as follows:

- Chapter 2 (Remote Access to Audio Databases) investigates the different techniques available for accessing remote databases and provides some background information on related technologies, such as current database technology, audio database retrieval techniques, and data transport protocols. This chapter also describes the overall architecture of the logical data communication component of WAPS, and shows how it was implemented using the concepts presented in this chapter.

- Chapters 3 (Internet Audio Codecs) presents various technologies and standards that are currently used by applications utilizing audio on the Internet. It discusses the basics of digital audio and offers a look at the current audio compression schemes that are suitable for use over the Internet. This chapter also discusses the procedure followed and the results of the subjective audio listening tests conducted on various audio codecs for bit rates between 6 and 56 kilobits per second.

- Chapter 4 (Streaming Audio Protocols) discusses some of the standard and widely used multimedia transport protocols, so as to give an overview of the means by which multimedia data is transmitted over the Internet. The technical issues that affect streaming audio over the Internet and attempts to solve them are discussed. This chapter lays out the common architecture used by Internet streaming applications and shows how this architecture was implemented in WAPS, thereby forming the WAPS logical audio communication component. Chapter 4 also discusses the results of the qualitative tests for various audio transport protocols that are utilized by WAPS.

- Chapter 5 (Internet Credit Card Transactions) discusses the security of credit card payment on the Internet. The general approach on how credit card transactions occur on both the Internet and the real world is outlined. Several payment security mechanisms are discussed, including the use of SSL, SET, and proprietary systems that are based on these two protocols. This chapter also discusses the overall payment mechanism implemented in WAPS (referred to as the WAPS logical payment communication component).

- Chapter 6 (Distributing Music over the Internet) discusses some of the legal issues and copyright concerns encountered when distributing high quality audio over the Internet. It introduces the issues of copyright protection of multimedia content on the Internet and presents several technological efforts that attempt to solve the problems of Intellectual Property Management and Protection of multimedia content.

- Chapter 7 (Conclusion) presents the concluding remarks on WAPS and the cost, feasibility, and the applicability of implementing a Web-based audio purchasing system.

Chapter 2

Remote Access to an Audio Database

This chapter will investigate the various techniques available for accessing a remote database. It presents general web database architectures and shows how they were implemented in WAPS. An analysis of the different communication mechanisms that can be used in distributed web database applications is presented. The analysis of these technologies will focus on the solutions that directly affect the development of the WAPS data communication component. Hence, the analysis of these different communication mechanisms will be based on approaches that are provided by the Java programming language. However, the results also apply to the more general field of distributed computing. Firstly, a brief overview of the current database technologies, multimedia retrieval techniques, and the general architectures used when developing web databases will be given. Thereafter, the various communication techniques that can be used between a client and a server application in a distributed environment, and the solutions selected for WAPS will be introduced. Finally, the chapter gives some experimental results on the performance of the implemented communication mechanisms.

2.1 Current Database Technology

There are three models that are available for handling multimedia data, and these are:

a) Object Relational Model

This model started when multimedia data first began to be included in databases. With this model, a multimedia object is represented as a binary large object (BLOB) [1]. A BLOB is a collection of binary data stored as a single entity in a database management system. BLOBs are used primarily to hold multimedia objects such as images, videos, sound, and they can even be used to store programs or fragments of code. BLOBs can be indexed by name or subject and this leads to them functioning as a black box i.e. one cannot get inside and ask questions about the contents of the object. For example, one cannot search an

audio file or get content information about a specific sequence of frames of a video file. This model uses SQL for data definition, data management, and data access and retrieval.

b) Object-oriented Database Model

Object-oriented databases employ a data model that has object-oriented aspects like a class with attributes, methods, and integrity constraints. They combine the aspects of object orientation and object-oriented programming languages with database capabilities. Object-oriented databases have the ability to deal with complex objects. They encapsulate the object along with the methods it needs for access and display. They provide more than persistent storage of programming language objects by extending the functionality of object programming languages to provide full-featured database programming capabilities. The object-oriented database model uses a declarative language called OQL (Object Query Language) defined by ODMG-93 [2] for querying of database objects. However, most object-oriented databases also support SQL, primarily in the context of ODBC.

c) Object Relational Database Model

The object relational database model employs a data model that attempts to add object-orientedness to database tables. All database information is still in tables, but some of the tabular entries can have richer data structures termed Abstract Data Types (ADTs). An ADT is a data type that is constructed by combining basic alphanumeric data types. The support for ADTs is attractive because the operations and functions associated with the new data type can be used to index, store, and retrieve records based on the context (e.g. multimedia) data type. This model supports an extended form of SQL, sometimes referred to as Object SQL, for querying databases. The point of the extension is to support the object model (e.g. queries involving object attributes). The typical extensions include queries involving nested objects, set-valued attributes, inclusion of methods, functions in search predicates, and queries involving abstract data types. This model is still relational because the data is stored in tables of rows and columns, and SQL (with the extensions mentioned) is the language for data definition, manipulation, and query.

2.2 Audio Database Retrieval Methods

Selecting a (small) subset of objects (or records) that meet certain specified criteria from a given large set of objects is a central problem in database technology. Two familiar methods, for achieving this are: the traditional alphanumeric method and content-based retrieval methods.

- **The Traditional Alphanumeric Method**

This is the most common way of retrieving multimedia data in a database. It is accomplished by assigning an alphanumeric 'tag' to each object and then retrieving the objects based on their tags. These tags can be structured into multiple attributes with an object itself being treated as merely one very large attribute from a logical perspective. The traditional Boolean algebra then becomes applicable in all the attributes in the tag and predicates on these can be used to retrieve the object. In most cases, the tag does not have as much structure as typical relational databases. For example, the tag may comprise English descriptions of what the object is about. Hence, standard keyword based indexing used in document retrieval can be used to retrieve the tags relevant to a given query and the associated data.

The problem with this approach is that the data in the object is not given the consideration it deserves. Hence, this gives rise to other problems. The first problem is how to select what to put in a tag, given an object. Multimedia objects, such as audio, have many different attributes that are of interest and it is not easy for any single tag to capture all of them. One solution could be to identify the commonly used characteristics and build tags based on these, though some information would still be lost. The other problem is how to create tags. Does one still have to look at the multimedia object and fill in the corresponding tag for each multimedia object in a database? The cost of such tagging may limit the potential size of a database. Also, part of the tagging process can be automated so as to reduce human effort. For example, in an audio database, a cataloguer could enter information such as the author, instruments, publisher, music type, performer, etc, while the tagging of the type-specific attributes such as compression type, sample size, rate, etc could be automated.

- **Content- based Retrieval Technique**

This method can be viewed as the solution to the traditional alphanumeric method. In this technique, the query is actually posed against the object rather than against the tag. The content of the object is analyzed and used to evaluate specified selection predicates. Retrieval in multimedia databases must allow for the type of queries known from the field of traditional databases as well as for extended functionality (such as full text search) known from the field of information retrieval [3]. For example, in the case of videos, content-based searching means the ability to search for a specific fragment of a video that starts with a given scene, or that includes a given object. In the case of audio, one might want to be able to retrieve all audio that is associated with a given topic.

Content-based retrieval may depend on the availability of rich metadata or meta-knowledge about the original multimedia data. The metadata might be based on additional knowledge incorporating the semantics of the data and its intended usage in a particular application, or it might be automatically derived from the original data by employing specific analysis techniques. One obvious concern with respect to the retrieval by content as described in the preceding paragraph is performance. Complex predicates can be very costly to evaluate, and performing such an evaluation on every object in a database may not be feasible. This is particularly applicable to large multimedia databases in which a database server has to process requests from several clients simultaneously. So, access structures can be provided to reduce this cost and this means that the data may have to reside in distributed storage.

2.3 WAPS Audio Storage Server

The digitizing, the cleaning, and the cataloguing of music material for WAPS was done at the ILAM labs by ILAM. The choice of a database management system was also made by ILAM, and in this case ILAM chose to use the Microsoft Access database as their database server. MS Access falls under the object relational database model. It provides a technology called OLE (Object Linking and Embedding) [4] for storing audio, images, and other binary data created outside it. With OLE technology, an object can either be linked to, or embedded into a table. Linked objects are those

that reside outside the database, can be updated independently of the database, and are managed by a file system, not the database. On the other hand, embedded objects reside completely within the database, are updated by first launching the database, and travel within the database as opposed to within the file system. Since MS Access does not support the retrieval of audio material by content, the traditional alphanumeric technique for the retrieval of the music material on the database was used. Here, the music data was stored as a string containing the filename (i.e. point/link) of the music files stored on the local hard disk. Another reason why the link method was chosen over the embedded data method was to save disk space. With the embedded method one copy of the audio data resides inside the database and the other remains in the file system. An update to the embedded audio data does not affect the copy on the file system and this affects data consistency. For example, since two copies of the multimedia object exist i.e. one embedded in the database and the other remaining in the file system, when a change is made to the object on the file system, it is not reflected on the embedded object. Hence, the updated object on the file system will now be different from the embedded object, though they both were the same object.

2.4 Architecture of Web Databases

Web database applications have a client-server structure. There are two architectures that are commonly used when developing these types of applications, namely two-tier architecture and three-tier-architecture (sometime this is referred to as the n-tier architecture).

2.4.1 Two-tier Approach

This is the most common architecture on LAN based systems. In two-tier client/server systems, the client talks directly to a server with no other intervening processes. With this approach, clients manage the user interface, validate data entered by the user, dispatch requests from the clients, execute business logic, and send data to users [5]. Figure 1 illustrates the structure of the two-tier approach.

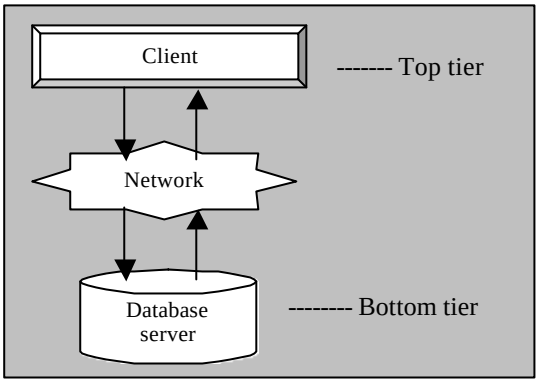

Figure 1: Two-tier approach

2.4.2 Three-tier Approach

The three-tier client/server architecture introduces a third layer of processing between the client and the server (Figure 2). As can be seen in Figure 2, this architecture is composed of three tiers:

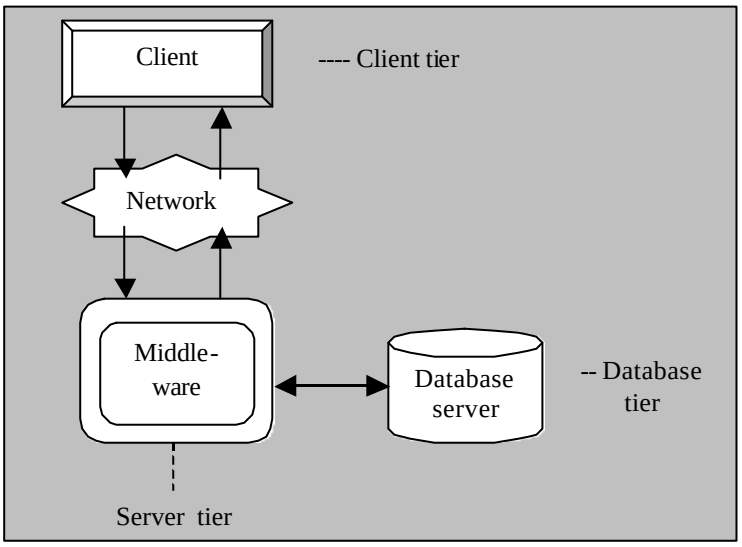

Figure 2: Three-tier approach

a) Client-tier (top-tier): is responsible for the presentation of data, receiving user events, and controlling the user interface. The actual application logic has been moved to the application server-tier.

b) Application server-tier (middleware-tier): protects data from direct access by clients. This tier is new and is not present in the two-tier architecture approach in this explicit form.

c) Database-tier (bottom tier): responsible for data storage.

The boundaries between tiers are logical. For example, it is easily possible to run all three tiers on one machine.

2.5 Architecture for the WAPS Logical Data Component

As mentioned in section 1.3.3 of Chapter 1, the WAPS data component describes the way in which the archival audio catalog information travels from the audio storage server to the Information Retrieval Engine of the Store Front. The data component is composed of three of the WAPS physical components, which include the Information Retrieval Engine, Data Access Server, and the Audio Storage Server, The three-tier architecture approach was chosen when designing the data component. The major advantage of the three-tier architecture over the two-tier architecture is that it allows clients and servers to lose weight and become "thin clients" and "thin servers". This means that partitioning of functions can be carried further and greater modularity can be achieved. The following are some advantages that justify the choice of a three-tier architecture over a two-tier architecture:

- Reduced network load.

  With the two-tier approach, the actual processing of data takes place on the remote client and this data has to be transported over the network. Hence increasing the network load. The introduction of the application server tier in the three-tier approach removes this complexity since the actual control processing is moved to this tier and clients only receive the results of the queries made to the database server, or any other operation defined by the application server-tier.

- Separation of user interface control and data presentation from application logic.

  Through this separation, more clients are able to have access to a wide variety of server applications.

- Re-definition of the storage strategy does not have much impact on the clients.

From Figure 2 above, the chosen three-tier architecture for the data component integrates the Information Retrieval Engine (Client-tier), the Data Access Server (Server-tier), and the Audio Storage Server (Database-tier). The interaction between the server-tier and the data storage-tier can generally be managed by database protocols such as ODBC or the Java Database Connectivity (JDBC). However one major concern is how to manage the interaction between the client-tier and the server-tier in a distributed environment. This section discusses different communication methods that can be used between the client and the server tiers. The focus will be on solutions that directly or indirectly affect the development of WAPS and Web distributed applications in general. Firstly, the JDBC approach that can be used to facilitate communication between the server-tier and the database-tier will be discussed followed by a discussion of the various communication methods that can be used between the client and the server tiers.

2.5.1 Communication between the server and the database tiers

2.5.1.1 The concept of JDBC

The Java Database Connectivity is a standard Java API for executing SQL statements [6]. It consists of a set of classes and interfaces written in the Java programming language. JDBC is a low-level interface that is used to invoke or call SQL commands directly. It was designed to be a base upon which to build higher-level interfaces and tools. It provides a uniform and a standard programming level interface for database access. Just like the Open Database Connectivity (ODBC), JDBC is based on the X/OPEN SQL Call Level Interface. JDBC can be used to write programs to send SQL statements to virtually any relational database (Figure 3).

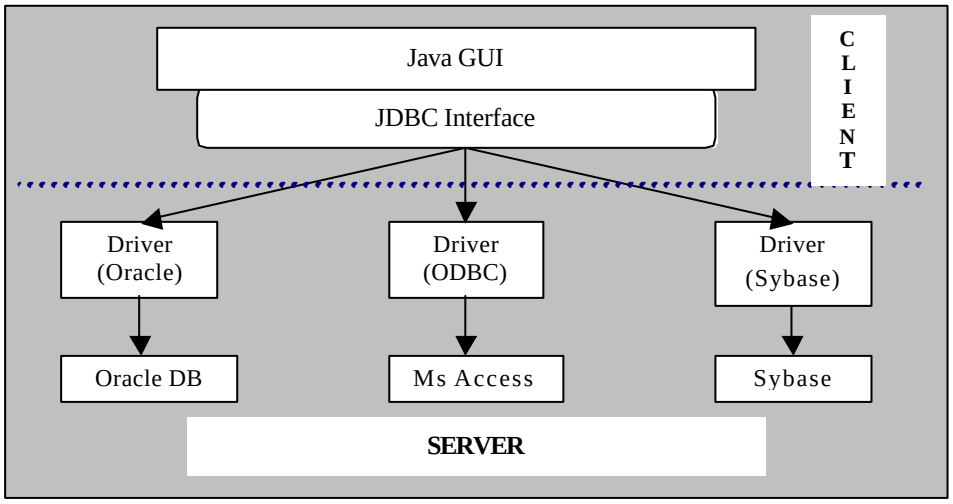

Figure 3: JDBC's relational DB support

JDBC allows application programmers to do three things:

a) Establish a connection with a relational database,

b) Send SQL statements, and

c) Process the results.

The following code fragment gives a basic example of these steps:

```
Connection con = DriverMananger.getConnection ( “jbdc:odbc:Access”, “Login”  “Password” )
Statements stmt = con.createStatement ();
ResultsSet rs = stmt.executeQuery(“SELECT a,b,c FROM  Table1”));
    while(rs.next())
     {
       int x = rs.getINT(“a”);
       String s = rs.getString(“b”);
       Double c = rs.getDouble(“c”);
     }
```

JBDC facilitates both the two-tier and the three-tier client/server communication models for accessing databases. In a two-tier model, a client application talks directly to the database. This requires a driver that can communicate with the particular database management system being accessed. The SQL statements are then sent to the database and the results sent back to the client application. Figure 4 illustrates the use of JBDC in a two-tier architecture.

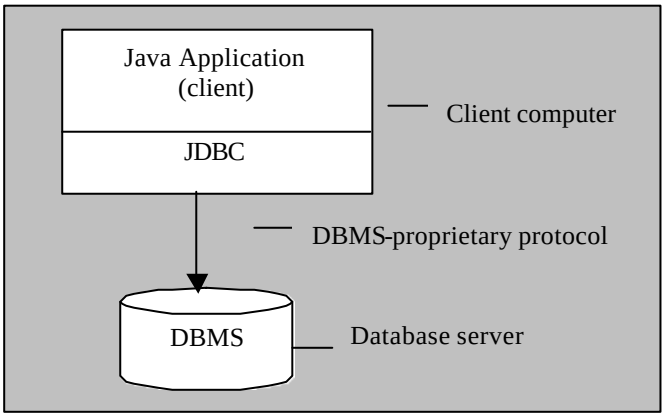

Figure 4: JDBC in a 2-tier architecture

In the three-tier architecture, commands from the client are sent to a “middleware-tier”, which then sends SQL statements to the database. The database processes the SQL statements and sends the results back to the middleware-tier, which in turn sends them to the client application. In this approach, the JDBC is implemented in the middleware–tier as opposed to the client application. Figure 5 shows how JDBC is implemented in a three-tier approach.

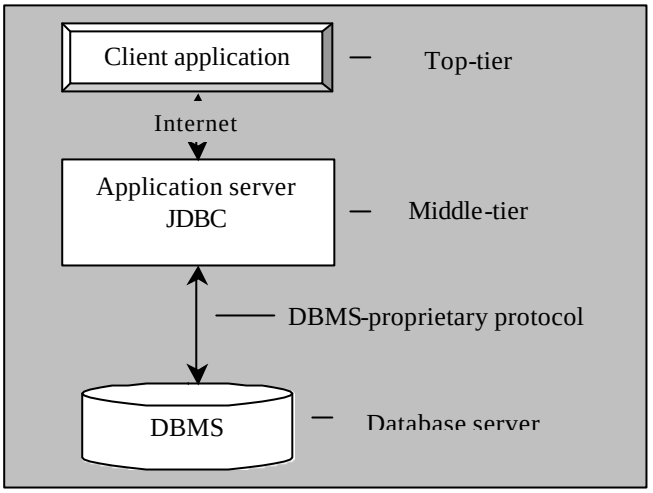

Figure 5: JDBC in a 3-tier model

2.5.2 Communication between the Client and the Server tiers

Four approaches based on Java were identified for communication between the client and server tiers in a distributed environment. These included HTTP, Java sockets classes, Java Remote Method Invocation (RMI), and the Common Object Request Broker Architecture (CORBA).

2.5.2.1 HTTP/CGI

The Hyper Text Transfer Protocol (HTTP) allows web browsers to communicate with HTTP servers. This communication protocol is implemented over TCP/IP. HTTP operates in a disconnected mode i.e. once the transaction is completed the client will disconnect itself from the server. Apart from serving simple web pages, HTTP combined with the Common Gateway Interface (CGI) can be used in developing distributed web applications. Figure 6 illustrates the architecture of such applications.

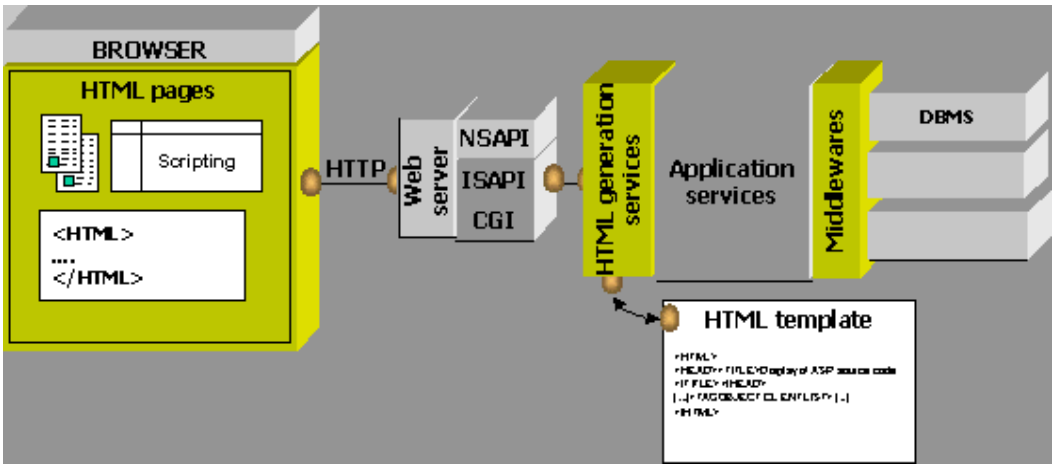

Figure 6: HTTP/CGI application architecture

The operation of such applications would be as follows [11]:

1) The user clicks on the "submit" button of the HTML form; the browser uses the POST method provided by HTML, specifying the target URL and the HTTP headers. The HTTP server receives the call via a TCP/IP connection, analyzes the message, and recognizes POST.

2) The web server launches a CGI program or transmits the request to an application server via an API, such as the Java Servelet [7].

3) The CGI program or the application server performs the processing (for instance the connection to a DBMS).

4) The CGI program or the application server generates the resulting HTML page with the results.

5) Finally, the HTTP server returns the results to the web browser for the user viewing.

This method is fairly easy to implement, but it has a number of significant drawbacks. These include: data marshalling, performance, and maintainability.

- **Data Marshalling**

With the HTTP approach it becomes the application programmer's responsibility to marshal and un-marshal a request, its arguments, and results, into a string representation according to the URL format for the CGI. The marshalling process is carried-out by the application server or the CGI script.

- **Performance**

Experiments have shown that the CGI creates a significance performance overhead. The CGI creates an extra layer of computation and (inter-process) communication.

- **Maintainability**

When the interface of the application evolves to a new version, the application programmer has to upgrade the client, the server, and the CGI script in a consistent manner. This creates unnecessary work and is time consuming; hence it becomes very difficult to maintain.

2.5.2.2 Java Socket Classes

This is the lowest level at which a client and a server application can communicate in a Java distributed system environment. It is the basis through which all the other techniques are built. With the Java programming language a client application can open an IP socket connection using either TCP or UDP data transport protocols. Though the socket-based communication can be efficiently implemented between a client and server application, an application programmer has to still be concerned with standard tasks such as data marshalling and session management. Also, support for version upgrades is non-existent. As data formats remain application specific, the re-use of these implementations is limited. Also, Java imposes a network security restriction that applets can only open a network connection to servers that reside on machines where the applets were downloaded. This poses a serious problem, especially when one thinks of migrating the server application to a different location. Hence, this limits the use of this approach on web-based applications.

2.5.2.3 RMI

The Java Remote Method Invocation (RMI) is an API standard for building distributed Java systems [8]. RMI can be compared with the Remote Procedure Call (RPC), which was intended for usage in procedural languages such as C. RPC allows one to write programs that call procedures over a network as if the code was stored and executed locally. RMI can be thought of as the object equivalent of RPC. However, instead of calling procedures over a network as if they were local, RMI invokes objects that are distributed throughout a network and physically residing on the different machines. From the application's point of view, a remote method and a local method are invoked in the same manner using the same semantics, while RMI takes care of the detail at a lower level. RMI is implemented as three layers between the application program and the Java Virtual Machine, which in turn sits on top of the operating system. The three layers are:

1) A stub class on the client side of the client/server relationship, and the corresponding skeleton at the server end. Both the stub and the skeleton classes are comprised of Java code that provides the necessary link between the client and the server.

2) A Remote Reference Layer that can behave differently depending on the parameters passed by the calling program. For example, this layer can determine whether the request is to call a single remote service or multiple remote programs as in multicasting.

3) A Transport Connection Layer, which sets up and manages the request.
   For the actual communication between two objects RMI uses a native protocol called the Remote Method Protocol (RMP). With this protocol, information transfers are performed via "serialization". In other words, the arguments and return objects are transmitted via RMP, which in turn formats them so that they can be transported over the network. There has been an API released by Sun Microsystems that allows RMI over the Internet Inter ORB Protocol (IIOP). This API allows client objects written in other languages to talk to server objects written in Java and vice versa.

RMI uses a network-based *registry* to keep track of the distributed objects. A server object makes a method available for remote invocation by binding it to a name in the RMI registry. The client object in turn, can check for availability of an object by looking up that particular object's name in the RMI registry. Basically, the registry acts as a limited central management point for RMI and it can be thought of simply as a name repository i.e. it keeps track of the names of all objects that have allowed their methods to be invoked. Now, one has to recall that we are dealing with objects, the client and the server that are physically residing on different machines. Hence, a mechanism is needed to transmit the client's request to invoke a method on the server object and provide a response back to the client.

In this regard RMI uses an approach very similar to RPC. Figure 7 illustrates an interaction between the client and the server objects using RMI. When a client invokes a server method, the Java Virtual Machine of the client object refers to the

stub for type checking (1). This request is then routed to the skeleton class on the server, which in turn calls the appropriate method on the server object (2 and 3), that may then send queries and receive results from a database. The stub acts a proxy to the skeleton (4) and the skeleton is a proxy to the actual remote method of the server object (5). In order for RMI to be useful, it must be able to pass arguments to invoked methods and also handle return values from the invoked methods. In RMI there are no restrictions on the types of arguments to be passed, and these involve the normal data types, and objects. Practically, RMI allows two ways for the passing of arguments: by remote object and by value.

The deciding factor, as to which approach to use, is whether or not arguments refer to remote objects. For example, if an argument or return value is a remote object, a reference to the object is passed. Otherwise an argument or return value is not a remote object, it is copied over the network, meaning that the argument is passed by value. In RMI, the process of encoding and decoding arguments is referred to as marshalling and un-marshalling respectively, and it is transparent to the application programmer. RMI' s major drawback is that it is limited to a single language environment, which is Java.

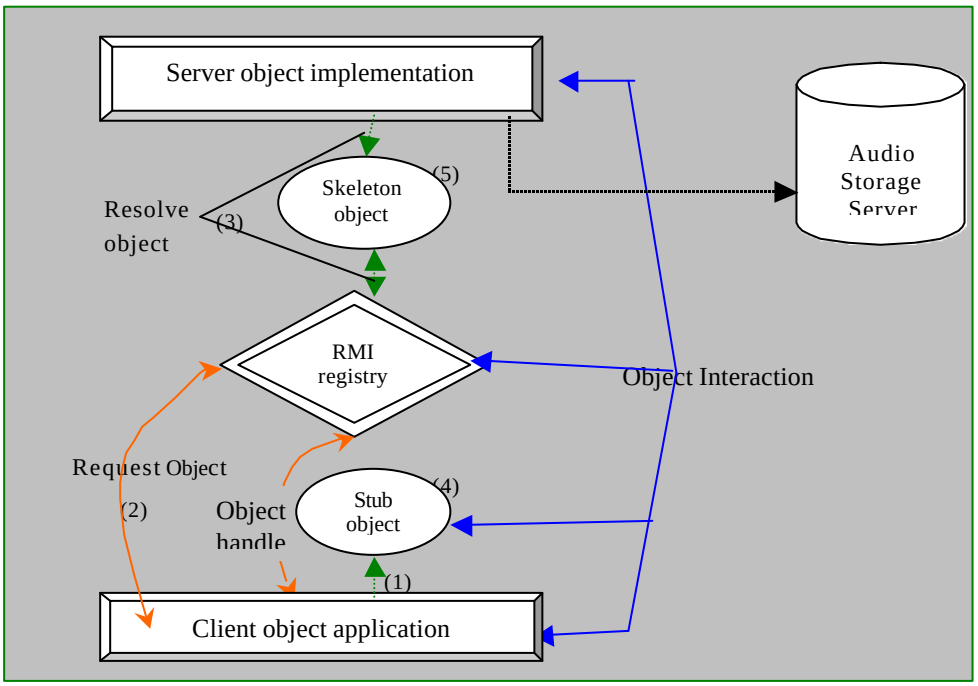

Figure 7: RMI's client/server interaction

2.5.2.4 CORBA

The Common Object Request Broker Architecture (CORBA) is a distributed object standard developed by an industry consortium of 500 companies called the Object Management Group (OMG) [4]. It is a set of specifications for providing interoperability and portability to distributed object-oriented applications. CORBA-compliant applications can communicate with each other regardless of their location, implementation language, and the underlying operating system and hardware architecture. The CORBA framework consists of the following elements [9]:

- An Object Request Broker (ORB)

  The ORB is the object bus. It provides the middleware that establishes the interaction between client applications needing services and server applications capable of providing them.

- An Interface Definition Language (IDL)

  IDL provides architecture and implementation independence to CORBA-compliant applications. It is a declarative language in which object interfaces are defined and advertised. The interface definition itself is independent of the actual object implementation.

- Implementation Repository

  This provides a run-time repository of information about the classes a server supports, the objects that are instantiated, and the object references. The implementation repository consists of all the run-time information about the invocation of methods on the client side.

- Interface Repository

  This is an on-line database of object definitions that can be queried at run-time for dynamic method calls, for locating potentially reusable software components, and for type checking of method signatures. Basically, the interface repository is a run-time distributed database that contains machine-readable versions of the IDL defined interfaces.

- Object Adapter

  The object adapter sits on top of the ORB core communication services and accepts requests on behalf of the server objects. It provides the run-time environment for the server application and handles incoming client calls. It is also responsible for registering the server classes with the implementation repository.

- CORBA Services

  These augment the functionality of the ORB. CORBA defines services for persistence, transactions, concurrency, database queries, licensing, etc.

- Inter-ORB Protocol (IIOP)

  The communication in the CORBA implementation of the data layer relies on IIOP and is performed via stubs and skeletons. IIOP is a binary protocol for communication between ORBs. It specifies how messages and common data type representations can be exchanged over a TCP/IP network.

- Dynamic Invocation Interface (DII)

  This is used for dynamic method invocations. CORBA defines Application Programming Interfaces (API) for looking up the metadata that defines the server interface, generating the parameters, issuing the remote call, and getting back the parameters.

- Dynamic Skeleton Interface (DSI)

  It provides run-time binding for servers that need to handle incoming method calls. The DSI looks at parameter values in an incoming message to figure out the object and its methods. DSI is the server equivalent of the DII.

- Client/Server IDL Stubs

  The stubs provide static interfaces to object services. They are created by using an IDL compiler. The client stubs define how clients invoke corresponding services on the servers. From a client perspective, the client stubs are local proxies for remote server objects. A CORBA client application must have an IDL stub for each interface it uses on the server. On the other

hand, the server stubs provide static interfaces to each object exported by the server.

Figure 8 shows the various components that form CORBA.

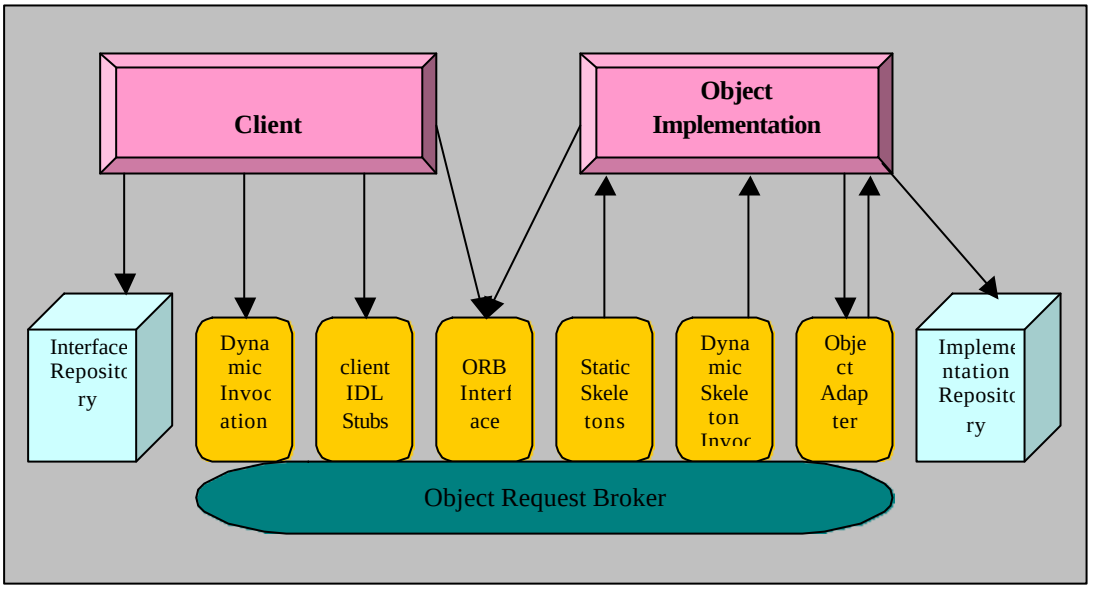

Figure 8: Components of CORBA

When a client sends a request to the server, the ORB intercepts this request and is responsible for finding an object that can implement the request, passing it the parameters, invoking its method, and returning the results. CORBA defines two common ways in which a client can receive an object reference: by using the Interoperable Object References (IOR) or by using naming services. Just like RMI, in CORBA, an object can receive a reference to a server object through a naming service. The naming service allows the client object to locate the server object by name. Another method for a client to receive a reference to an object is by using IORs.

Every object on the ORB has an IOR. The IOR is a global identifier string that identifies the machine on which its associated object is located and the interface that the object supports. If given the IOR for an object, a client can use standard function calls on the ORB to turn it into an object reference. Figure 9 illustrates how a client makes a request to the server object via an ORB.

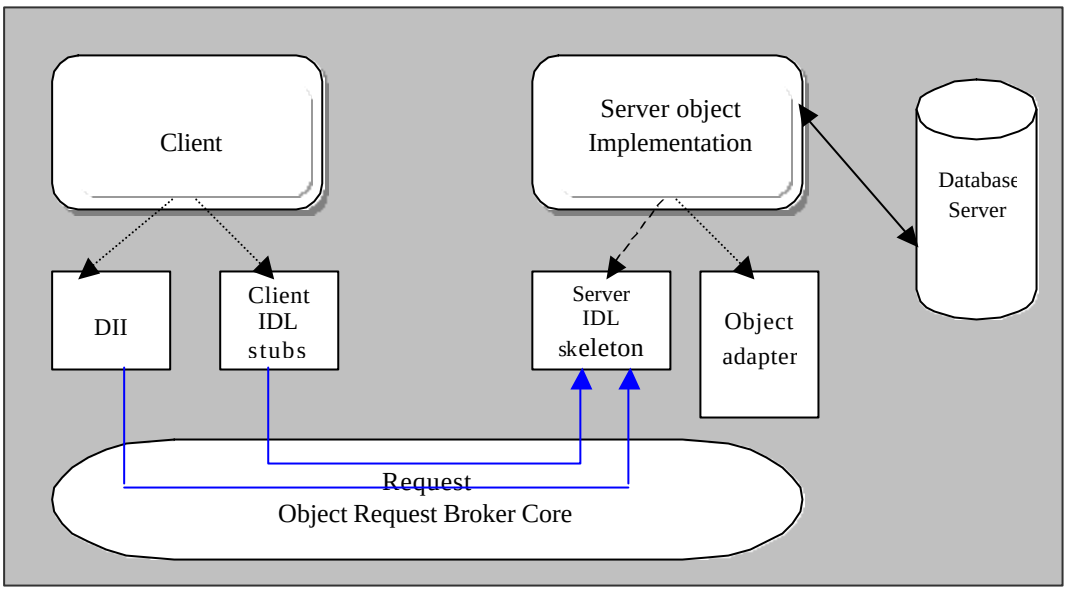

Figure 9: Client/server request

From Figure 9, the client either uses the DII or its stub class to forward a request to the ORB interface. The ORB then intercepts the client's request, finds the object that implements the request through the server object skeleton or the object adapter, passes the parameters to skeleton, which then hands them over to the server object implementation. The server object may then query the database, get the results back and sends them to the ORB interface through its skeleton class. The ORB then forwards the results to the client object through the stub class.

2.6 Implementation of the 3-tier Architecture that forms the WAPS Data Communication Component

As mentioned in section 1.3.3 of Chapter 1, the data communication component defines the way in which the WAPS audio catalogue information travels from the bottom tier (Audio Storage Server), through the middleware tier (Data Access Server) to the client-tier (Information Retrieval Engine). Basically, this logical component integrates three of the WAPS physical components, which are the Information Retrieval Engine, the Data Access Server, and the Audio Storage Server, to form a 3-tier architecture structure. Two similar versions of the data component's three-tier architecture were designed and developed. In one version the Java's Remote Method Invocation was implemented for transporting database information between the client

and the server tiers. The other version implements CORBA between the client and the server tiers.

However, both these versions implement the JDBC approach for transporting the database information between the middleware server and the data storage tiers. Figure 10 shows the overall implemented 3-tier architecture of the data component and it also illustrates how the data communication component's framework tiers relate to each other.

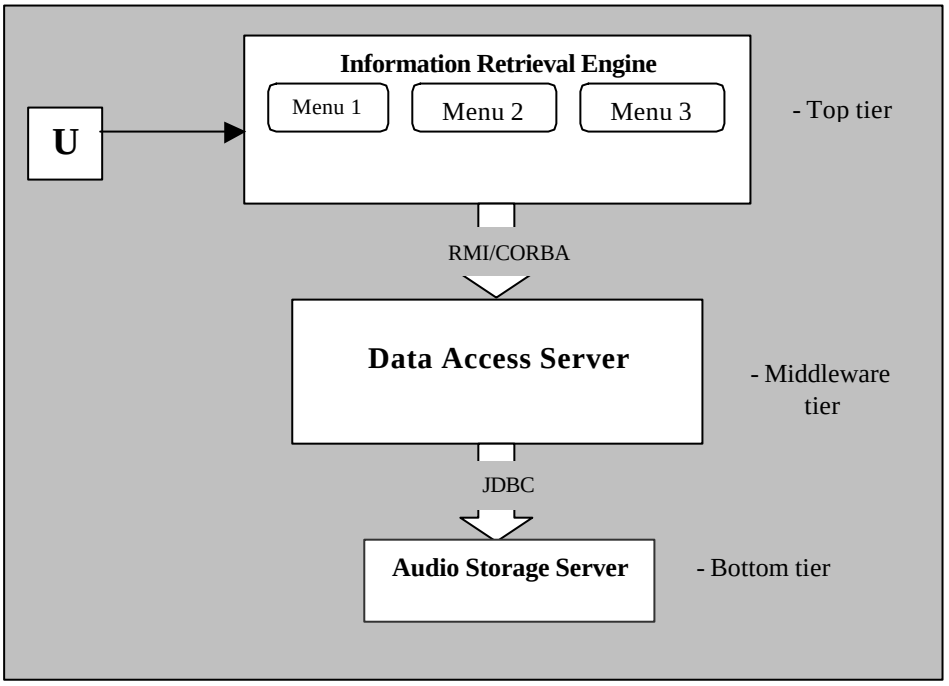

Figure 10: Data component's framework

Before the protocol communication between the client UI-tier, and the data server-tier is defined, the information one would like to pass among these three-tiers will be examined. A typical data retrieval transaction in WAPS would have the following steps.

This transaction does not make use of the shopping cart, payment manager, audio player, audio server, payment gateway, and the bank modules. It is only concerned with how data travels from the client UI, through the data access server, to the data storage server and back to the client UI. Figure 11 gives a diagrammatic flow of this transaction.

- The user feeds his input for a song or an artist request to the client UI (Information Retrieval Engine).

- The request goes to the middleware server (Data Access Server), named server. A method on the client application passes the data request to the server by invoking its method remotely.

- The server packages the data request into an SQL query object, makes a connection to the database (Audio Storage Server), forwards the SQL query for execution, and waits for results.

- The storage server executes the query and sends the results back to the server.

- The server receives the results from the storage server, flattens them into a string representation and stores them into a hash table. It then forwards the hash table to the client.

- The client receives the hash table with the search results, retrieves the results from the hash table, un-flattens them to their original data types, and displays them for user viewing.

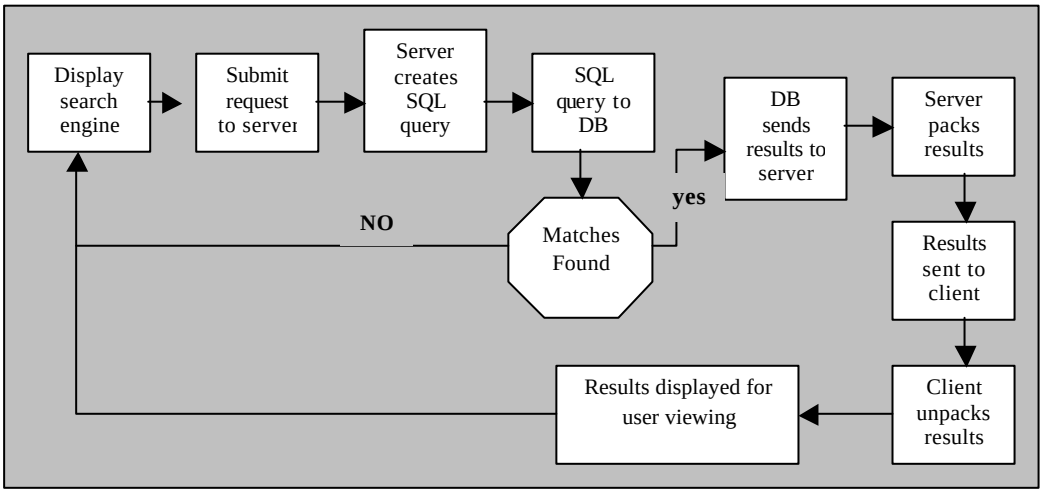

Figure 11: Transaction flow

The following two sections (sections 3.61 and 3.62) discuss the two 3-tier architecture implementation versions of the WAPS logical data communication component, which are based on RMI and CORBA respectively.

2.6.1 The Pure Java with RMI Implementation

Here, RMI was utilized to transfer the audio database information between the client-tier and the server-tier. Figure 12 illustrates the 3-tier RMI architecture approach implemented. The pure Java with RMI implementation of the WAPS data communication component was developed in two parts. The first part is the development of the RMI client (i.e. the client-tier named the Information Retrieval Engine). The second part is the RMI server object (i.e. middleware-tier named the Data Access Server).

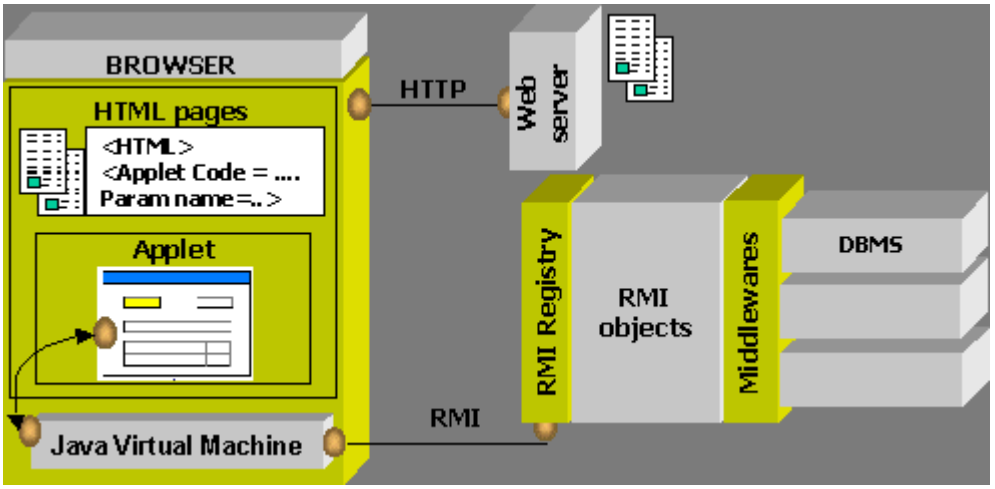

Figure 12: RMI architecture of the data component

The following four classes form the core of the RMI implementation of the data component. They implement the scenario described by the above-mentioned transaction flow represented by Figure 11. The logic in these classes is hard coded. These classes are:

1) RMIRequestClient: Receives a search from the user, forwards it to the RMIAccessServer, and passes the results to the DataReceiver.

2) RMIAcccessServer: Receives the user request in the form of a string, generates an SQL query and forwards it to the DBConnector.

3) DBConnector: Receives SQL query from the RMIAccessServer, connects to the database using the JDBC-OBDC bridge, sends the SQL query for

execution, gets the results back, flattens them into a string representation and stores them into hash table.

4) DataReceiver: Receives flattened results from the RMIRequestClient, reconstructs them into their original form, and displays them for the user viewing.

The RMIRequestClient and the DataReceiver classes are implemented by the Information Retrieval Engine, while the RMIAccessServer and the DBConnector classes are implemented by the Data Access Server. Figure 13 illustrates how these classes interact.

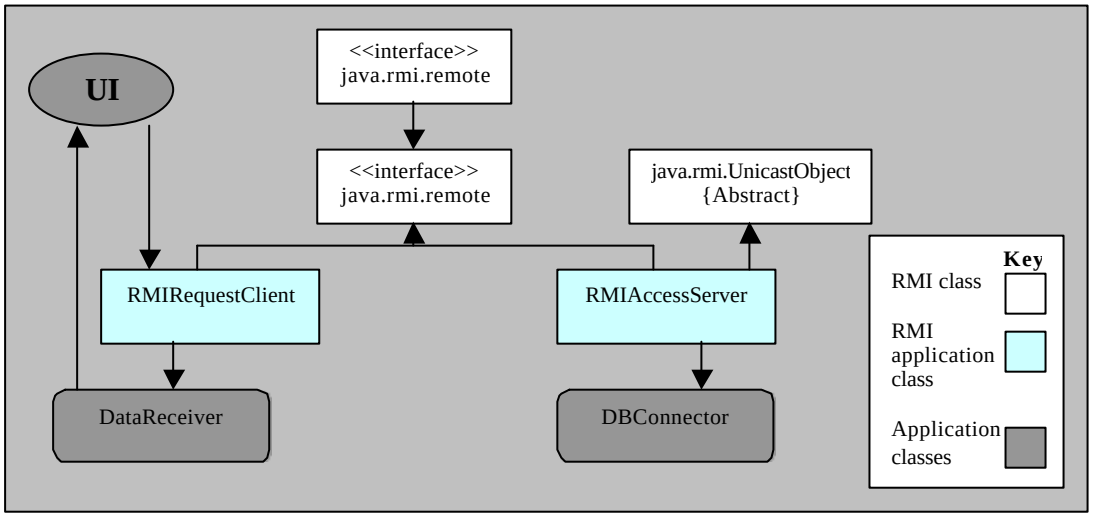

Figure 13: How the classes interact

2.6.1.1 Part 1: The Development of the RMI Client (Information Retrieval Engine)

The following are the primary features provided by the RMI client implementations via a graphical user interface:

a) Binding to the server's implementation using RMI.
b) Invoking the server's implementation with the appropriate commands
c) Displaying the query results to the user.

The graphical user interface part was built with the AWT, the standard graphical library of the Java Development Kit. It consists of three canvases, which include the main canvas, search canvas, and the display canvas.

- Main canvas

This is the outermost canvas. It allows users to search for audio information by artist, song title, and key words. A snapshot of this canvas is shown in Figure 14.

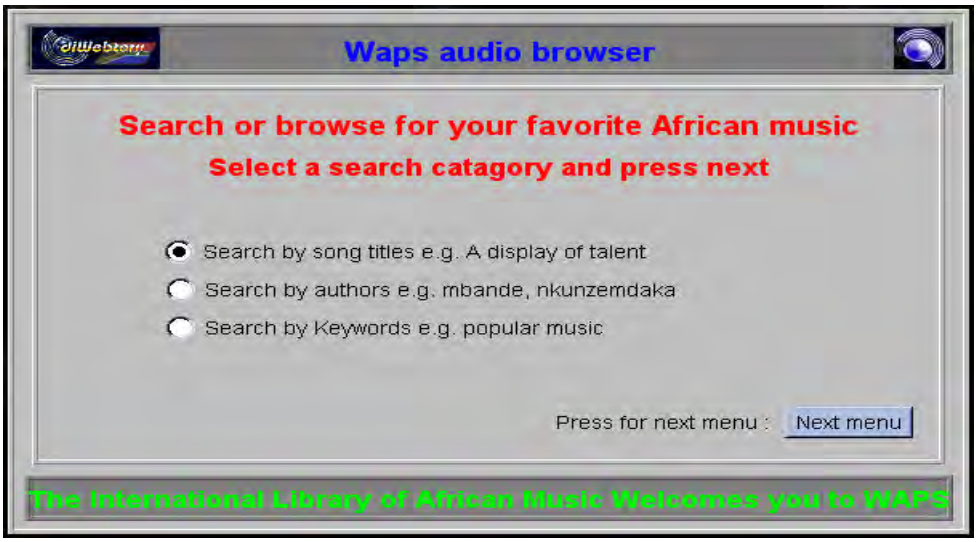

Figure 14: Main canvas

- Search canvas

This is a search engine. It provides a text field for submitting a search query to the server sited remotely. Figure 15 shows a snapshot of this canvas.

Figure 15: Search canvas

- Display canvas

This is for displaying the user's search results and allowing a user to navigate through the results. A snapshot of this canvas is shown in Figure 16.

Figure 16: Display canvas

To move between the canvases, a user uses the *Next*, and the *Back* buttons. The following code is run when the user presses the *Go get it* button on the search canvas.

```
if (initialized)
{
 try
  {
   String host = "rmi"://"+getCodeBase().getHost()=":4444/RMIAccessServer";
    //Binding the server's implementation using RMI
    server=(RMIAccessInterface)Naming.lookup(host);
    //Invoke an RMIAccessServer method and assign the results to a hashtable
    Hashtable authorTable=server.searchByAuthor(textField.get.Text());
   }
  catch (RemoteException e)
   {
    e.printStack Trace();
  }}
```

The calls to the server are printed in bold characters above. The first call makes a connection to the server object, and the second call invokes one of the server's methods (searchByAuthor()), passes a search string, and gets the returned hash table (authorTable). Once the results have been received from the server, they are then displayed within a List Box (left of the display canvas), and the user can then use the

Next, Previous, Open, Clear, and *Close* buttons to navigate through the audio information.

2.6.1.2 Part 2: The Development of the RMI Server Object (Data Access Server)

The server object is made up of two parts. The first part is the server's interface definition, which is a Java class file that defines all the methods that the server objects will make available for remote invocation. The interface allows for the establishment of interface definitions between the client and the server. It is equivalent to CORBA's IDL interface. To make the interface available to the client, RMI uses the RMI registry (naming service). The registry answers connection requests to RMI servers that registered themselves by returning their interfaces to the client. The calls are then performed through this interface. The interface for Data Access Server object is as follows:

```
import java.rmi.*
import java.util.Hashtable;
import DataConnector;

public interface AccessInterface extends Remote
{
 public Hashtable searchByAuthor (String name) throws RemoteExceptions;
 public Hashtable searchByTitle (String name) throws RemoteExceptions;
 public Hashtable searchByKeyword (String name) throws RemoteExceptions
 }
```

The second part of the RMI server object is the server object itself i.e. the RMIAccessServer. The server object basically is the implementation of the interface, and it makes its methods available for remote invocation through the interface and registers its name with the RMI registry server. It also establishes a connection with the database (Audio Storage Server) and waits for client queries. During a search request by the user, the client first makes a connection to the server object. This connection is performed in two steps:

- Firstly, a variable, which is the same type as the server interface is declared and

- Secondly, a connection with the naming service is established by passing on the server object's name on which one wants to retrieve a reference. This is

given by the URL: **rmi://<Name IP address of host>:<port>/<server object name>,** which is represented by the following code:

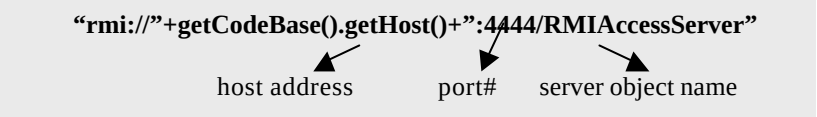

Figure 17 illustrates how this connection to the server is established.

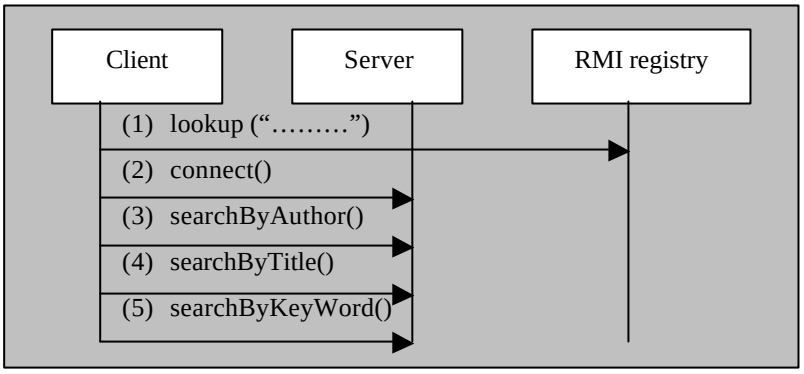

Figure 17: client server connection

The diagram below (Figure 18) summarizes the various development stages of the RMI implementation of the data component.

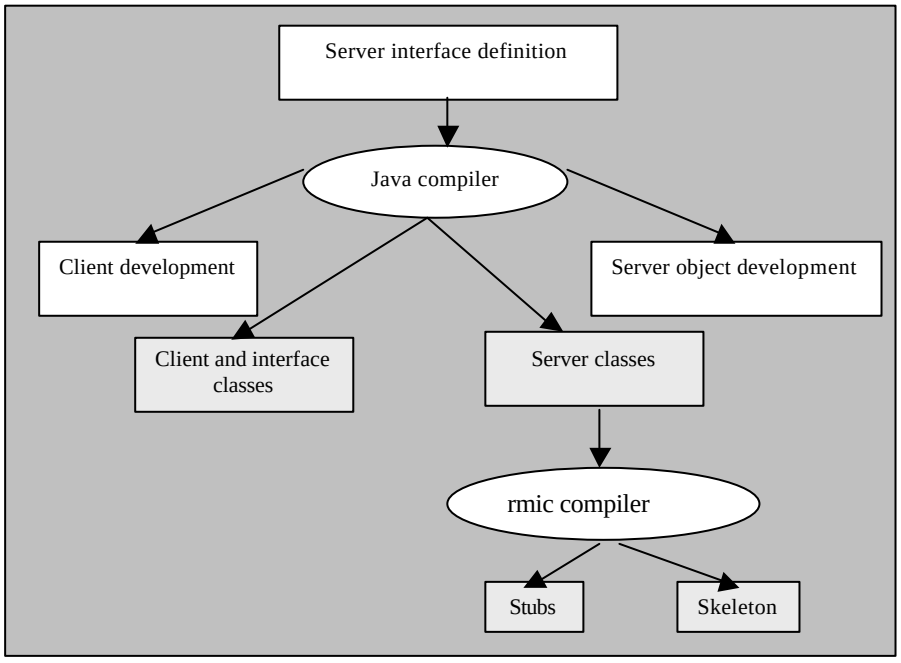

Figure 18: RMI development stages

The white rectangles represent the code writing steps i.e. the definition of the server object interface, writing the client and the RMI server object. The dark rectangles show the various classes that were automatically generated by the Java tools.

2.6.2 The Pure Java with CORBA Implementation

Here, the concept of CORBA is utilized to transfer audio information in a distributed environment. Just as with the RMI approach, the pure Java with CORBA implementation of the WAPS data communication component was also developed in two phases that included the development of the CORBA client (i.e. the client-tier named the Information Retrieval Engine) and the CORBA server object (i.e. middleware-tier named the Data Access Server). Figure 19 illustrates the 3-tier CORBA architecture implemented.

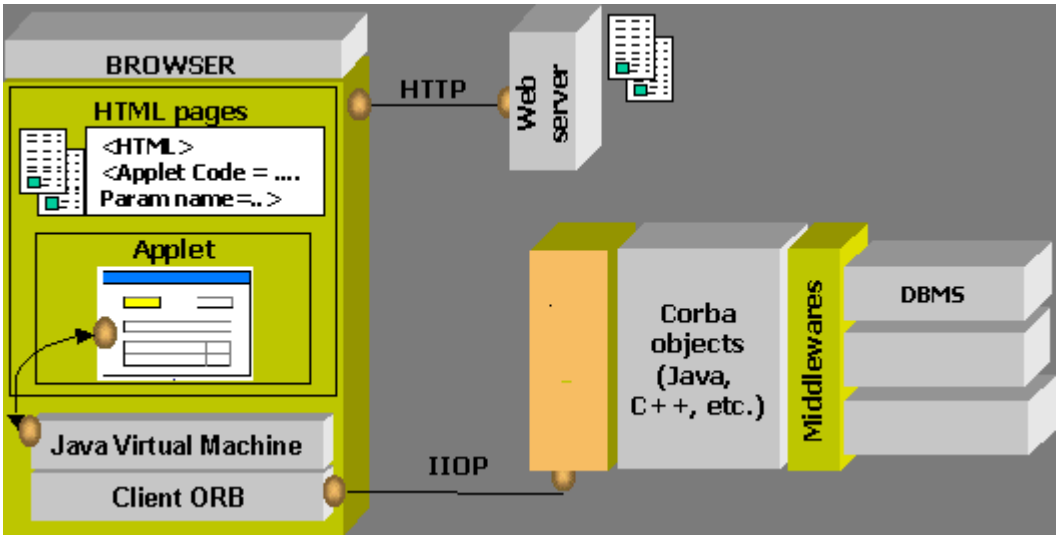

Figure 19: CORBA architecture of the data component

The CORBA implementation also makes use of the four main classes used in the RMI approach. These are the DataReceiver, CORBARequestClient (same functionality as the RMIRequestClient), CORBAccessServer (same functionality as the RMIAccessServer), and the DBConnector. The ORB used during the implementation comes standard with the JDK 1.2 version of Java. Sun Microsystems Java ORB complies with the OMG CORBA 2.0 specification and it implements a portion of CORBA's services. It provides mechanisms for the launching of server object interfaces using IDL and the use of an IDL-to-Java compiler to do the conversion to

Java class files. The IDL-to-Java mapping enables one to associate the CORBA data types to one of the languages chosen to develop the objects, which is Java in this case. With CORBA, each server object has to have an interface that specifies the methods that the server will make available for remote invocation.

This is, in principle, similar to the RMI interface. The following is the IDL file, which defines the interface of the Data Access Server's CORBA implementation.

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA

 module DataAccess
{
  interface AccessInterface extends Remote
  {
    string searchByAuthor (in string name)
    string search ByTitle (in string name)
    string search ByKeyword (in string name)
  }
}
```

Once defined, the idl2java compiler [12] is then used to create the necessary CORBA infrastructure (including stubs and skeletons). The diagram below (Figure 20) shows the various development stages of the CORBA version for the data component.

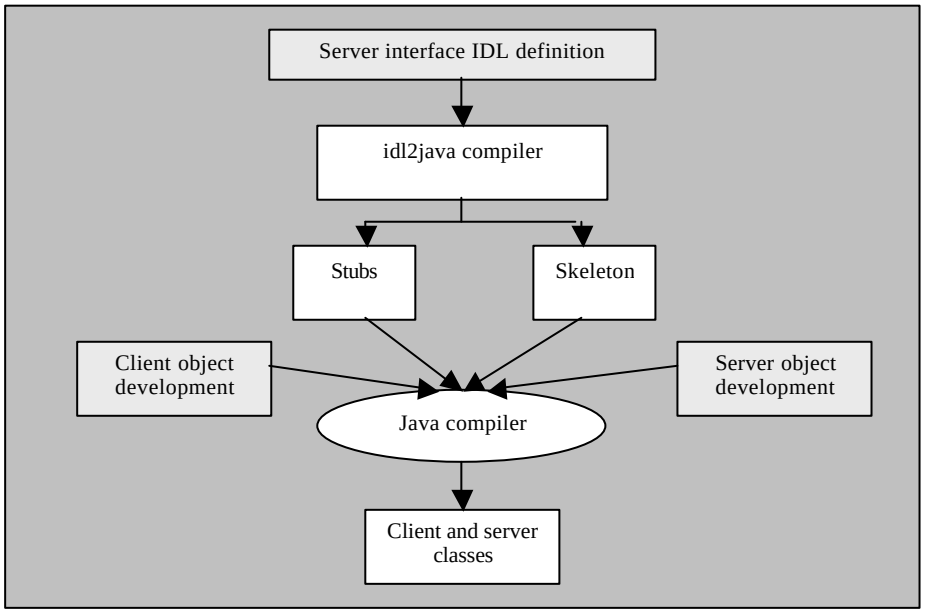

Figure 20: CORBA development stages

The dark rectangles represent code that was written i.e. the definition of the server object interface, writing of the client applet and the CORBA server object. The white rectangles show the various classes that were automatically generated by the Java tools used.

2.6.2.1 Phase 1: The Development of the CORBA Client

The entire functional code of the RMI client was used again with no further development needed. Only the communication mechanisms with the server were modified. This was only of concern at two points: access to the ORB, and the use of tokenized strings instead of hash tables to transmit the results of the method calls. The code needed to interface with the server is as follows:

```
try {
  // create and initialize the ORB
  ORB orb = Orb. int (parent, null);
  //Get the root naming context
  org.omg.CORBA.Objects obj = orb. resolve_initial_references("NamingServices");
  NamingContext nc = NamingContextHelper.narrow(obj);
  //Resolve the objects reference in naming
  NameComponent ncp = NameComponent ("waps","");
  NameComponent  path[] = {ncp};
  WapsRef = wapsHelper.narrow(nc.resolve(path));}
catch (Exceptions e) {
  e.printStackTrace(System.out); }
```

2.6.2.2 Phase 2: The Development of the CORBA Server Object

The implementation of the DBConnector objects uses 100% of the code from the RMI version, while the implementation of the CORBAccessServer object uses 95% of the code from the RMI version (RMIAccessServer). The only difference between the two is the way in which data is transported between the client and the server objects. For example, the Hashtable object does not correspond to any IDL type; hence the solution used in RMI could not be kept. Thus the results were tokenized into a string as the return type of the various methods of the server object e.g. row1+"$"+row2 +"$"+…+rowN. On the client side the dollar ("$") sign was then used to split or to un-tokenize the results into their original form.

2.6.3 Why RMI and CORBA?

As shown above, the data communication component has been implemented using two of the widely used distributed system architectures: CORBA and Java RMI. CORBA and RMI are two distributed object standards supported by the Object Management Group. They are not competing systems, but complementary ones. They each suit different needs, and the choice between them depends on one's requirements. The most important requirement is whether one requires homogeneity or heterogeneity. For example, systems that are made up of components in languages

other than Java demand CORBA. At the same time, by using RMI on a pure Java environment (such as WAPS tails) one is able to gain the benefits of distributed garbage collection, object serialization, and other Java features.

However, even in the 100% pure Java scenario, there is a place for CORBA. RMI is specifically not capable of handling highly distributed systems. It is still missing some of the key services, such as transaction management, that CORBA supports. The introduction of CORBA in the Java environment is still new and is not well supported by many Java virtual machines. In view of the above factors, both the CORBA and the RMI support were included in WAPS.

2.6.4 Response times RMI vs. CORBA

The literature discussing response times states that there are three important limits that are acceptable to users of web-based distributed systems [10]:

- 0.1 second

  This is just about the limit for having a user feel that the system is reacting instantaneously.

- 1.0 seconds

  This is about the limit for a user's flow of thought to stay uninterrupted, even though the delay will be noticeable.

- 10 seconds

  This is about the limit for keeping a user's attention focused on dialogue. For longer periods, users should be given feedback indicating when the system expects to be done.

Feedback during the delay is especially important if the response time is likely to be highly inconsistent, otherwise users will not know what to expect. This is particularly true on systems that are developed to function on the Internet, like WAPS. Hence in WAPS feedback is provided on the Information Retrieval Engine's status bar when a user searches for some musical material. It was mentioned that the Information Retrieval Engine communicates with the Data Access Server through both RMI and

CORBA. Using this client/server scenario some experimental tests were conducted in different network conditions in order to get readings on the performance and the response times of both RMI and CORBA method invocations. A search query method invocation was performed to the Data Access Server through the Information Retrieval Engine. With this query between fifty and one hundred records were received. Three of these search method invocations were undertaken. The tests were conducted across a 10 Mb/s Ethernet LAN, a 10 Mb/s dedicated connection, and finally over the Internet via a 28.8 kbps modern connection line. These tests measured the average response times of the RMI and the CORBA search method invocations.

The client machine used was a 133- MHz Pentium running Windows 98 and the server machine was a 233-MHz Pentium running Windows NT server 4.0. Table 1 compares the average response times from the experiments.

	RMI	CORBA
LAN	1.303 s	0.971 s
Dedicated	1.123 s	0.898 s
Internet	2.363 s	2.179 s

Table 1: RMI Vs CORBA method invocations

The RMI remote invocations performance numbers were higher than those of CORBA. This is due to the RMI's native protocol, the Remote Method Protocol (RMP), which uses Java serialization to marshal and un-marshal each argument. This causes some performance overhead, which eventually degrades the performance of RMI as opposed to CORBA's transport protocol (IIOP), which does not support object serialization

Summary

In this chapter, four approaches provided by the Java programming language, for accessing remote audio databases were identified. These include techniques based on database protocols e.g. JBDC, HTTP/CGI, RMI, and CORBA. The approaches based on HTTP/CGI and straight IP sockets have several drawbacks, such as data marshalling and performance, especially when it comes to 3-tier web database applications. The RMI mechanism offers an efficient and a lightweight communication between remote Java objects, and is the preferred solution for applications that are completely written in Java. Its object serialization technique is an added advantage over CORBA and reduces the need for data marshalling. Although CORBA offers more services than RMI and the performance tests proved that it is faster than RMI, it is still not well supported by many Java virtual machines on the market.

Chapter 3

Internet Audio Codecs

The purpose of this chapter is to briefly present technologies and standards that are currently used by audio applications on the Internet. It will show the environment in which WAPS was designed, and help justify the utilization of some technologies and the rejection of others. In this chapter data compression will be discussed, specifically focusing on audio, and several existing audio compression standards that facilitate the design and the implementation of audio applications in the Internet environment will be presented. The procedure followed and the results of the subjective audio listening tests, which were conducted for some of the standard and the most popular Internet audio codecs are presented.

3.1 Digital Audio

Audio files pose one major problem when sent across data communication networks, especially the Internet. They consume a large amount of bandwidth. For example, uncompressed CD-quality audio is stored as 16 bit samples and requires 44100 samples per second to capture the full range of frequencies that a listener can hear. Although the quality is very high, this requires a data rate of 700 kbps and for stereo files the data rate doubles to 1400 kbps. Given that the physical access to the Internet is limited, this becomes far too large to be delivered in real-time. Two factors contribute to the bandwidth consumed by an audio file: the sample rate and the quantization level.

3.1.1 Sample Rate

Frequencies perceived by the human ear as sound are typically between 20 Hz and 20 kHz. The human voice can produce frequencies between 40 Hz and 4 kHz. In order to preserve all the frequencies in the range of human hearing (20 Hz – 20 kHz), an analog to digital converter must sample an analog signal at 40 kHz. This is in accordance with the Shannon-Nyquist theorem [13], which states that the sample rate must be at least twice the highest frequency to be sampled. A sampling rate any lower

than twice the highest frequency produces aliasing distortion. In practice a sampling rate of 44.1 kHz or 48 kHz is used for highest audio quality. However, sampling at high frequencies increases the size of the audio file.

3.1.2 Quantization Level

When an analog signal is sampled, the amplitude of each sample has to be stored and any number of bits can be used to represent this amplitude. The number of bits that are used determines the number of quantization levels. For digital audio, this precision usually ranges from 8 bits per sample (256 levels) to 16 bits per sample (65536 levels). Normally, samples that fall in between two quantization levels are assigned the closest value. For example, Figure 1 shows a value lying half way between two quantization levels and it is assigned a value of 1. In Figure 1, four quantization levels are used and hence 2 bits are needed to encode each sample. This process is referred to as quantizing. The difference between the actual analog value and the chosen quantization interval value is called quantization error [14].

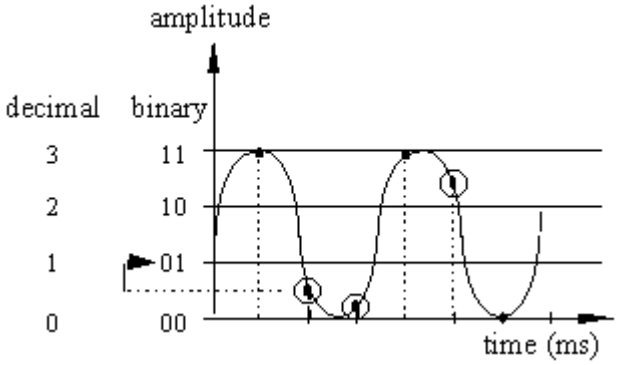

Figure 1: Quantization error shown on a 2-bit system

Quantization error produces audible distortion. The more bits used in the sampling process, the less audible the quantization distortion. Hence, it is important to use enough bits to lower the quantization distortion to a level where it cannot be heard. Most of the digital audio industry has agreed that 16 to 20 bits is adequate. On the other hand, using more bits during the sampling process increases audio file size.

3.1.3 Bandwidth Consideration

As much as one would like the highest audio quality possible, with 16 bits being generated 441100 times per second, it does not take long to generate a very large

audio file. For example, a one-minute stereo song will generate a 10.58 million-byte file (10.34 Megabytes). A 64-minute compact disc requires 650 MB of storage. Apart from the need for very large storage devices for the files, these large files present a problem when they are to be transferred across networks, especially over standard telephone lines. For example, using a sampling rate of 44.1 kHz and 16 bits per sample, stereo audio generates a 1.4 MBPS real-time audio signal. Considering that all transmission protocols add bits to the audio file for routing purposes, this amount of bandwidth eliminates the possibility of playing high quality audio through a 28.8 kbps modem in real-time. Thus, some way is needed to reduce the amount of data in an audio file whilst retaining quality. This is achieved through compression.

3.2 Audio Compression

Data compression reduces the amount of space that must be allocated for information [15]. This is particularly important in current network environments where bandwidth is the most crucial constraint. Compression schemes can either be lossless or lossy.

- **Lossless Compression**

  This is also known as redundancy reduction. This scheme removes or reduces the redundancy in the information, but enables these modifications to be inverted and data structures to be reconstructed. As the name implies, lossless compression does not introduce any signal distortion or information losses. In lossless data compression, the output (decompressed) data is guaranteed to exactly match the input data after the compress/decompress cycle. This type of compression is mainly used when storing database records, spreadsheets, or word processing files. In these applications, the loss of even a single bit could prove catastrophic.

- **Lossy Compression**

  This is also known as entropy reduction. Compression techniques that are based on this method result in a loss of information, which cannot be recovered. This method is irreversible and introduces signal distortion. Lossy data compression gives up a certain loss of accuracy in exchange for increased compression. It is generally applied to digital multimedia data. With

multimedia data, the idea of output and input signals not exactly matching is somewhat acceptable. Lossy compression techniques can be adjusted to different quality levels, gaining higher accuracy in exchange for less compression.

3.3 Audio Compression Schemes

Audio compression coder/decoders (codecs) attempt to preserve all of the frequencies in the range of human hearing (20 Hz – 20 kHz). In the case of human speech, only frequencies in the range of 20 Hz – 3.4 kHz are necessary, so speech codecs are only concerned with this range. However, this chapter will focus on audio codecs. Codecs are implemented either as software programs or as dedicated hardware that can be added to a computer or an audio device. In this discussion only software based audio codecs are considered. During audio compression, a user typically loads a 16 bit, 44.1 kHz PCM audio file into an encoder and then saves the new compressed copy of the file. The output file is not in the same format as the original file, so a corresponding decoder is required to reconstruct the audio file into the original format. There are two basic types of audio compression codecs: time domain and frequency domain.

- **Time Domain Codecs**

In time domain codecs the parameters that model the audio signal are derived in the compressor. These parameters are transmitted to the decompressor, which then attempts to re-synthesize the audio based on the same model used in the compressor. *Vocoders* and *linear predictive codecs* are the most often used examples of time domain codecs [14]. These codecs have been successfully used to achieve low-bit rate coding of speech. Their drawback is that they are based on a model of the human vocal tract that can be used to compress voice very efficiently, but does not accurately reproduce music (and other types of audio).

- **Frequency Domain Codecs**

With the frequency domain codecs, the audio signal is divided into a set of frequency components that are encoded separately. This type of coding scheme is mainly used in music. With this scheme, the highest coding efficiency is achieved with algorithms exploiting signal redundancies and irrelevancies in the frequency

domain based on a model of the human auditory system. All codecs use the same basic structure. Frequency domain codecs include two more coding types, transform and subband. Transform codecs map the time-based PCM samples into the frequency domain. The result is a set of frequency components that can be coded separately. MPEG-1 Layer III is one example of a transform codec. Subband codecs are hybrids of frequency domain codecs and time domain coding methods. The signal is broken into smaller equal bandwidth subbands of different frequency ranges while remaining in the time domain. Concurrently, the same signal is mapped to the frequency domain and the spectral lines fed to a psychoacoustic model. The model then dictates the quantization level of each subband in the time domain. The most popular examples of subband codecs are MPEG-1 Layer I and II coders.

3.4 Current Audio Coding Schemes

The existing audio coding schemes that are used for Internet applications can be roughly divided into three groups: MPEG codecs, derivatives of the AC-3 codec, and speech codecs [16]. In this section some of the standard audio codecs and those that are proprietary but widely used are discussed. The codecs to be discussed include MPEG-1 (Layers I, II, and III), MPEG-2 AAC, MPEG-4, RealAudio G2, and TwinVQ,

3.4.1 MPEG Audio

The Motion Picture Experts Group (MPEG) audio compression algorithm is an International Standards Organization (ISO) standard for high fidelity audio compression. The composite standard addresses the compression of synchronized video and audio at a total bit rate of roughly 1.5 megabits per second. MPEG works in phases, which are normally denoted by Arabic numbers (MPEG-1, MPEG-2, MPEG-4). The audio activities of the first phase, MPEG-1, were finalized in 1992, while those of the second phase, MPEG-2, were finalized in 1997. Another phase currently under way is called MPEG-4 and is in its final stages of development. Both the MPEG-1 and the MPEG-2 phases define three different layers. These layers represent a family of coding algorithms and they are normally denoted by Roman Numbers i.e. Layer I, Layer II, and Layer III. Each layer offers increased compression as well as

increased encoding complexity. All these layers are defined within the audio part of the existing international standards, MPEG-1 and MPEG-2.

By using MPEG audio coding, one may compress the original sound data from a CD by a factor of 12 without losing any significant sound quality. Factors of 24 and even more are possible and still maintain a sound quality that is significantly better than one would get by just reducing the sampling rate and the resolution of the samples. Basically, this is realized by perceptual coding techniques addressing the perception of sound waves by the human ear. Using the MPEG audio standard, one may achieve a typical data reduction shown in Table 1, but still maintain the original CD sound quality.

Ratio	MPEG Layer
1:4	By Layer I (Corresponds to 384 kbps for a stereo signal)
1:6 – 1:8	By Layer II (Corresponds to 256 – 192 kbps for a stereo signal)
1:10 – 1:12	By Layer III (Corresponds to 128 – 112 kbps for a stereo signal)

Table 1: MPEG reduction ratio vs CD quality

By exploiting stereo effects and by limiting the audio bandwidth, these layers can achieve an acceptable sound quality at even lower bit rates.

3.4.1.1 MPEG-1 Layer I Audio

This is the simplest layer that is suitable for consumer use. It features subband filtering with 32 equal width subbands, adaptive bit allocation, and block compounding. Bit rates range from 32 kbps (mono) up to 448 kbps (stereo). Depending on the complexity of the encoder, high (near CD) audio quality requires a bit rate of about 256 – 384 kbps for stereo audio. The complexity of the decoder is low, while that of the encoder is about 1.5 to 3 times higher than that of the decoder. The Layer I algorithm uses a basic filter bank, found in all layers, to divide the audio signal into 32 constant width frequency bands. These filters are relatively simple and provide good time resolution with reasonable frequency resolution relative to the perceptual properties of the human ear [13]. The filter bank provides 32 frequency

samples, one sample per band, for every 32 input audio samples. The Layer I algorithm groups together 12 samples from each of the 32 bands. Each group of 12 samples receives a bit allocation and if the bit allocation is not zero, a scale factor. The bit allocation determines the number of bits used to represent each sample. The scale factor is a multiplier that sizes the samples to maximize the resolution of the quantifier. The Layer I encoder formats the 32 groups of 12 samples (i.e. 384 samples) into a frame. Besides the audio data, each frame contains a header, an optional cyclic redundancy code (CRC) check word, and possibly ancillary data.

3.4.1.2 MPEG-1 Layer II Audio

MPEG-1 layer II audio offers more compression than Layer I. It has numerous applications in both consumer and professional audio, such as audio broadcasting, television, telecommunication, and the Internet. Bit rates range from 32 – 192 kbps for mono and from 64 – 384 kbps for stereo. Depending on the complexity of the encoder, a high (near CD) audio quality requires a bit rate of about 192 – 256 kbps per stereo channel. The complexity of the decoder is about 25% higher than for Layer I decoder, with the encoder complexity about 2 to 4 times higher.

The Layer II algorithm is a simple enhancement of Layer I. It improves compression performance by coding data in larger groups. Encoders based on Layer II form frames of 3 by 12 by 32 = 152 samples per audio channel. Whereas Layer I codes data in single groups of 12 samples for each subband, Layer II codes data in 3 groups of 12 samples for each subband. Discounting the stereo redundancy, there is one bit allocation and up to three scale factors for each trio of 12 samples. The encoder encodes with a unique scale factor for each group of 12 samples only if necessary to avoid audible distortions.

The encoder shares scale factor values between two or three groups in two other cases:

(i) when the values of the scale factors are sufficiently close and

(ii) when the encoder anticipates that the temporal noise masking by the ear will hide the consequent distortion.

Layer II also improves performance over Layer I by representing the bit allocation, the scale factor values, and the quantified samples with a more efficient code.

3.4.1.3 MPEG-1 Layer III Audio

MPEG Layer III audio is one of the most powerful members of the MPEG audio coding family [17]. It offers even more compression than the Layer I and Layer II coders. Its applications extend into narrow band telecommunication and certain specialist areas of professional audio. Table 2 gives an overview of the level of complexity of all the three layers.

Layer	Complexity	Level
	Encoder	**Decoder**
I	1.5 – 3	1.0
II	2 – 4	1.25
III	> 7.5	2.5

Table 2: Encoder/Decoder level of complexity

It has to be taken into account that the complexity has to do with the amount of processing power and memory that is required if a general-purpose machine performs the encoding and decoding functions.

While MPEG Layer II specifies a coded representation that can be used for compressing audio sequences, both mono and stereo, the Layer III algorithm is a much more refined approach. The algorithm is illustrated in Figure 2 below. Input audio samples are fed into the encoder. The mapping creates a filtered and sub-sampled representation of the input audio stream. A psychoacoustic model creates a set of data to control the quantifier and coding. The quantifier and the coding block create a set of coding symbols from the mapped input samples. The block 'frame packing' assembles the actual bit stream from the output data of the other blocks, and adds other information (e.g. error correction) when necessary. Several international listening tests for high quality audio have been undertaken, whereby MPEG-1 Layer III impressively proved to have superior performance, maintaining the original sound

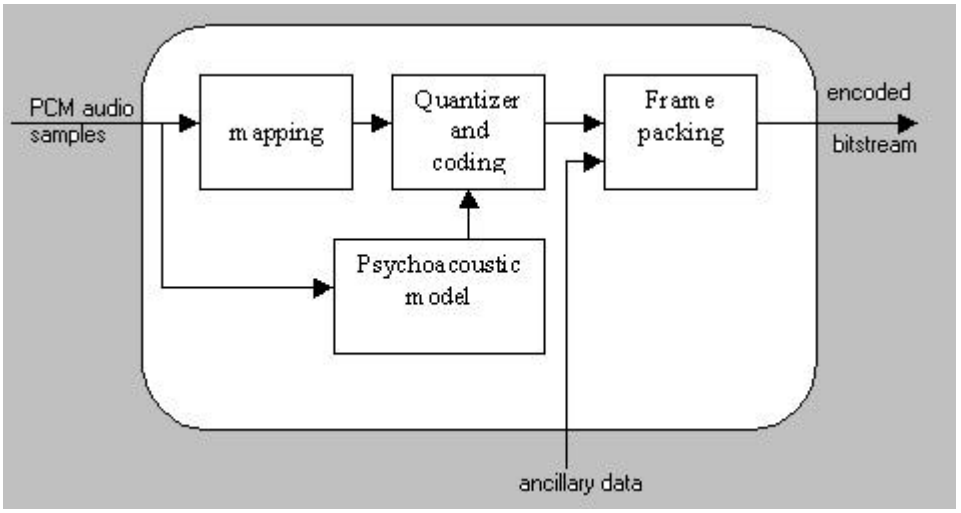

Figure 2: MPEG Layer III encoder

quality at a data reduction of 1:12 (around 64 kbps per audio channel) [18]. However, there have been hardly any listening tests conducted for lower bit rates that could benefit web-based applications that require bandwidth of around 6–10 kbps. Table 3 illustrates the typical performance data values of MPEG Layer III [18].

SOUND QUALITY	BANDWIDTH	MODE	BITRATE	REDUCTION RATIO
Telephone sound	2.5 kHz	mono	8 kbps	96:1
Better than short-wave	4.5 kHz	mono	16 kbps	48:1
Better than AM radio	7.5 kHz	mono	32 kbps	24:1
Similar to FM radio	11 kHz	stereo	56-64 kbps	26-24:1
Near – CD	15 kHz	stereo	96 kbps	16:1
CD	>15 kHz	stereo	112-128 kbps	14-12:1

Table 3: MPEG Layer III performance values

3.4.1.4 MPEG-2 Advanced Audio Coding

MPEG-2 AAC is the consequent continuation of MPEG Audio Layer III. Previously known as MPEG-2 NBC (Non-backward compatible), MPEG-2 AAC was declared an international standard by MPEG in 1997 [19]. It supports sampling frequencies between 8 kHz and 96kHz and any number of channels between 1 and 48. Presently MPEG-2 AAC is the most efficient method for audio data compression. The driving force behind its development was the quest for an efficient coding method for

surround signals, like 5.1 channel signals (left, right, center, left surround, and right surround) as are being used in cinemas and digital television systems. However, the high performance of MPEG-2 AAC also applies to mono and 2-channel stereo signals, and its high coding efficiency makes it an exceptional candidate for the Internet.

MPEG-2 AAC basically makes use of the signal masking properties of the human ear, in order to reduce the amount of data, just as all perceptual coding schemes do. The quantization noise is distributed among frequency bands in such a way that it is masked by the total signal i.e. it remains inaudible. The basic structure of this coding method is shown in Figure 3.

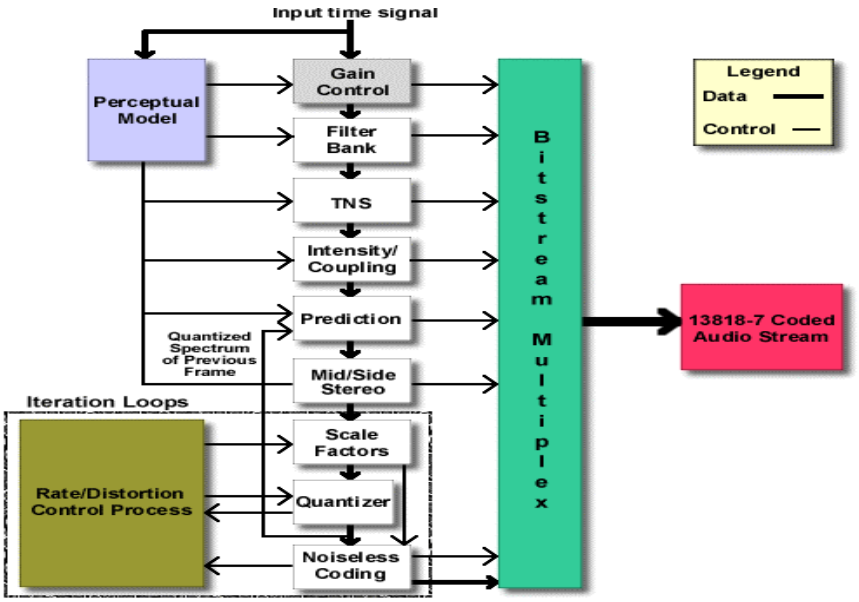

Figure 3: MPEG-2 AAC structure

There are some crucial differences between MPEG-2 AAC and MPEG Audio Layer III. These are shown as follows:

- **Filter bank**

  MPEG-2 AAC uses a plain Modified Discrete Cosine Transform (MDCT). Together with the increased window length (2048 instead of 1152 lines per transformation) the MDCT outperforms the filter banks of the previous coding methods.

- **Temporal Noise Shaping (TNS)**

This shapes the distribution of quantization noise in time by prediction in the frequency domain.

- **Prediction**

This technique is commonly established in the area of speech coding systems. It benefits from the fact that a certain type of audio signal is easy to predict.

- **Quantization**

  By allowing finer control of quantization resolution, the given bit rate can be used more efficiently.

- **Bit-Stream**

  The information to be transmitted undergoes entropy coding in order to keep redundancy as low as possible. The optimization of these coding methods together with a flexible bit-stream structure has made further improvement of the coding efficiency possible.

3.4.1.5 The MPEG-4 standard

After MPEG-1 and MPEG-2, the next phase of compression work by the Moving Picture Experts Group is called MPEG-4. While the former MPEG standards (i.e. MPEG-1 and MPEG-2) were only concerned about compression, MPEG-4 provides additional functionality (e.g. bit rate scalability, object-based representation, intellectual property management and protection, etc). MPEG-4 is based on a rich set of coding tools, starting at bit rates as low as 2 kbps for a channel. The MPEG audio codec includes coding tools from different coding paradigms. These include parametric audio coding, synthetic audio, speech coding, and subband/transform coding. The high-quality part of the MPEG-4 audio functions is covered by what is called “t/f” coding. In a t/f coder the input signal is first decomposed into a time/frequency (t/f) spectral representation by means of an analysis filterbank prior to subsequent quantization and coding. The core part of the MPEG-4 audio t/f coder is based on MPEG-2 AAC technology. At present there are no fully completed MPEG-4 implementations. The MPEG-4 standard is meant to become universal in

broadcasting, movies, and multimedia applications, mostly applications that are based on the Web, due to its low bit rate compression techniques.

3.4.2 Transform-domain Weighted Interleave Vector Quantization (TwinVQ)

TwinVQ is a music compression technology that has been developed by the NTT Human Interface Laboratories in Japan [21]. It is a transform coding method like MPEG Audio Layer III or AAC. However, TwinVQ encoded files are much smaller than those of MPEG Audio Layer III, whilst providing high quality. In this method, the individual bits of music data are not directly encoded, but are combined into pattern segments (vectors). These patterns are then compared against standard patterns, which are prepared in advance. The standard pattern, which provides the closest match, is selected and the number associated with that pattern is transmitted as the compression code. Data is then packed into long frame mode or short frame mode (8 sub-frames) using a constant bit rate in order to enhance error robustness. The coding distortion is minimized even at low bit rates so that music can be successfully regenerated.

3.4.3 RealAudio G2

RealAudio is one of the oldest formats for compressed audio. It was first designed for voice applications on the web, but later was extended to music and video applications as well. It is developed by Real Networks [22] and it implements a codec developed at Dolby Laboratories, called DolbyNet [23]. This music codec was designed for maximum quality at typical Internet speeds of 16 to 32 kbps. It was, however, extended to rates outside the original design to offer data rates ranging from 6 kbps to 96 kbps with an average frequency response ranging from 3.5 to 22.4 kHz. Table 4 and Table 5 illustrate the average frequency responses achieved by RealAudio.

Data rate	6 kbps	8 kbps	11 kbps	16 kbps	20 kbps	32 kbps	44 kbps	96 kbps
Frequency response	3 kHz	4 kHz	5.5 kHz	8 kHz	10 kHz	16 kHz	22 kHz	22 kHz
Sampling rate	8 kHz	8 kHz	11 kHz	16 kHz	22 kHz	44 kHz	44 kHz	44 kHz

Table 4: Frequency responses for mono music

Data rate	20 kbps	32 kbps	44 kbps	64 kbps	96 kbps
Frequency response	3 kHz	8 kHz	11 kHz	16 kHz	22 kHz
Sampling rate	11 kHz	16 kHz	22 kHz	44 kHz	44 kHz

Table 5: Frequency responses for stereo music

The RealAudio coder uses a combination of fast algorithms, approximation, and clever implementation to achieve fast operation with little loss of quality. Instead of expensive functions, RealAudio uses lookup tables or simple approximations, and instead of iterative procedures, it uses prediction. RealAudio has been designed to limit the algorithmic dependencies on prior data, and allows encoded data to be handled in relatively small and independently decoded units. This prevents a data frame loss from affecting the frames around it during transmission. It also allows the compressed frames to be interleaved before being grouped into packets for transmission over the network. These small audio gaps are then "filled-in" using an interpolation scheme that uses past and future data to estimate content that might be lost during transmission.

3.4.4 Codecs used in WAPS

The audio formats that are currently supported by WAPS include: MPEG-1 Layer II, Sun/NexT (AU) format, QuickTime audio, and MPEG-1 Layer III. The support for these audio formats depended on the support of these formats by the Java Media Framework API, which was used when developing the WAPS audio player. Also, the WAPS audio player provides support for plug-in players such as Liquid Player, RealNetwork's RealPlayer G2, MP3 players such as Bitcasting's MP3 player, and Microsoft's Windows Media Player.

3.5 Audio Verification Tests

Listening tests are how audio formats are evaluated. For example, established formats like MPEG Audio Layer III or AC-3 (from Dolby laboratories) undergo extensive tests to establish whether listeners can tell the difference between an original and a

compressed signal. These listening tests have to be conducted extremely carefully to legitimately confirm the "transparency" (similarity to the original signal) of a given format. Internet audio applications utilize audio coding tools that cover a bit rate range from 6 kbps to 56 kbps, with a corresponding subjective audio quality that needs to be evaluated. The purpose of this section is to describe some subjective testing procedures that have been followed in this project and to present the outcome of the verification subjective tests on Internet audio codecs. These tests show the performance of particular encoders and decoders, and do not necessarily show the performance of the algorithm. Under investigation were low bit rate schemes suitable for streaming (real-time delivery) of audio via 33.6 or 56 kbps modems.

3.5.1 Test Motivation

The demand for music transmission in real-time over networks, like the Internet, forms the background for these tests, which evaluate recent audio coders at bit rates suitable for analog modem and ISDN connections. The tests also helped identify the audio codecs that could be used by WAPS and any similar Internet applications. The comparisons of interest are:

- to compare the recent coding tools for transmission of audio at bit rates below 10 kbps.
- to compare the recent coding tools for transmission of audio at bit rates between 10 and 30 kbps and
- to compare the recent coding tools for transmission of audio at bit rates between 40 and 56 kbps.

3.5.2 Codecs under Test

The test was divided into three groups of coding schemes/bit rate:

- Group A tests the codecs at 6 and 8 kbps mono signal.
- Group B tests the codecs at 16 and 24 kbps mono signal.
- Group C tests the codecs at 40 and 56 kbps stereo signal.

The codecs tested are listed in Table 6:

Codec family	Source	Label	Mode	Bitrates (kbps)
MPEG-1 Layer II	FhG-IIS[1]	L II	mono, stereo (40 kbps)	24, 56
MPEG-1 Layer III	FhG-IIS	L III	mono, stereo (40 kbps)	8, 16, 40
TwinVQ	Yamaha	VQF	mono, stereo (40 kbps)	8, 16, 40
MsAudio	Microsoft	MA	mono, stereo (40 kbps)	8, 16, 40
RealAudio G2	Real Networks	RA	mono, stereo (44 kbps)	20, 44

Table 6: Codecs tested

In these tests, the term *codec family* is used to identify the various codec proponents (e.g. Layer II, Layer III, RealAudio, etc), while the term codec represents a specific codec family/bit rate combination (e.g. Layer II at 56 kbps, MsAudio at 8 kbps, etc.). The term codec group will be used to identify a group of codecs operating at a certain bit rate interval (e.g. Group A refers to codecs with bit rates between 6 and 8 kbps, while Group B refers to bit rates between 16 and 24 kbps, as indicated above. A total of 13 codecs were examined in these subjective tests. These consisted of 5 codec families operating at various bit rates to yield 13 codecs shown in Table 6 above. Unfortunately, the tested codecs were not available at all bit rates for each codec family.

3.5.3 Test Material

When conducting audio listening tests, samples must be chosen which are both hard to compress and likely to create artifacts which can be easily identified. Examples of such content include isolated instruments, drums, acoustic guitars, and solo vocal passages. In these experiments, three samples based on the above examples were selected. The samples included a female voice speech, an acoustic guitar, and a rock piece with a strong drum component

3.5.4 Test Methodology

The subjective assessment of sound quality according to ITU-Recommendation BS.562.3 [24] was followed. This method (see Table 7) uses a five-grade scale for scoring:

[1] The Fraunhofer Institute for Integrated Circuits

	BS.562.3 Quality Scale
	5 Excellent
	4 Good
	3 Fair
	2 Poor
	1 Bad

Table 7: The BS.562.3 Quality Scale

This method also recommends the use of this scale as a continuous scale with one decimal place. This means that users can provide grades between these values down to one decimal place e.g. 4.3. Within each test group (A, B, C), the codecs were compared to a bandwidth-limited reference. The bandwidth of this reference was equal to the bandwidth of the coder with the highest bandwidth, which was 56 kbps. For all test groups i.e. A, B, and C, the audio playback sequence type (shown in Table 8) used during the experiments was R (Reference) – A (Coded), i.e. the reference sample was played first followed by the coded sample. The reference used was a MPEG-1 Layer III audio file encoded at 56 kbps.

Experimental group	*A*	*B*	*C*
Sequence type	R-A	R-A	R-A
Scoring	BS.	562.	3

Table 8: Audio playback sequence

3.5.5 Test Results

In this section the results of the subjective test are examined. The discussion begins with an assessment of subject reliability and is followed by an analysis of the performance of all codecs tested.

3.5.5.1 Subject Reliability

Listener reliability was evaluated by ensuring that each listener could consistently distinguish between the original reference sound and the coded sound. Then statistical t-tests were performed for each listener over the listener's aggregate responses to all coded items, to test that the responses differed from 5 (maximum grade). In this

scenario, this is not a strong criterion since the reference was not hidden from listeners and they were instructed to score only the coded samples with grades between 5.0 and 1.0 inclusive. Hence the listeners had mean grades different from 5 with $p < 0.1$ for all the codecs. Thus no listener grades were discarded.

A strong criterion could have been taken whereby it would have been ensured that the listeners are able to make clear distinctions between the coded samples and the original reference. For example, for each subject, a one-way ANOVA would be conducted for each of the test groups. Subjects would then be retained only if they showed significant distinctions between codecs on half or more than half of the tested codecs in which they tested. This strong criterion was not followed in these tests due to the fact that it is mainly applied to double blind subjective tests of high quality audio codecs against a CD-quality reference (instead of a bandwidth limited reference, as in these tests). The testing procedures for double blind subjective tests are outlined in the ITU-R Recommendation BS.1116 and the bit rates for the codecs to be tested vary between 64 and 192 kilobits per second per stereo pair, which is contrary to the conducted tests that focussed on bit rates between 6 and 56 kilobits per second suitable for Internet audio transmission purposes.

3.5.5.2 Codec performance

Outcomes of the experiments are shown in the following three sections:

3.5.5.2.1 Overall quality by codec groups

In this section the results of the subjective tests are analyzed with respect to the overall quality of the three codec groups i.e. Group A for bit rates between 6 and 8 kilobits per second, Group B for bit rates between 16 and 24 kilobits per second, and Group C for bit rates between 40 and 56 kilobits per second. The overall codec group performance is examined in Figure 4. In Figure 4, the horizontal axis indicates the increasing bit rate while the vertical axis indicates the overall grading scale.

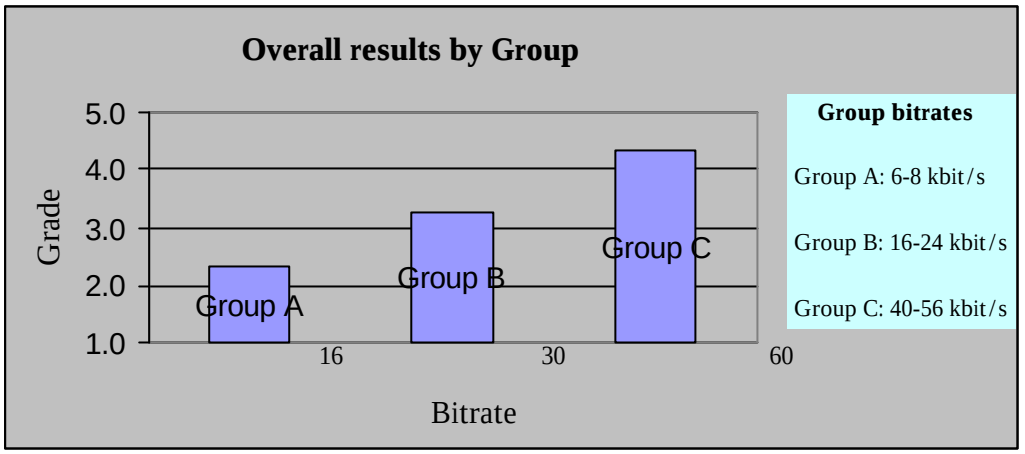

Figure 4: Overall quality by individual codec group

It is apparent from Figure 4 that, as expected, the performance of each codec group improves monotonically with increasing bit rate. The ranking of the codec groups with respect to quality is Group A, Group B, Group C, from lowest to highest.

3.5.5.2.2 Overall quality by individual codec

In this section, the results of the subjective tests are examined with respect to the overall quality of each of the individual codecs. The results are summarized in a novel graph-Table (Figure 5) that simultaneously allows one to compare codecs in terms of quality and bit rate.

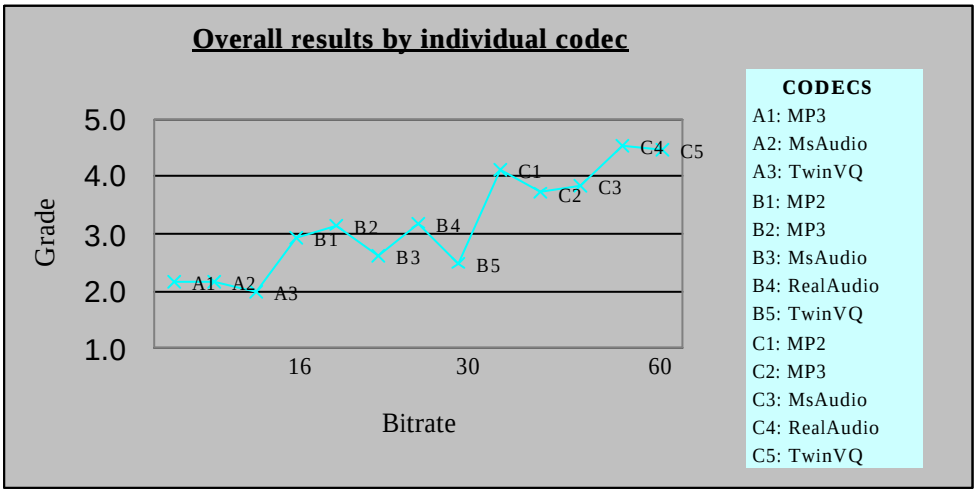

Figure 5: Codec performance versus subjective audio quality

Just as with the results of the overall codec quality by group, it is also apparent from the figure that, as expected, the performance of each codec improves with increasing bit rate. The ranking of the individual codecs with respect to quality is given in Table 9 below. The TwinVQ and RealAudio codecs performed identically at bit rates of 40 kbps and are in the "Excellent" range on the quality scale. The MPEG-1 LII at 56 kbps, MsAudio at 40 kbps, and MPEG-1 LIII at 40 kbps codecs are the next, falling on the boundary between " Excellent" and "Good", with MPEG-1 LII at slightly higher bit rates than others. The MPEG-1 LIII, MsAudio, and TwinVQ codecs performed identically at 8 kbps and all are in the "Poor" range.

Grading	Codecs
(5) Excellent	VQF/40, RA/44
(4) Good	L II/56, MA/40, L III/40
(3) Fair	L III/16, RA/20, L II/24, MA/16, VQF/16
(2) Poor	L III/8, MA/8, VQF/8
(1) Bad	

Table 9: Ranking of the individual codecs

3.5.5.2.3 Quality of each audio test material by individual codecs

This section presents the results of the individual codecs for each test material. The results of the overall quality of each of the individual codecs for the acoustic guitar (item 1), the contemporary rock piece with drums (item 2), and the English female speech (item 3) are summarized in the graph-Tables: Figures 6, 7, and 8 respectively. Figure 6 and Table 10 present the results of the individual codecs for the acoustic guitar, Figure 7 and Table 11 show the results for the contemporary rock piece with drums, and Figure 8 and Table 12 show the results for the female voice test material.

From (Figure 6, Table 10) codec performance for item 1 (acoustic guitar), RealAudio at 44 kbps performed better than all the other codecs; MsAudio at 8 kbps, and TwinVQ at 8 kbps performed at "Poor"; while the rest of the codecs performed between "Fair" and "Good" on the quality scale.

From (Figure 7, Table 11) none of the codecs performed at excellent. The MsAudio at 8 and 16 kbps , MPEG-1 L III at 8 kbps, and Twin VQ at 8 kbps, codecs

remained at “Poor”. The rest of the codecs performed between “Fair” and “Good” on the quality scale.

From (Figure 8, Table 12), RealAudio at 44 kbps and TwinVQ at 40 kbps performed at “Excellent” on the quality scale. The TwinVQ at 16 kbps, MPEG-1 L III at 16 kbps, and MsAudio at 16 kbps joined MPEG-1 L III at 8 kbps , MsAudio at 8kbps , and Twin VQ at 8 kbps to perform at “Poor” on the quality scale. The rest of the codecs remained between “Fair” and “Poor” on the quality scale.

On both item 1 and item 3 RealAudio at 44 kbps performed better than all the other codecs and its performance was very consistent across all the audio items. Group A codecs (i.e. MPEG-1 L III, MsAudio, and TwinVQ, all at 8 kbps) performed at “Poor” on the quality scale for all the items. Generally, the Group B and Group C codecs performed from “Fair” to “Excellent” for all the items despite the poorer performances of TwinVQ at 16 kbps , MPEG-1 L III at 16 kbps , and MsAudio at 16 kbps for item 3.

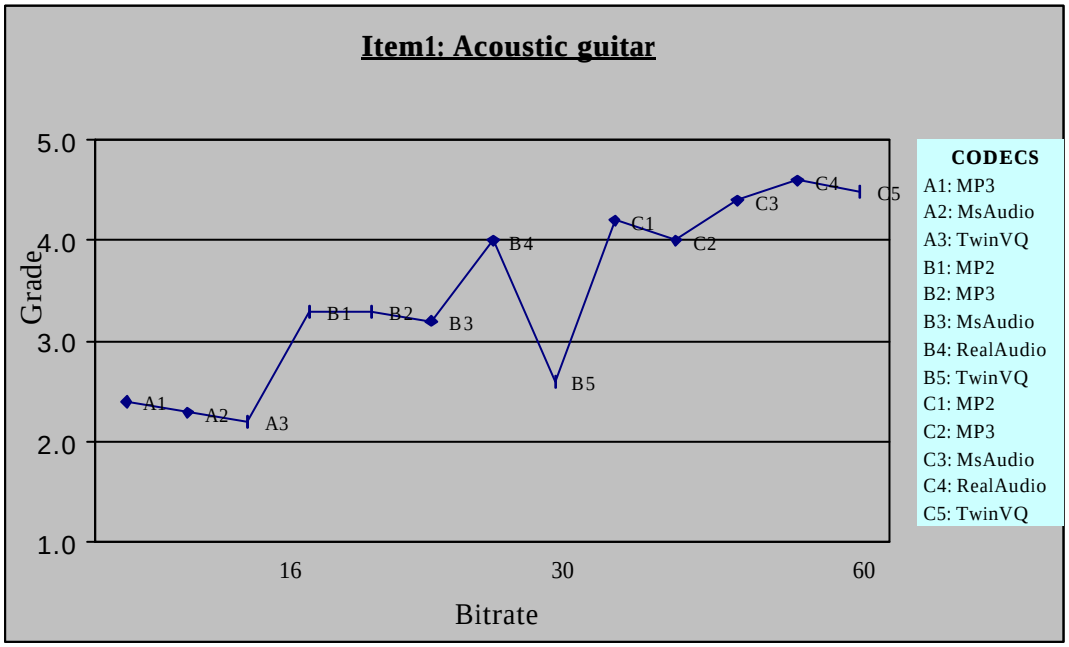

Figure 6: Overall quality by individual codecs for item 1

Table 10 gives the different equivalent groupings of the individual codecs for the acoustic guitar test material in order of merit.

Grading	Acoustic guitar
(5) Excellent	RA/44
(4) Good	L II/56, L III/56, MA/40, VQF/40, RA/20
(3) Fair	L II/24, L III/16, MA/16 VQF/16
(2) Poor	L III/8, MA/8, VQF/8
(1) Bad	

Table 10: Equivalence groupings of the individual codecs for item 1

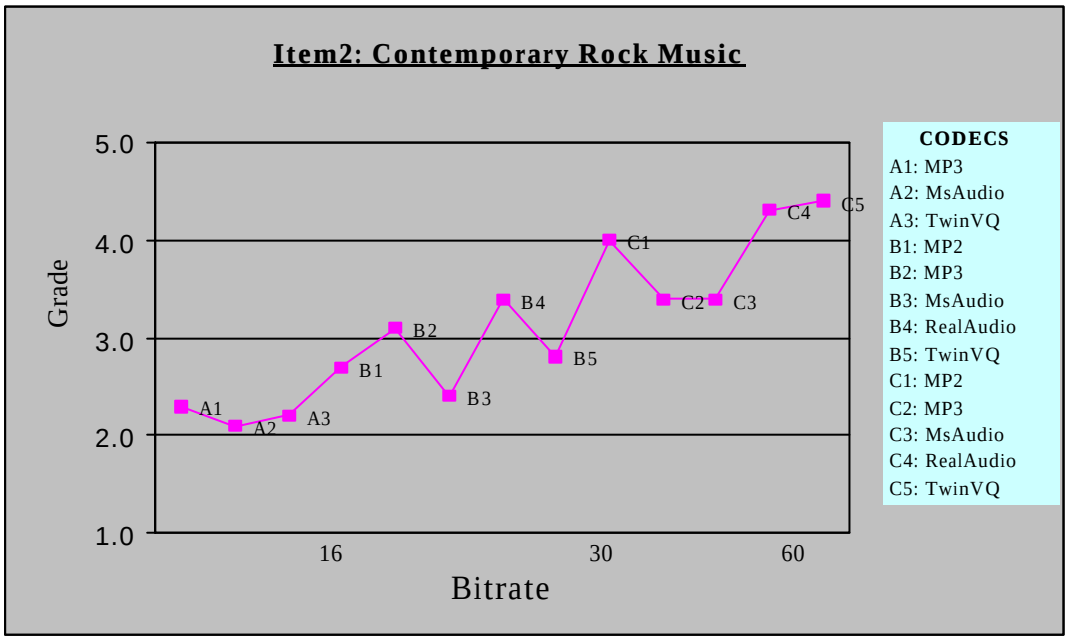

Figure 7: Overall quality by individual codecs for item 2

Table 11 summarizes the different equivalent groupings of the individual codecs for the rock piece test material in order of merit.

Grading	Contemporary rock piece
(5) Excellent	
(4) Good	L II/56, L III/40, RA/44, VQF/40
(3) Fair	L III/40, MA/40, L II/24, L III/16, MA/16, VQF/16
(2) Poor	MA/16, L III/8, MA/8, VQF/8
(1) Bad	

Table 11: Equivalence groupings of the individual codecs for item 2

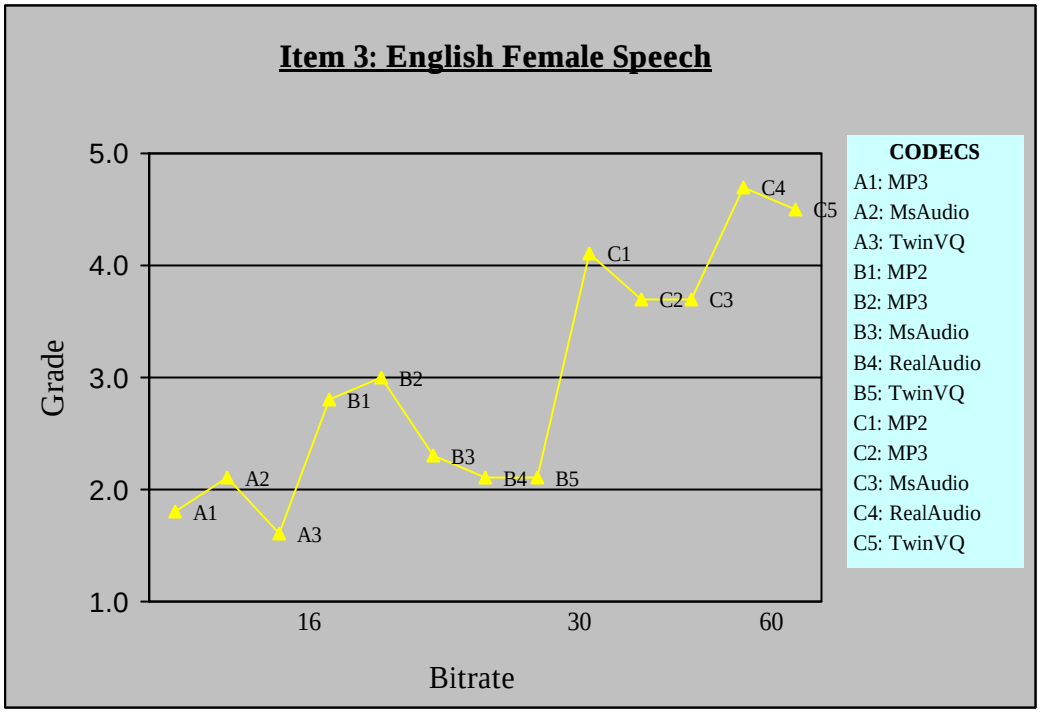

Figure 8: Overall quality by individual codecs for item 3

Table 12 gives the different equivalent groupings of the individual codecs for the female voice speech test material, in order of merit.

Grading	Female voice
(5) Excellent	RA/44, VQF/40
(4) Good	L II/56, L III/40, MA/40, VQF/40, RA/20
(3) Fair	L II/24, L III/16
(2) Poor	VQF/16, L III/16, MA/16, L III/8, MA/8, VQF/8
(1) Bad	

Table 12: Equivalence groupings of the individual codecs for item 3

Summary

In this chapter, digital audio compression and the two compression schemes, lossy and lossless compression, were discussed and analyzed. Also, the current audio codecs suitable for the Internet were presented. These included MPEG-1 Layer I, II, and III, MPEG-2 AAC, TwinVQ, RealAudio G2, and the new MPEG-4 standard.

This chapter also presented some experimental tests on the performance of these codecs at bit rates suitable for the Internet. The results of the subjective tests, evaluating the audio quality of the 13 audio codecs tested, were presented. The tests are unique since this was the first time in which all of these codecs have been compared in a single formal test. The tested codecs operated at bit rates between 6 and 56 kilobits per second. The tests examined the performance of the codecs under "low bandwidth" conditions and were conducted in accordance with the methodologies outlined in ITU-R Recommendation BS.562.3. The results show that the tested codecs are clearly delineated with respect to quality. The overall ranking of the individual codecs with respect to quality in order of merit is:

(1) RealAudio/44 and TwinVQ/40
(2) MPEG-1 LII/56, MPEG-1 LIII/40, and MsAudio/40
(3) MPEG-1 LII/24, MPEG-1 LIII/16, and RealAudio/20
(4) MsAudio/16 and TwinVQ/16
(5) MPEG-1 LIII/8, MsAudio/8, and TwinVQ/8

The tests reported in this chapter only considered audio quality with respect to bit rate.

Chapter 4

Streaming Audio Protocols

One of the major attractions of an audio store on the Internet is to provide excerpts from its collection in real-time. This feature allows users of the system to preview a short audio sample of the original musical item they are interested in before purchasing it. The main problem with this feature is the Internet bandwidth, especially when considering on-demand audio. Many professional Internet users have access to the net with wide-band connections. But private consumers usually use low-band connections with phone line modems with 33.6 kbps or ISDN mid-band access with 56/64 kbps. This poses a problem for applications that need to provide high quality on-demand audio services, especially when considering that one of the aims of on-demand audio is to use the low transmission rates available in the most effective way. One solution to this problem is to investigate and use audio codecs that are suitable for the Internet, as discussed in the previous chapter.

In this chapter the use of streaming audio technology as an added solution to the bandwidth limitation problem will be discussed. Audio streaming and some technical issues that go with it will be described first, and then the enabling technologies will be discussed, followed by a description of how the support for audio streaming was provided in WAPS. Finally the chapter concludes by discussing the experimental procedure taken and test results for the performance of various audio transport protocols supported by WAPS.

4.1 Streaming Audio

The traditional method of "bulk" transfer (i.e. transferring the entire file before playback) for audio files has the drawback of lengthy delay before the user can start the playback. This becomes a major problem when considering a large music site that allows electronic sale of thousands of digitized musical items and provides short audio excerpts for each of these items. Typically, audio files have an average transfer ratio of 5:1 on dial-up connections [23].

This implies that one must spend 5 minutes downloading a file that contains 60 seconds of audio. To reduce this delay to an acceptable level (less than 10 seconds) regardless of the file size, streaming technology is necessary. An audio stream is data that is sent from a server to a client at a controlled rate defined by the server [25]. This means that even if more bandwidth is available, the data rate will not exceed the predetermined rate. This ensures that the server's sending rate does not exceed the client's ability to handle the data. In audio streaming, an audio file is broken down into small chunks (data that normally contains just a fraction of a second's worth of sound). Then, each packet is individually transmitted over a suitable protocol to the client. Upon arrival, the client may then begin the playback after receiving the first few packets, rather than waiting for the entire file. With this approach, the listener does not have to wait an arbitrarily long time before playback begins. Implementing a bi-directional communication between the client and the server gives the client the ability to randomly access different portions of the audio file.

To support the rate of flow of packets, the bandwidth between the server and the client must be high enough. For example, playing uncompressed CD quality music in real-time requires bandwidth in excess of 1.4 MBPS (i.e. 44100 samples/sec x 16 bits/sec x 2). This is simply unrealistic for a client connected to the net via a 28.8 Kbaud modem, which would take 49 seconds to download one second of audio. Clearly audio compression is required in order to support real-time delivery of audio. Hence the selection of audio codecs suitable for the Internet, as discussed in Chapter 3, is an important factor in audio streaming.

4.2 Common Architecture for Streaming Audio Applications

When delivering real-time audio over the Internet, streaming audio applications employ a common architecture. This architecture is illustrated in Figure 1.

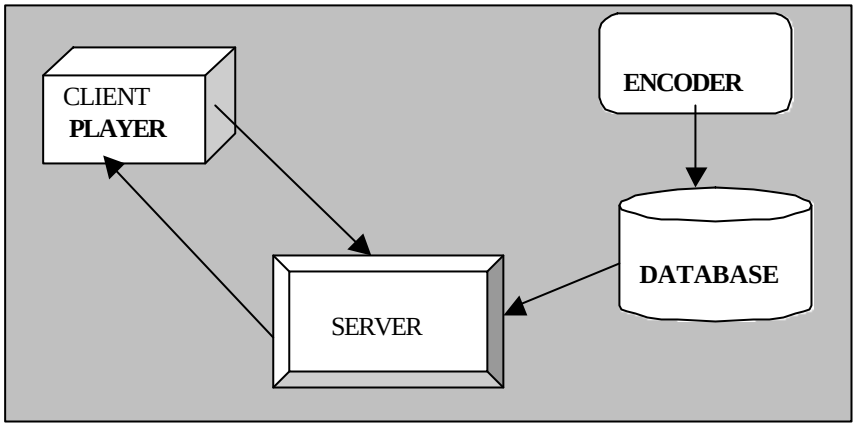

Figure 1: Common architecture

There are three components to the architecture:

- The Encoder

This encodes the audio data into a compressed format that can be efficiently transmitted over the Internet. This process is typically done off-line and the encoded files can then be catalogued and stored in a database.

- The Client

The client application contains a player that decompresses (decodes) in real time the encoded data streams into a format that can be played back on the client's computer. Hence the server and the client should have a matching encoder/decoder pair (codec) for audio to be successfully transferred and rendered.

- The Server

The server program is responsible for delivering audio data, controlling the flow of audio data, and some other tasks related to performance monitoring and control of audio streams. Most streaming audio applications implement the server program within a normal HTTP server. Examples of audio applications that do this are: Truespeech, Internet Wave, Video Live, RealAudio, etc. All these player applications use any HTTP server for audio delivery during playback. However, some other audio streaming applications implement a dedicated server program to offer more functionality and user capacity; these include RealAudio that uses the RealServer.

4.3 Technical Challenges Associated with Audio Streaming

Several technical challenges have increased the difficulty of real-time audio delivery over the Internet. These include: coping with packet losses, bandwidth limitations while having to meet minimum audio quality requirements, measuring the available bandwidth between the client and the server application, and the need for specialized protocols.

4.3.1 Bandwidth Limitations

One of the goals for audio delivery over the Internet is to use the low transmission rates available in the most effective way [26]. The subjective quality must be kept as high as possible with the given bit rate. As mentioned earlier, the real-time playback of CD-quality music requires more than 1.4 MBPS of bandwidth. However, with the majority of Internet users still connected to the net with low bandwidth connections, bandwidth limitation is the major factor that hinders the growth of on-demand audio applications. All on-demand audio applications require real-time traffic. This means that audio data must be played back continuously at the rate at which they are sampled. If the data does not arrive in time, the playback process will stop and the human ear can easily pick up the interruption. One major contributing factor to this delay is network congestion. If the network is congested, real-time data becomes obsolete and data packets will be dropped. On the other hand, for non real-time traffic, this is not such a problem since it simply means that the data transfer takes longer to complete.

4.3.2 Bandwidth Measurement

Several Internet streaming audio applications could benefit from knowing the bottleneck bandwidth of a route. This could help judge the efficiency of the utilized protocols and audio formats. For example, if an audio server is delivering audio data at close to the bottleneck bandwidth, then increasing the bandwidth of that link may increase the application performance. However, if the bottleneck link already has plenty of bandwidth to spare, increasing its bandwidth will probably not improve application performance. The bottleneck bandwidth of a route is the ideal bandwidth of the lowest bandwidth link (the bottleneck link) on the route between two hosts

[30]. The current bandwidth measurement techniques include: Pathchar, and Packet Pair algorithms.

- **Pathchar**

Pathchar is a tool written by Van Jacobin [49] that tries to estimate latency and the bandwidth of a route by sending UDP packets from a single source and measuring round-trip times. With Pathchar, packets of varying sizes are sent by a sender program to a receiver program, and the round trips of each of these packets are measured. Then, the round trip times are correlated with the packet sizes to calculate bandwidth. The problem with Pathchar is that it can consume a significant amount of network bandwidth. In particular, Pathchar runs in time proportional to the round trip time of the network and sends a varying sized data packets regardless of the actual bandwidth of the network. For example, if more hosts were to run Pathchar, its packets would become a significant burden on the network. Even for isolated hosts with low bandwidth connections, Pathchar could consume too much bandwidth to be usable regularly.

- **Packet Pair algorithm**

This algorithm relies on the fact that if two packets are queued next to one another at the bottleneck link, then they will exit the link *t* seconds apart:

$$\mathbf{t = S_2/b_{bn1}}$$

where $\mathbf{S_2}$ is the size of the second packet and $\mathbf{b_{bn1}}$ is the bottleneck bandwidth. There are two types of Packet Pair algorithm: Receiver and Sender Based Packet Pair. These two differ in how the $\boldsymbol{t}$ from the above equation is measured. Figure 2 shows the difference in where timing measurements must be taken for the two packets.

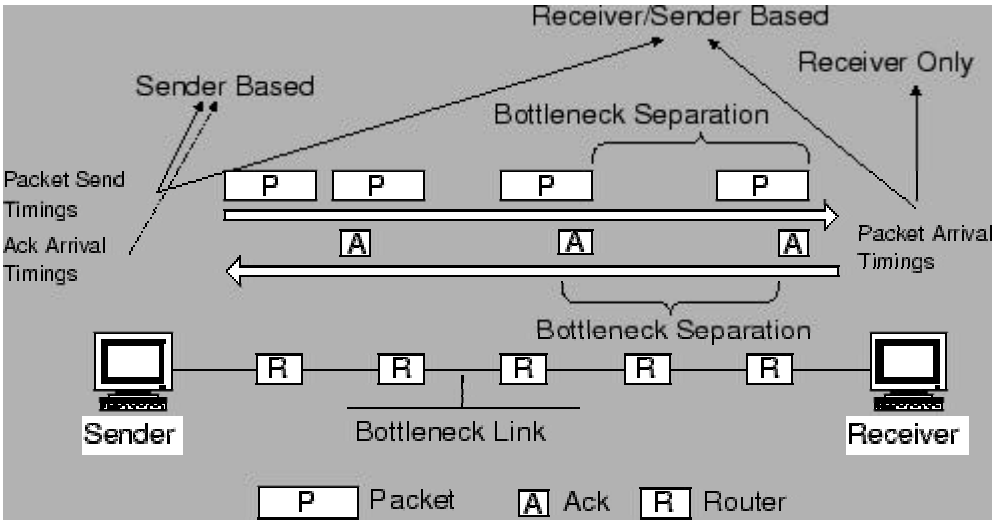

Figure 2: How Packet Pair works

The arrows pointing to SBPP (Sender Based Packet Pair), RBPP (Receiver Based Packet Pair), and ROPP (Receiver Only Packet Pair) indicate what timing information must be sent from the sender and receiver. The two packets have to be the same size because different size packets have different "velocities". For example if the second packet were smaller than the first, then its transmission delay would always be less than the first packet's. Consequently it would pass through links faster than the first packet and quickly eliminate the bottleneck separation. Similarly, if the first packet is smaller, then it will be faster than the second packet and continuously grow the bottleneck separation. Assuming the bottleneck separation is constant, the two packets will arrive at the receiver spaced t seconds apart. Since we know S_2, we can then calculate the bottleneck bandwidth as in the following equation:

$$b_{bn1} = S_2/t$$

In Receiver Based Packet Pair, t is measured at the receiver, hence the equation becomes

$$b_{bn1} = S_2/(a_2 - a_1)$$

where **a1** and **a2** are arrival times of the first and the second packets, respectively. If the arrival times cannot be measured at the receiver, the round trip time is used, which is measured at the sender (SBPP). Now the equation becomes:

$$b_{bn1} = S_2/(r_2 - r_1)$$

where **r1** and **r2** are the arrival times of the acknowledgements to the first and the second packets, respectively. SBPP is easy to deploy, but its results can be highly inaccurate during congestion. Also, the SBPP approach requires that packets be

acknowledged and that the acknowledgements be constant size and relatively small. This is because variation in the acknowledgement's size causes variation in the total round trip time, which would cause noise in the bandwidth samples. The acknowledgments need to be small in size because if they become larger, the bandwidth of the path back to the sender would start to become the bottleneck.

4.3.2.1 The Implemented Algorithm

In the WAPS audio communication component, the audio player first measures the bottleneck bandwidth of the route between the audio server and itself before actually requesting the audio clip from the audio server. The player then uses this information for deciding which codec e.g. MPEG-1 LIII at 16 kbps or RealAudio at 40 kbps, would be suitable for that particular bandwidth. The idea here is to encode the short audio excerpts at different bit rates in such a way that f the bottleneck bandwidth of the route between the audio player and the audio server is saturated, then an appropriate bit rate is selected so as to provide smooth playback at reasonable quality. For measuring the bandwidth of the route between the distributor's (ILAM) server and the user's machine, the Receiver based Packet Pair bandwidth measurement algorithm was implemented. This algorithm was implemented in the WAPS audio communication component using a client/server approach, whereby the receiver was a Java thread class named *BandwidthClient*, implemented by the audio player, and the sender was also a Java thread class named *BandwidthServer*, executing as a daemon process on the merchant's server. Three pairs of 1024 bytes sized data packets are sent from the sender to the receiver and their arrival times are then recorded by the sender. The bottleneck bandwidth is then calculated for each pair and the average of the three pairs is used as the final bandwidth of the route between the sender and the receiver.

Using this information, the audio player can then recommend the bit rate of the audio clip to be delivered by the audio server. For example, the music distributor can have two 30 second audio clips of one musical item on sale whereby the first clip would be encoded at 16 kbps while the second excerpt would be encoded at 40 kbps (using an MPEG-1 Layer II encoder, these two clips would be saved as **artistR102_16.mp2** and **artistR102_40.mp2**, where **artist** is the name of the artist, **R1** the album release, **02** the track number, while **16** and **40** are two bit rates at which the excerpts have been

encoded). Then, if the measured bottleneck bandwidth falls, say, below 40 kbps, the audio player will request the first clip, otherwise it will request the second clip.

4.3.3 Jitter Control

During real-time audio transmission, the client ideally needs to have regular arrival of audio packets so that the audio can be played back smoothly. However, in many instances this does not happen, as packets sent by the server at regular intervals tend to arrive at an irregular rate. This is mostly caused by varying delays that depend on changing congestion levels at different routers along the way [25]. This phenomenon is called jitter, and it normally results in choppy sounding audio. The jitter problem is usually solved by using a "play-out buffer" at the receiver end. Using this buffer, a player stores the first several packets of data into the buffer before starting playback. As new packets arrive from the server, they are queued into the buffer, and playback continues from the buffer.

4.3.4 The Need for Specialized Protocols

Streaming audio technology developers have to choose between two packet transport protocols, of which neither is particularly effective for real-time data delivery. Some rely on the User Datagram Protocol (UDP) to send packets as quickly as possible, but with no guarantee of delivery. Others use the Transmission Control Protocol (TCP), ensuring that all packets will arrive in proper order. The Internet has been primarily used for the reliable transmission of data with minimal or no delay constraints [27]. The TCP protocol was designed for this type of traffic and it works very well in this context. However, audio traffic possesses different characteristics and hence requires the use of different protocols to provide the necessary services. For example, if an audio receiver has to wait for a TCP retransmission, there can be noticeable and unacceptable gaps in play-out of the real-time data. In addition, the "slow start" TCP congestion control mechanism can interfere with the audio "natural" play-out rate. Also, TCP does not provide timing information, which is a critical requirement in on-demand audio.

On the other hand, UDP sends packets as quickly as possible to the destination and does not bother to retransmit lost packets. Instead, given that some packets may

be lost, the receiver may try to fill the gaps with appropriate space fillers. Since the type of compression used determines the quality of real-time audio and the amount of data lost, the audio quality of UDP-based applications will degrade accordingly when the net becomes congested and packets get lost.

Using either of these two transport protocols, packets frequently do not reach the destination in time to achieve smooth audio playback. This has resulted in a number of protocols being developed to enhance the Internet architecture and improve support of applications that make use of audio and video. These protocols include: the Real-time Transport Protocol (RTP), Real-time Control Protocol (RTCP), Real-time Streaming Protocol (RTSP), and many other proprietary protocols such as the Progressive Network Audio (PNA) protocol. A brief description of these protocols is given below.

4.3.4.1 RTP/RTCP

The Real-time Transport Protocol (RTP) and its companion, Real-time Control Protocol, defined in RFC's 1889 and 1890, specify a standard for transmitting time dependent data such as audio or video streams [28]. RTP provides end-to-end delivery services to support applications transmitting real-time data. It was primarily designed for multicasting real-time data, but can also be used in unicasting mode. Among the services that RTP provides are payload type identification, packet sequence numbering, and time stamping. The RTP specification defines the use of translators that can convert an RTP stream from one format or bandwidth to another, and mixers, which can pull multiple streams into a single stream to conserve bandwidth. RTP primarily runs on top of UDP, thereby utilizing its multiplexing and checksum services.

However, other transport protocols besides UDP can carry RTP packets as well. UDP was chosen as the target transport protocol for RTP for two reasons: First, RTP was primarily designed for multicast, so the connection-oriented TCP does not scale well and therefore is not suitable. Second, for real-time data, reliability is not as important as timely delivery. Hence, TCP is not a suitable candidate due to its guaranteed delivery of packets. RTP does not provide all of the typical functionality of audio transport protocols. It does not ensure timely delivery or provide Quality of

Service (QoS) guarantees. It does not guarantee delivery or prevent out-of order delivery, nor does it assume that the underlying network is reliable. The Real-time Control Protocol is used to monitor the delivery of RTP packets. RTCP ensures a channel of communication between the server and client applications so that proper adjustments can be made as needed to facilitate delivery. It provides feedback information about the quality of the transmission to the applications.

The statistics include the number of packets sent, the number of packets lost, interval jitter, etc. RTCP also identifies the RTP source address through its transport-level identifier called the canonical name (CNAME). The CNAME is used to keep track of participants in a RTP session in order to synchronize audio and video. RTCP controls its transmission intervals in order to prevent control traffic from overwhelming network resources. This control allows RTP to scale up to a large number of participants. An optional function can be used to convey a small amount of information to all session participants.

4.3.4.2 RTSP

The Real-time Streaming Protocol is a client-server multimedia presentation protocol that enables controlled delivery of streamed multimedia over IP networks. RTSP is an application level protocol designed to work with lower level protocols, such as RTP, to provide a complete streaming service over the Internet [28]. It is intended to:

- control multiple data delivery sessions
- provide a means for choosing a data transport protocol such as UDP, TCP, or IP multicast.

RTSP was designed to have a similar syntax and mode of operation as HTTP. For instance, in RTSP, an RTSP URL identifies each presentation. The following are some of the features of RTSP:

- It is an application level protocol with syntax and operations similar to HTTP, but works for audio.

- It uses URLs like those in HTTP.

The RTSP protocol itself is different from the data protocol, e.g. may use RTP as its data protocol.

- It is a bi-directional protocol, i.e. both the server and client can issue requests.

- It allows interoperability between clients and servers from different manufactures.

RTSP provides "VCR-style" remote control functionality for audio and video streams. These include: pause, fast-forward, reverse, and absolute positioning. RTSP establishes and controls streams of continuous audio and video media between a server and a client. For example, an audio server may provide transmitting or recording services for audio streams, while a client may request continuous audio data from the audio server. RTSP can be considered more of a framework than a protocol. In many respects, it resembles a "network remote control" between a server and a client, allowing the retrieval of media from a media server, i.e. the client can request a presentation description and ask the server to set up a session to send the requested data.

4.3.4.3 PNA

Progressive Networks Audio (PNA) is a proprietary client/server protocol designed and used by Real Networks in the RealSystem (i.e. RealAudio and RealServer). PNA is one of the two main protocols (the other being RTSP) used by the Real Server G2 system to communicate with its clients. This protocol implements a two-way TCP/IP connection for sending commands, such as "start and stop" between the client and the server and uses a one-way UDP channel, which is separate from the TCP channel, for carrying audio data. For packet loss, PNA employs what is known as forward error correction, where each packet sent to the client contains some information about the previous packet. This technique is based on the observation that on the Internet the probability of losing multiple packets in a row is relatively low compared to the probability of losing a single packet. The PNA protocol also uses another technique called sample interleaving for reducing the impact of packet loss. With this technique, about 3 seconds of sound is divided into 144 bundles of 20 milliseconds each. The bundles are then allocated to 12 packets where the first, thirteenth, twenty-fifth, and

so on go into the first packets, and the second, fourteenth, twenty-sixth and so on go to the second packet, etc. Figure 3 illustrates this technique.

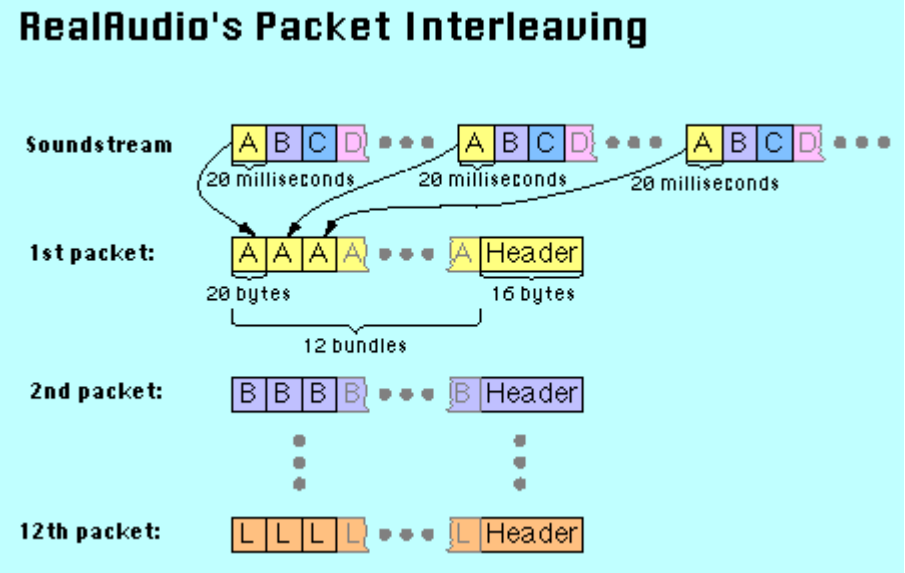

Figure 3: Sample interleaving

The sample Interleaving technique does introduce some computational overhead during encoding and decoding, however its advantage is that if one packet is lost, the receiving client application does not loose 240 milliseconds of continuous audio, rather it loses 20 milliseconds about every quarter of a second.

4.4 Implemented architecture for the WAPS audio component

The WAPS audio communication component provides the means for the real-time transmission of audio clips between a client and a server application. Intel's Java Media Framework (JMF) API [29] was used for developing the audio component, in particular the client (player). The audio component is based on a client/server architecture, whereby the client application is an audio player that decompresses (decodes) the encoded data streams into a format that can be played back on the client's computer, and the server application (audio server) is responsible for delivering audio data to the player. The WAPS audio component architecture implements the common architecture for streaming audio applications, described in

section 4.2. Thus, the WAPS audio communication component architecture is made up of the following sub components:

- **The Encoders**

As mentioned in section 4.2, the encoders are used to encode (compress) the short audio clips of the musical items into a compressed format that can be efficiently transmitted from the audio server to the audio player in real-time. The process of compressing the audio clips was done off-line. Each musical item on sale was encoded into a 30 second audio clip, which was then stored on the local hard drive of the merchant's (distributor's) computer. These short audio clips were catalogued along with their parents (original high quality) musical items into the audio storage server. Each audio clip consisted of a unique name that corresponded to its parent. This name becomes a pointer or link and is then stored in the database for retrieval purposes. For example a parent musical item could be named *mbandeR102.wav*, where *Mbande* is the name of the artist *R1* the album release, and *02* being the track number in that particular album. Now, the audio clip corresponding to this track would be *mbandeR102_BT.mp3,* where *BT* would be the bit rate at which the audio clip was encoded at. The use of the *BT* allows for the encoding of multiple audio clips of the same track but at different bit rates targeting different Internet connections.

For the encoding of the audio clips, several industry standard audio encoders are supported. These include: the QuickTime audio encoder, MPEG-1 Layer II and III encoders, Microsoft Windows Media encoder, Real Producer, and CoolEdit Pro (for SuN/NexT audio – AU). The use of these encoders corresponds to the decoders currently supported by the WAPS audio player (as indicated in section 3.4.4 of Chapter 3). Providing support for these different coders does not necessarily mean that all of them have to be used at one time. The music distributor (such as ILAM) will have to decide which codecs their music stores will utilize. They can then encode the audio clips using that encoder, which may be decoded by the matching decoder pair supported by the WAPS audio player.

- **The Client (audio player)**

This is a continuous audio playback application that runs on a Java capable browser to decompress the encoded data streams on demand using an appropriate transport protocol. Figure 4 shows a snapshot of the WAPS audio player.

Figure 4: The WAPS audio player

The audio player forms the control interface of the audio component. It allows users to browse and control audio clips. The following user controls are provided by the player:

- *Play*: Start to play the music sample.
- *Stop*: Ends the playing of the sample. If user presses Play, start playback from the beginning.
- *Fast Forward*: Skip and play the audio on another position.
- *Rewind*: Play the audio from a previously played position.
- *Pause*: Stop playback for a while until user presses play again, then start playback at exactly the same position

- **The Server**

This is responsible for delivering audio data to the remote audio player. The audio component integrates with some of the industry standard audio servers (such as RealServer G2 and MPEG-3 server). An application specific server was developed for transporting static audio files, such as SuN/NexT (AU), on-demand. The audio component also makes use of any HTTP server for transporting audio data to the player.

For transporting audio data between a client and a server application, specialized protocols are needed. The WAPS audio component provides support for both the standard and proprietary audio transport protocols. These include HTTP, RTSP, and PNA, which have been described in section 4.3.4 of this chapter. An application

specific protocol was designed and implemented, named Simple Audio Transport Protocol (SATP), for transporting static audio files, such as AU files and WAV files, between the audio player and a server in real-time. Figure 5 illustrates the overall client/server architecture of the audio component. This figure also illustrates the two physical components (audio player and supported audio servers) of WAPS that form the audio component along with the various audio streaming protocols used.

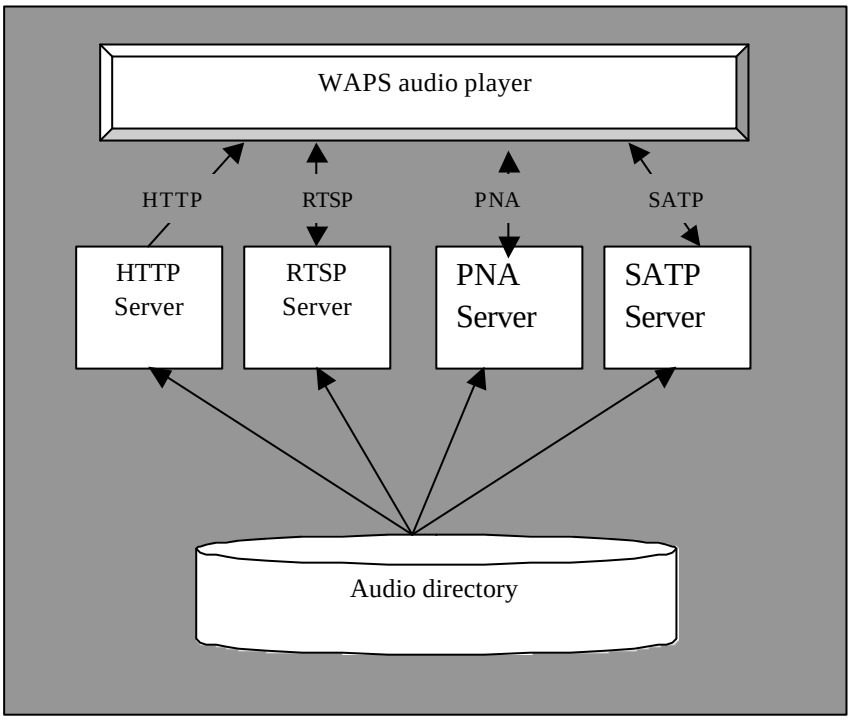

Figure 5: WAPS audio component

4.4.1 The Audio Component's core Classes

The following Java classes form the core of the WAPS audio communication component:

- *ControlInterface*: is a graphical user interface that provides buttons for playing, browsing, and stopping the player.
- *BandwidthServer*: is a server thread that implements the server side of the bandwidth measurement algorithm.
- *BandwidthClient*: is a thread that implements the client side (receiver) of the bandwidth measurement algorithm. It receives sample data from the BandwidthServer.

- *ProtocolManager*: manages the supported transport protocols, decides as to which protocol to use, and instantiates the appropriate classes to receive audio data over the selected protocols.
- *HTTPclientControl*: receives audio data carried via HTTP from an HTTP server and sends it to the audio device (sound card) for playback.
- *RTSP/PNAControl*: receives and renders audio data transported by the RTSP and the PNA protocols.
- *PlugInManager*: launches helper applications that are supported by the audio player e.g. RealAudio G2, Windows Media Player, etc.
- *SATPClientControl*: receives and renders audio data from the S*ATPServe*r.
- *SATPServer*: sends audio data (AU files) to the SATPClientControl.

Figure 6 illustrates the implemented class files and shows how they interact.

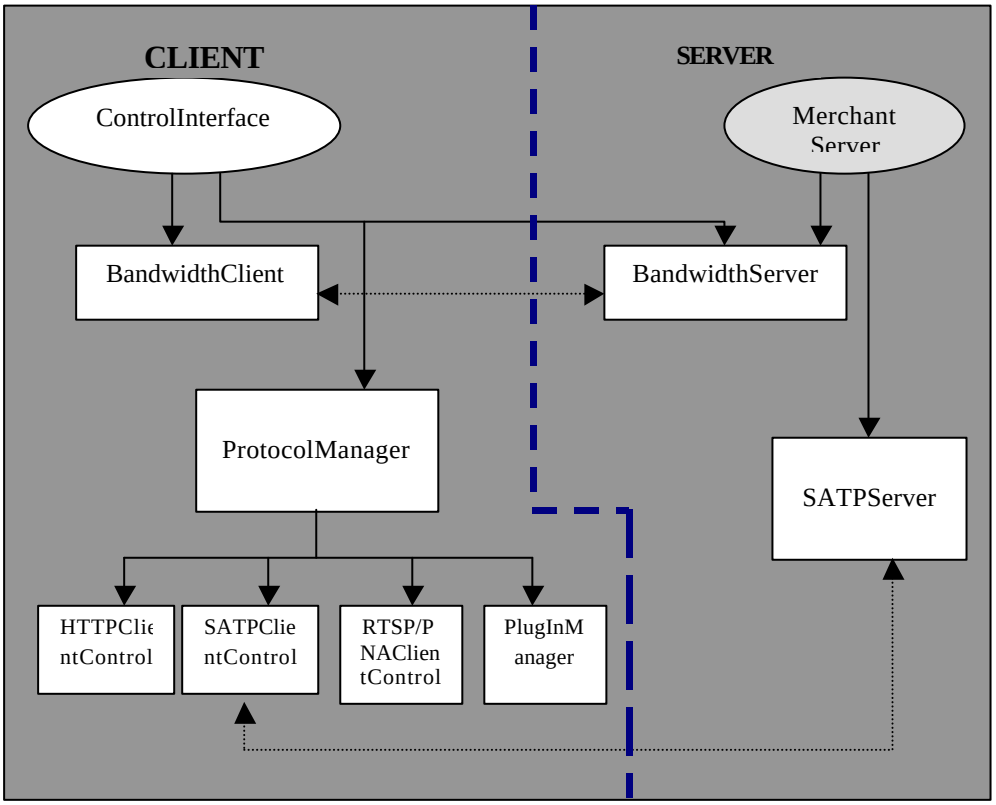

Figure 6: Implemented class files

4.4.2 The message flow within the audio communication component

The following are a series of steps that occur when a user wants to preview a music sample through WAPS:

1) User browses via Information Retrieval Engine and clicks the "*preview*" button, which is linked to the audio player.

2) The audio player is launched and is forwarded the URL of the audio clip it has to render by the Information Retrieval Engine. This URL contains the names of the available servers that can be used to stream the clip, address of the machine where the audio clip physically resides, and the name of the audio excerpt. The format of the URL is shown in Table 1 below.

SVR1:SVR2:SVR3:\\servername\filename

Table 1: URL format

SVR1, SVR2, SVR3, etc give the available audio servers on the merchant's computer, which can be used to stream the audio clip. S*ervername* is the host name, and *filename* is the name of the audio clip to be played. For example, if a merchant uses the RealServer G2 and an HTTP server for serving audio data to the player, the sample URL would be: **G2:HTTP:\\ilam.ru.ac.za\ mbandeR102_BT.rm.** Then, given this URL the audio player would choose which server and transport protocol to use depending on whether that media type is supported by that particular transport protocol.

3) Once the audio player has identified the URL, it then estimates the bottleneck bandwidth between itself and the server. The audio player uses the BandwidthClient object to open a TCP connection to the BandwidthServer object on the merchant's computer. This connection is then used to estimate the bandwidth between the player and the server and an appropriate clip encoded at the relevant bit rate will be selected. For example, the player will modify the above URL to **G2:\\ilam.ru.ac.za\mbandeR102_40.rm,** indicating that the RealServer G2 system will be used to send the clip to the player and that the bottleneck bandwidth available is enough for a clip encoded at 40 kbps.

4) Player requests the music sample from audio server and waits for audio data.
   Based on the results of the estimated bandwidth, the player chooses a suitable transport protocol, makes a connection to the audio server, and requests transmission of the audio sample.

5) Audio server receives the request, retrieves the audio file, and streams it to the player, and playback resumes immediately after a small amount of data has been received.

Figure 7 illustrates the internal structure of the audio component and shows how the various messages flow between the different components.

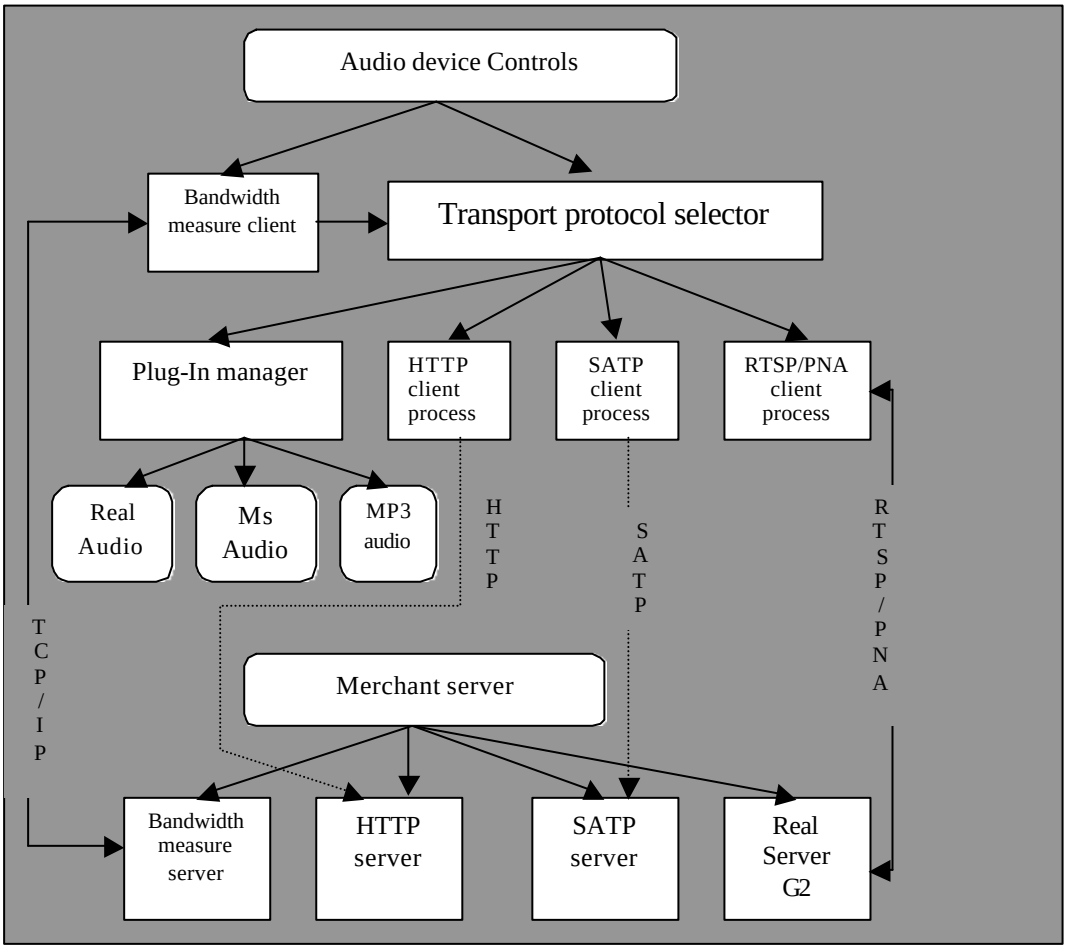

Figure 7: Internal structure of the audio layer

4.4.3 Implemented transport protocols

The following audio transport protocols were implemented by the audio player for rendering audio data from the various supported audio servers. These protocols, which include HTTP, RTSP, PNA, and SATP, form the core of the audio communication between the WAPS audio player and the audio servers supported.

4.4.3.1 HTTP

The Hyper Text Transfer Protocol, which is already used by all web servers to store and transmit ordinary text and graphic files can also be used for the transmission of on-demand audio. When using an HTTP connection to download audio files in certain 'streamable' formats (e.g. MPEG, RealAudio, Ms Audio, 'fast-start' QuickTime, etc) the audio player can begin playback before all of the audio data is received. This is called HTTP streaming. HTTP streaming has been transparently implemented in the JMF API using the javac.media.Player object, which is also used for the playback of the audio data transported over HTTP. The *Player* class resides in the javavax.media class library of the JMF API. A separate class named HTTPClientControl, defined in section 4.4.1, was implemented as part of the audio player to provide support for HTTP streaming. HTTPClientControl imported the *javavax.media* class library and inherited from the Player class of this library. When the HTTPClientControl class is instantiated by the ProtocolManager (also defined in section 4.4.1) of the audio player, it gets forwarded the URL of the audio sample to be rendered. Using this URL, it then calls the Player.start() method, which makes a connection to the HTTP server and waits for the audio data.

The *Player* will attempt to begin playback as soon as it has buffered a small amount of data. If the download rate is not high enough to keep up with the rate at which the *Player* is using data, an under-run condition occurs. For example, if an MPEG file is being downloaded over a low bit rate modem connection, and the download cannot keep up with playback, then the *Player* will run out of data. The default behavior of the *Player* in an under-run condition is to generate a DataStarvedEvent. This event is intercepted and playback paused so that the *Player* continues to download data waiting until enough data has been received to begin playback. The following code fragment shows how the *Player* object was used

```
import javax.media.*;
 ……
  try{
    //Create a URL from the file forwarded by the ProtocolManager, e.g. the file could be
    //   Http://ilam.ru.ac.za/mbandeR102.mp2
    URL mediaURL = new URL( sampleURL);
    // Create an instance of an appropriate media player for the audio type requested
    Player player = new Manager.createPlayer(mediaURL);
    ……
    // Start the player
   player.start();
  }
catch(MalformedURLException e)
 {}
 ……
```

4.4.3.2 RTSP/PNA

The RTSP/PNAControl mentioned in section 4.4.1, an inner class of the audio player, was implemented to provide support for the RTSP and the PNA protocols using Real Network's JMF implementation in conjunction with the Intel's JMF. This class implemented exactly the same code used for the HTTPClientControl class. It also imported the javax.media class package and inherited from the Player class defined in this class package. The only difference between the HTTPClientControl and the RTSP/PNAControl classes is when specifying the URL of the audio sample to be played. The setting up of the RTSP or PNA connection is handled automatically by the Real Network's JMF API transparently from the application programmer. The following code illustrates how to connect to a server using both the RTSP and the PNA protocols:

```
import javax.media.*;
 ……
  try{
     //Create a URL from the file forwarded by the ProtocolManager, e.g. the file could be
     //  rtp://ilam.ru.ac.za/mbandeR102_16.mp2 or pna://ilam.ru.ac.za/mbandeR102_40.mp2
     URL mediaURL = new URL( sampleURL);
     // Create an instance of an appropriate media player for the audio type requested
     Player player = new Manager.createPlayer(mediaURL);
     ……
    // Start the player
    player.start();  }
catch(MalformedURLException e) {}
 ……
```

4.4.3.3 SATP

The Simple Audio Transport Protocol is an application specific protocol that was designed to stream static audio files (such as the Sun/NexT format or WAV files)

from a server process to a client process in real-time. SATP was implemented in Java and is made up of two objects: the SATPClientControl named 'client' and the SATPServer named 'server'. With this protocol, the client does not download any information from the server until such time that information is needed. For example, when the user presses the *play* button on the control interface, the client tells the server to initiate the flow of audio over the network to the client. With a little bit of buffering to smooth out packet arrival jitter, audio data is played as soon as it arrives over the network, and prior to that no audio data is ever downloaded to the client. The server is a multi-threaded stand-alone Java application. For each new request, it creates a new audio file server thread that in turn serves the audio data to the client.

By default the server restricts the number of simultaneous audio on-demand requests to twenty streams. However, this number can be increased at run-time. For decoding and playing the compressed audio, the client requires use of Sun Microsystems's AudioPlayer class in the sun.audio class library. The fact that these classes are housed in the *sun.** package hierarchy indicates that they are not part of the standard Java API and may change in future releases of the Java core. The AudioPlayer class requires an InputStream object from which it reads and plays ulaw encoded audio data until it reaches the end of the file. An InputStream object called FifoInputStream (which will be referred as FIS) was derived. This FIS is then submitted to the AudioPlayer class. The AudioPlayer class reads and plays from the FIS object. The FIS allows one write thread and one read thread. For controlling jitter, a download thread class was derived for receiving audio data from the server and depositing it into the FIS object. Figure 8 illustrates how jitter control works.

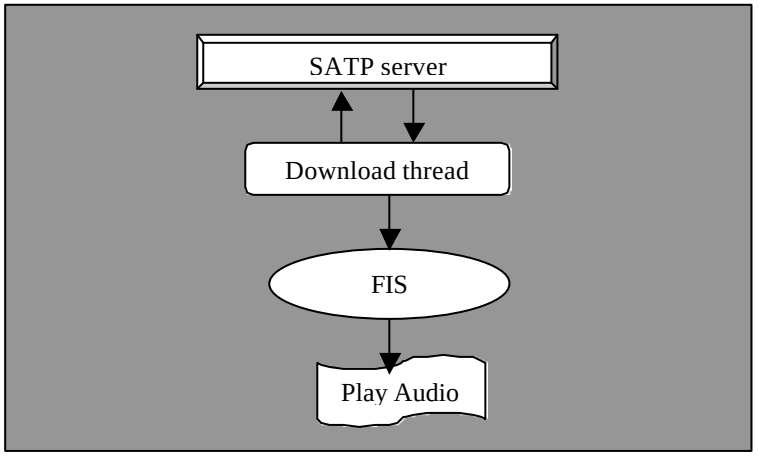

Figure 8: The client's jitter control

The illustrated solution proved to be successful especially on a LAN, but noticeable gaps could be heard when played over a 28.8 kbps modem connection. The following code fragment demonstrates how the client processes audio using the AudioPlayer class.

```
import sun.audio.*;
……..
{
 AudioStream audioStream = new AudioStream(new FifioInputStream());
 AudioPlayer.player.start(audioStream);
 ……………
}
```

SATP uses TCP to maintain a control connection to the audio server and for relaying user commands to the server process. This command channel carries an application specific VCR-type control prototype between the client and the server. For the actual audio data transport, SATP uses UDP. A three-way stop and wait handshake is used between the client and the server upon connection setup. The client process initiates the streaming audio connection by opening a TCP connection with the server, and sending a packet to the server containing the desired audio file's name. If the file exists, the server sends an acknowledgement back to the client containing the desired audio file's header information to be used by the client when creating initial buffers. The client then sends back an "acknowledgement received" packet to inform the server that the connection is established and is awaiting audio data via UDP. On arrival of the client's acknowledgement, the server then chops the requested audio file into chunks of 1024 bytes, packages it into UDP packets and sends it to the client. The client accepts the UDP packets through the download thread, which deposits them into the FIS object.

The main goal of SATP has been to facilitate the streaming and playback of static audio files (e.g. AU files). The baseline-encoding scheme for the AU files consumes 64 kbps of network bandwidth. This is not a problem for internal Ethernet networks or when using ISDN. However, to be able to transmit the data through a modem, clearly some form of audio compression is required.

4.4.4 Implemented support for Plug-in players

As mentioned in section 3.4.4 of Chapter 3, the WAPS audio player also provides support for an unlimited number of audio Plug-in applications, such as RealAudio player, Windows Media Player, etc. Launching an audio plug-in application through the audio player is achieved by constructing a java.net.URL object of the audio sample requested. The actual launching of the plug-in is handled via the showDocument method of the Java API. There are two forms of this method: one instructs the browser, from which the player is executing, to display a URL on the current window, and the other can be used to send the output of the URL to another target browser window. In this project, the latter approach was chosen so as to allow users to navigate through the Information Retrieval Engine, while at the same time controlling the player on a separate browser window. The plug-in support is implemented by the PlugInManager class mentioned in section 4.3.1 of this chapter. The following is the code that this class implements to launch the plug-in applications supported

```
if (useHelperPlayer)
{
 try{
   // construct a URL object of the audio sample to be played
   URL real = new URL ("rtsp:\\ilam.ru.ac.za\Samples\mbandeR102_40.rm");
   }
 catch(MalformedURLException e)
  {}
 // Launch the RealAudio helper application
 Applet.getAppletContext().showDocument(real);
}
```

This concept of plug-in players was used to provide support for a potentially unlimited list of audio formats.

4.5 Experimental tests

This section reports on the results of some real time playout testing activities, which were conducted on the transmission of the audio samples using the WAPS framework of a distributed audio database system. The main goal of these tests is to evaluate the applicability of the HTTP, RTSP, and PNA protocols to satisfying the real-time play-out requirements of compressed audio streams.

4.5.1 Testing scenario

The following environments seemed to be the most likely for the transport protocols:

a) LAN: >= 10 Mbits/s

A Local Area Network (e.g. based on Ethernet) offers bandwidth far greater than needed for the highest quality MPEG-Layer III file. It is even possible to retransmit lost or corrupted information, without delaying real-time playback.

b) WAN/dedicated connection

Typical for this environment is a single PC on a dial-up line, and bit rates of 14.4/28.8……n*64 kbps (n=1,2).

c) WAN/Internet

This is the most difficult scenario for synchronous audio playback. The Quality of Service can vary by orders of magnitude and is mostly unpredictable.

The comparative tests of the HTTP, RTSP, and PNA protocols were conducted for all the above three cases.

4.5.2 Testing environment

The testing environment described in section 4.5.1 above was set-up as follows:

- **LAN**

The LAN used was part of Rhodes University's Ethernet connected via a CISCO switch as shown in Figure 9. The load on the Ethernet during testing was roughly between 5% and 35 %.

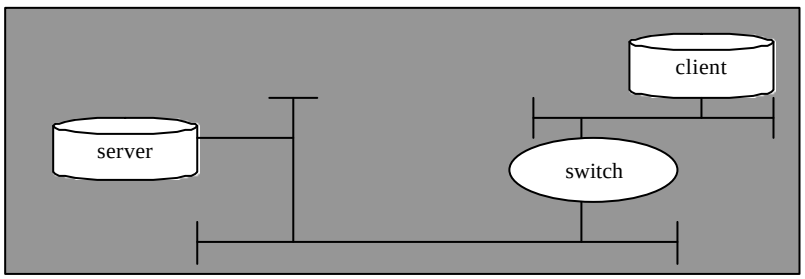

Figure 9: The LAN environment used

- **WAN**

  For this environment two computers were connected back-to-back using a crossover UTP cable and two 10 Mbits/s Ethernet cards were used, one on each computer, as shown in Figure 10 below.

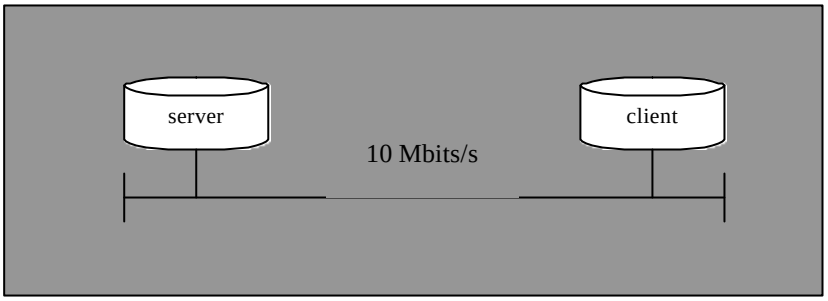

Figure 10: WAN environment

- **Internet**

A remote server was connected to the LAN via a 28.8 kbps modem connection, as shown in Figure 11.

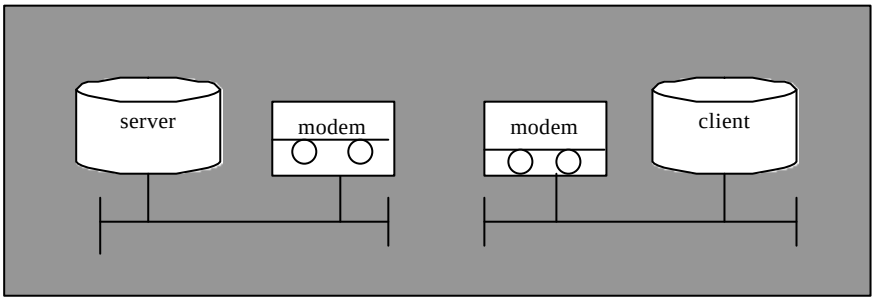

Figure 11: Internet environment

The tests consisted of transmitting an MPEG-1 Layer III audio stream from a producer (server) to a consumer (client) through the various networking environments. The producer is simply in charge of retrieving and transmitting the compressed stream, while the consumer decompresses and plays the audio as soon as new chunks of information are available. For the RTSP and the PNA protocols RealNetwork's RealServer G2 system was used as the producer, while the Microsoft's Internet Information Server was used for HTTP. For the playback of the compressed audio stream, the RealAudio G2 system was used and it was launched as a plug-in player through the PlugInManager object. For measuring the performance of the tested transport protocols, RealAudio G2's built-in protocol performance monitoring tool was used during the tests. These tests are based on one subjective evaluation of

the performance of the HTTP, RTSP, and PNA protocols as opposed to the audio listening tests (conducted in Chapter 3) which require a number of subjects in order to have decisive results due to the nature of the tests. In other words the audio listening tests were concerned with sound quality as opposed to performance, which was the main concern in the transport tests.

4.5.3 Experimental Results

As may be expected, no problems were detected when using the HTTP, RTSP, and the PNA protocols on both LAN and WAN network environments. There were no substantial delivery delays detected and on both the LAN and WAN environments the bandwidth was widely sufficient to put out smooth audio playback without any noticeable gaps between packets. On the Internet environments, the performances of all the tested protocols were greatly influenced by the available bandwidth. The tests revealed that particularly for limited bandwidth, the network capacity could be quickly saturated depending on the audio stream characteristics. When using HTTP, noticeable gaps could be heard during playback and the player had to pause several times to buffer more data. This was because HTTP uses TCP as its primary delivery protocol, and when packets get lost TCP retransmits them resulting in poor playback quality. On the whole, the playback when using the RTSP and the PNA protocols was much smoother than with HTTP, and RTSP performed better than the PNA protocol. Table 2 summarizes the approximate results for the 20 kbps RealAudio bit stream used.

Stream	Bit rate	Protocol	Bandwidth occupation
RealAudio	20 kbps	HTTP	21.2 kbps
RealAudio	20 kbps	RTSP	20.6 kbps
RealAudio	20 kbps	PNA	20.8 kbps

Table 2: Bit rates, HTTP, RTSP, and PNA

From the above table the performance values of HTTP appear to be better than those of the other protocols. This however, does not mean that HTTP is a preferred approach, since the other protocols offer more services than in HTTP, which include

intelligent buffering, jitter control and packet loss. The other protocols also offer bi-directional communication between client and the server.

Summary

In this chapter the major audio transport protocols, the technologies, and the various technical issues that need to be considered when developing systems that make use of audio streaming technology were discussed. These proven protocols make audio streaming over IP networks a practical solution to the limited bandwidth nature of the web. Also the implementation of these protocols in WAPS to facilitate the preview of short audio clips of the musical items on-demand, was shown. A brief report was given for the performance tests of the HTTP, RTSP, and the PNA protocols, as used in the audio communication component within different network environments. As anticipated, the results showed that the real-time transmission of compressed audio has no substantial delivery delay within both the LAN and WAN/dedicated networking environments. However, on the Internet, established protocols such as RTSP are needed.

Chapter 5

Internet Credit Card Transactions

The emergence of Internet commerce has presented an unprecedented opportunity for new ways of conducting business. This has led to the development of systems that facilitate online credit card payment and processing. The vast growth potential for Internet commerce is however tempered by legitimate concerns over the security of such systems. Securing electronic commerce must occur on all fronts, including data confidentiality, authentication of the participating parties, data integrity, and non-repudiation [31]. This chapter discusses the security of payment on the Internet. It first introduces an overview of the credit card payment process in the real world and on the Internet. It then shows how the credit card payment process was implemented in WAPS.

5.1 Credit Card Payment Process Overview

Many people use credit cards frequently to purchase groceries, put fuel on their cars, charge meals at restaurants, buy books at *Amazon.com*, etc. In most cases these people never really know what is involved when making a credit card purchase nor do they even think about the process running in the background that lets them make these purchases. For example, what actually happens when a merchant swipes a credit card through a Point of Sale device? Who authorizes the charge payment? What did the merchant have to do to be able to handle credit card transactions? To have a complete picture of how credit cards work both in the real world and on the Internet, one needs to know the following:

- The parties involved in credit card transactions,
- How the credit card information travels among the involved parties, and
- The security measures applied to protect the payment details

Making purchases using a credit card both in the real world and over the Internet involves four parties:

1) The credit card holder or the consumer,
2) The merchant who is offering products or services for sale,
3) The merchant's bank (also known as the acquiring bank or the acquirer) that has contracted with the merchant to enable the merchant to accept credit cards.
4) The issuing bank that has issued the consumer's bank credit card (e.g. MasterCard or Visa)

The acquiring bank processes merchant's credit card transactions with the card-issuing bank through a private financial network. This network is usually the Internet i.e. dial-up standard or ISDN modems, or sometimes leased lines. For a better understanding of a credit card transaction, the following is a step-by-step breakdown from the customer making a purchase to the merchant making an authorization request to the bank. This description covers the starting point, when the merchant swipes the consumer's credit card on a point of sale device, through to the final point when the acquiring bank sends an approval of purchase to the merchant. However, the interaction between the acquiring bank and the card-issuing bank is treated as a black box, as there is no public information available about it. No one source could readily be found to present the whole picture of how the acquiring bank and the issuing bank interact. The steps involved in a normal credit card transaction are as follows [32]:

a) Merchant calculates the amount of purchase and asks buyer for payment.

b) Buyer presents merchant with a credit card.

c) Merchant swipes the credit card through a point of sale device (Speed point terminal). During this process, either the sales amount is hand-entered on the point of sale device or is automatically transmitted by the cash register.

d) Merchant transmits the credit card data and sales amount with a request for authorization of the sale to their acquiring bank. RFC1898 (CyberCash Credit Card Protocol Version 0.8) points out that point of sale devices are set to request authorization at the time of sale, but actually capture the sales draft at a later time.

e) The acquiring bank routes the authorization request to the card-issuing bank. The credit card number identifies type of card, issuing bank, and the cardholder's account.

f) If the cardholder has enough credit on his account to cover the sale, the issuing bank authorizes the transaction and generates an authorization code. The issuing bank then puts a hold on the cardholder's account for the amount of the sale. Note that the cardholder's account has not been charged yet. This code is then sent back to the acquiring bank.

g) The acquiring bank receives the approval or denial code and sends it to the merchant's point of sale device.

h) The point of sale device prints out a sale draft, or slip. The merchant asks the buyer to sign the sales slip, which obligates them to reimburse the card-issuing bank for the amount of sale.

i) At a later time, probably that night when the store closes up, the merchant reviews all the authorizations stored on the point of sale device against the signed sales slips. When all the credit card authorizations have been verified to match the actual sales drafts, the merchant will then capture or transmit the data on each authorized credit card transaction to the acquiring bank for deposit.

j) The acquiring bank performs what is called an interchange for each sales draft with the appropriate card-issuing bank. The card-issuing bank transfers the amount of the sales draft, minus any interchange fees, to the acquiring bank.

k) The acquiring bank then deposits the amount of all the sales drafts submitted by the merchant, less a service fee, into the merchant's bank account.

Figure 1 illustrates the above process.

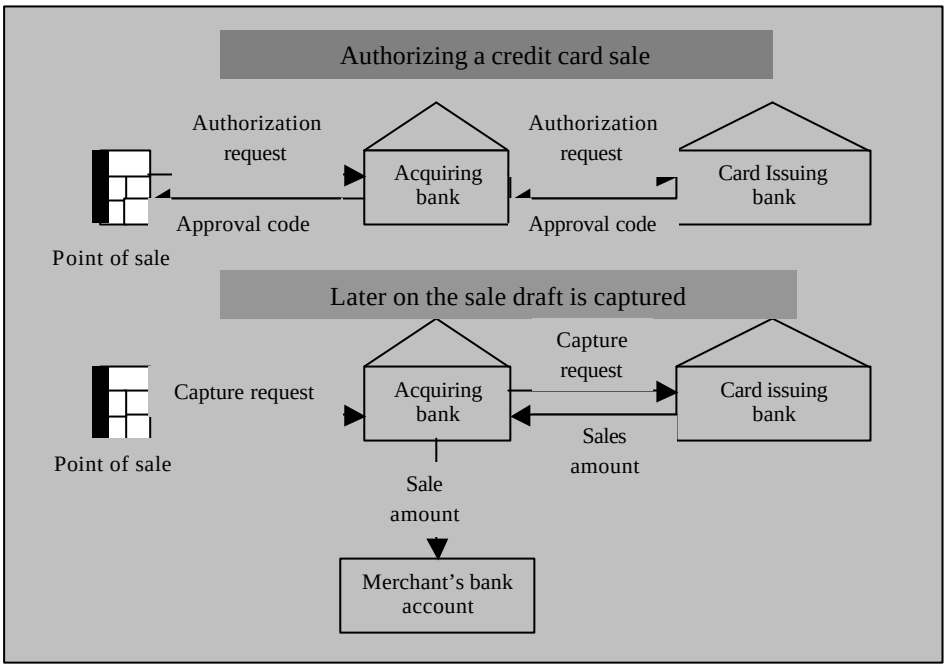

Figure 1: Real-world credit card transaction process

As mentioned earlier, the overview presented above is not complete. It does not cover the interaction between the acquiring bank and the card-issuing bank and it is also geared towards MasterCard and Visa transactions. For example, there is no card-issuing bank associated with American Express or Discover cards.

5.2 Online Credit Card Transactions

It seems natural that online commerce will be done with credit cards, since no physical paper needs to be passed (such as cash and cheques). People simply type their credit card numbers into the merchant's web page payment form and wait for the purchased goods to be delivered to them. This process might seem very simple, but the problem is that people still have legitimate fears about giving their credit card numbers over the Internet. The reason is that the Internet is an open network without any basic security provisions built-in, unless secure servers are involved for transporting data. Due to these fears, several methods have been and are still being developed to make purchasing products online more secure. There are two main methods for securing online credit card transactions:

- Taking the transaction off-line.

This was more or less the first attempt at making online credit card transactions secure. With this approach a commerce site would allow one to call-in the credit card number to a customer support person using normal telephone lines. This solved the problem of passing credit card numbers over the Internet, but eliminated the merchant's ability to automate the purchasing process. Also the merchant has to have employees available 24 hours a day to take phone calls from buyers. This method raises further problems when a buyer has one phone line, meaning that he has to log-off the Internet in order to actually make a purchase.

- Using payment security protocols.

This method is currently being used by many web sites. In this approach secure servers that use Internet security protocols are used by merchants when transmitting payment data between a customer and a merchant. With these protocols, the data being transmitted is encrypted such that the data travels securely when one submits a credit card number through a web storefront.

5.3 Current Internet Security Approaches

It was apparent that for online commerce to flourish, advanced and truly secure means of making payment needed to be developed. The typical current security approaches revolve around server-based systems. These systems are usually based on a "Web of trust" around the network. This trust basically consists of multiple layers of firewalls and centralized servers connected via secured network tunnels to individual client PCs. Secure transactions, therefore, require a series of complex steps including the utilization of technologies such as:

- Authentication,
- Digital Certificates,
- Checking the validity of credit card number and the available balance, and
- Encoding/decoding the payment data at each end of the network.

For e-commerce to grow, consumers and businesses need to be able to establish a "trust" that their transactions are secure and private. At present there are many software-based initiatives to establish this trust. These include the use of the Secure Socket Layer (SSL) and the Secure Electronic Transaction (SET) protocols. There are also numerous proprietary credit card payment systems that are based on the above two security protocols. In this section we shall discuss the two standard protocols (i.e. SSL and SET) and two of proprietary credit card payment systems, Virtual Vendor [35] and the BankGate systems [38].

5.3.1 SSL

At a minimum, an Internet payment system should provide the SSL protocol security for encrypting the customer's payment information. SSL was developed by Netscape communications [37]. It is a protocol that allows the authentication and encryption of data between client and server applications. SSL has two major aims [33]:

a) To authenticate the server and (optionally) the client using public key signatures.

b) To provide an encrypted connection for the client and server to exchange messages.

SSL is generally comprised of two protocols: the record protocol and the handshake protocol. The record protocol defines a way in which messages passed between the client and server are encapsulated. It defines a set of parameters, known as cipher suites that define the cryptographic methods being used at any point in time. There are a number of cipher suites defined by the SSL standard with names that describe their content. For example, the cipher suite named: SSL_RSA_EXPORT_WITH_ RC4_MD5 can be interpreted as using:

- RSA public key encryption for key exchange using an export strength modulus,
- RC4 cipher for bulk data encryption using a 40-bit (export strength) key, and
- MD5 hashing to ensure data integrity.

When a SSL record protocol session is first established, it has a default cipher suite of SSL_NULL_WITH_NULL_NULL (i.e. no encryption at all). This is where the SSL handshake protocol comes in. The handshake protocol defines a series of messages in which client and server negotiate the type of connection that they can support, use to perform authentication, and generate an encryption key. Once the handshake is over, the client and server exchange ChangeCipherSpec messages, which switches the default cipher suite of the record protocol to the one that they negotiated during handshake. Figure 2 illustrates how the SSL handshake occurs.

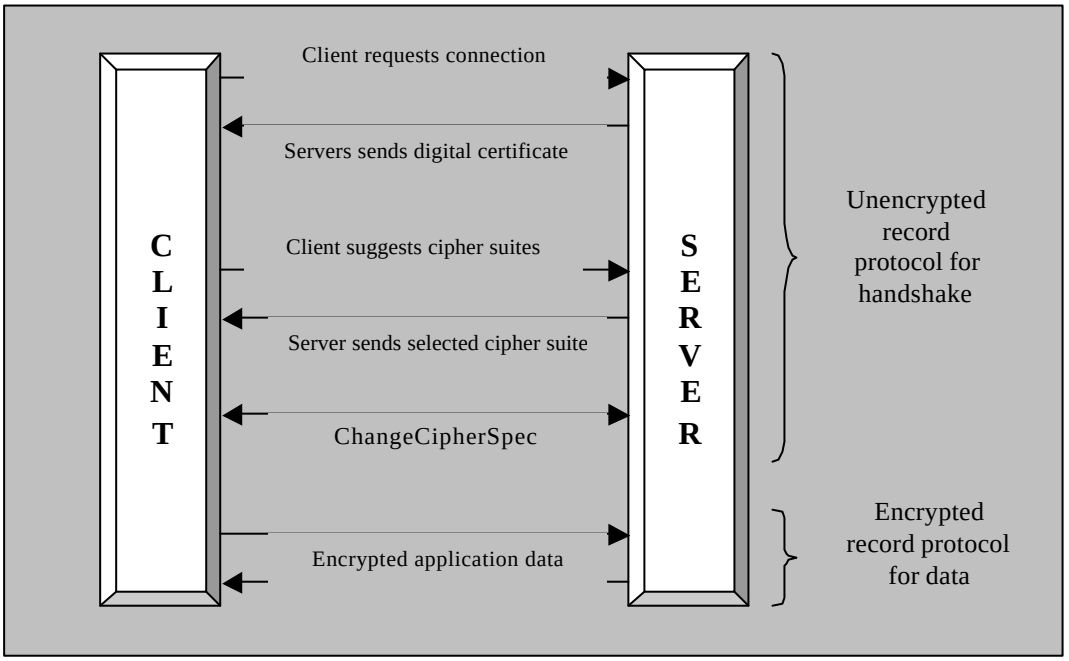

Figure 2: SSL handshake

From the case shown in the diagram, only the server is authenticated, hence the client does not need to provide a Digital Certificate (or Digital ID). But, the server has to have a valid Digital Certificate issued by a Certification Authority (CA). This certificate is then used by the client to prove the server's identity during handshake. A Digital Certificate or Digital ID is a means of verifying one's identity on the Internet. Digital Certificates are issued by a third party company known as a Certification Authority (CA) for the purpose of proving ones' identity. They are also used as way of creating trust between groups conducting transactions that may involve personal information or monetary exchange.

5.3.2 SET

The SET (Secure Electronic Transaction) specification is a technical standard for making credit card purchases over open networks. SET was initiated on February 1, 1996, and its development was driven by two of the world's largest credit card organizations, Visa International and MasterCard, with participation from leading technology companies including IBM, Microsoft, Netscape, RSA Data Security, Terissa Systems, Verisign, and others. Its significance over existing security protocols is found in the use of Digital Certificates and public key cryptography to protect the privacy and integrity of payment information sent over the Internet. SET incorporates public key cryptography from RSA Data Security, which includes 56-bit DES (Data Encryption Standard), and RSA public/private key pairs using a 1024-bit modulus. To utilize SET, merchants, shoppers, and acquiring banks need software that follows the SET protocol. SET also requires that the merchant, shopper, and the acquiring bank have Digital Certificates from a Certification Authority. These certificates are then used by the protocol to authenticate all the parties involved in a SET transaction. The SET protocol addresses seven major business requirements [34]:

- SET provides "confidentiality" of payment, and enables confidentiality of order information.
- It ensures integrity for transmitted data.
- It authenticates the cardholder as a legitimate card account user.
- It authenticates the merchant as authorized to accept bankcard payments.
- It ensures the use of the best security practices and system design techniques to protect all legitimate parties involved in a transaction.
- It provides a protocol that neither depends on transport nor security mechanisms.
- SET facilitates interoperability across network and software providers.

The confidentiality of transmitted information in SET is ensured by message encryption and digital signatures for ensuring the integrity of payment information. SET uses DES symmetric key encryption along with RSA digital signatures for authentication and key transfer. During the process of authentication, a cardholder (customer) is authenticated through a digital signature along with a cardholder

certificate issued by the cardholder's bank or an approved Certification Authority. A merchant is authenticated through a digital signature along with a merchant certificate issued by the acquiring bank. The interoperability is enabled through the use of specific protocols and message formats. SET messages are transferred as MIME messages and their cryptographic encapsulation follows PKCS-7 (Public Key Cryptography Standard, Number 7) [34].

SET uses 160-bit message digests (i.e. algorithms that generate a "hashing" of a piece of text or binary data) produced by the SHA hash algorithm, and two public/private keys pairs. One pair is used in private key exchange. During encryption, a message is encrypted using a randomly generated DES private session key, where the first public key pair is used to encrypt and decrypt the DES session key. The second key pair is then used to create and verify digital signatures. A message is hashed by the SHA-1 hashing algorithm and then signed with the private key of the second key pair, and afterward verified with the corresponding public key. The following is a simple summary of the steps that occur in a SET commercial transaction:

1) Cardholder shops.
2) Cardholder's software sends an initiate request to merchant asking for public keys of both the payment gateway (acquiring bank's software for processing credit card authorization and settlement services) and that of the merchant. This request indicates the type of credit card the cardholder will use. The cardholder's software needs both the merchant's and the payment gateway's public key before it can send transaction details to the merchant.
3) Merchant software generates a response to the request and replies to the cardholder software. This report includes a unique transaction identifier generated by the merchant, and the requested certificates.
4) Cardholder software verifies both the merchant and the payment gateway.
5) Cardholder software generates two packets of information to send back to the merchant and these are:
    a) An order information (OI) packet

       This is created using the information from the shopping phase. It is the data meant for the merchant to see containing the transaction identifier, brand of credit card being used, and the transaction date. The OI is

encrypted using the merchant's public key. The merchant does not get to see the cardholder's credit card number.

b) The purchase instruction (PI) packet

The PI is the data meant for the acquirer to see. It is tunneled through the merchant to the payment gateway. The PI is encrypted with the payment gateway's public key. It includes a credit card number and expiry date, purchase amounts agreed to by the buyer, and a description of the order.

6) Cardholder software transmits the OI and the PI to the merchant.
7) Merchant software verifies the cardholder certificate for tampering. If no tampering is found, it then starts the process of requesting authorization from the acquirer.
8) Merchant software creates an authorization request for payment.

Included in this request is the transaction identifier that the merchant generated at the beginning of the payment process. It then encrypts the authorization request using the payment gateway's public key.

9) Merchant software transmits the encrypted authorization request and the encrypted PI from the cardholder purchase request to the payment gateway.
10) Payment gateway verifies the merchant and decrypts the message and its various components such as the PI from the cardholder. It checks the various parts of the message for tampering. These include:
    - Making sure the transaction identifier in the authorization matches the one in the cardholder's PI packet.
    - Making sure the merchant has not tried to temper with the data in the cardholder's PI packet.
11) The gateway sends an authorization request through an existing financial network to the issuing bank.
12) The issuing bank sends back an approval or denial response to the payment gateway in response to the authorization request.
13) Payment gateway creates an authorization response message to be sent back to the merchant. This message includes:
    - The issuer's response, and
    - A capture token to be used by the merchant when requesting capture of the sale later on.

14) Payment gateway encrypts and sends the authorization response back to the merchant software.
15) Merchant software verifies the payment gateway and decrypts the authorization response from the payment gateway. It then stores the authorization response message and the capture token sent by the payment gateway for later use when capturing sales.
16) Merchant software creates a purchase response message to the cardholder's software. This message informs the cardholder that the payment was accepted or not accepted and that the goods will or will not be delivered.
17) The cardholder's software processes the purchase response message and informs the cardholder that payment was accepted or not accepted.
18) At a later time, the merchant software creates a capture (settlement) request message to send to the payment gateway. This request includes the capture token, transaction ID, and the authorization information. The sequence of events surrounding the capture request is very similar to steps 11 – 17 of the authorization process.
19) Merchant software creates a capture request and encrypts it using the payment gateway's public key.
20) Merchant transmits the encrypted request to the payment gateway.
21) Payment gateway verifies the merchant and then decrypts the capture request and ensures consistency between the merchant's capture request and the capture token.
22) The gateway sends the capture request through a financial network to the issuing bank.
23) The issuer sends back an approval or denial response to the payment gateway in response to the capture request.
24) The gateway creates a capture response, encrypts it and transmits it to the merchant.
25) Merchant verifies the gateway and stores the capture response to be used for verification with payment received from the acquirer.

From the above SET transaction flow, one must notice that "confidentiality" means that the messages cannot be read in transit, such as at Internet gateway routers or local access providers. The customer's credit card number is likewise concealed from

merchants. However, the SET acquiring bank software is privy to all transaction information. It can both link together transactions made by a customer, and it can link purchases to credit withdrawals on the credit card.

5.3.3 Virtual Vendor

Developed by Virtual Vendor CC [35], virtual vendor is an e-mail message-based credit card payment model that was initially developed for mail order and subscription related transactions. Unlike SET, virtual vendor only protects the payment data between the merchant and the acquiring bank. This means that a way has to be provided to transfer payment data from the client (storefront) to the merchant securely across the Internet. Virtual vendor relies on e-mail messages being exchanged between the merchant and the acquiring bank. Since virtual vendor uses SMTP (Simple Mail Transfer Protocol) for the transmission of payment data, the sales settlements do not get effected immediately but do take place before the bank's daily cut-off times. Virtual vendor uses the PGP (Pretty Good Privacy) [50] encryption and authentication algorithms for the message-based credit card authorization and settlement services.

The virtual vendor method offers four different services: Single Authorization Service, Multiple Authorization Service, Transaction Management Service, and a Settlement Service. In each case, once the payment data has been received from a customer via a secure channel, the merchant creates a message containing an authorization request, which is then encrypted using PGP and transmitted to the acquiring bank through SMTP. In the Single Authorization Service, only one authorization is handled at a time and, once the issuing bank has approved the transaction and a response has been received by the acquiring bank, it is recorded on a database for future references, encrypted, and returned to the merchant by e-mail. The Multiple Authorization Service allows for at least 10 authorization requests to be handled at any one time. The Transaction Management and Settlement services allow the merchant to send requests for settling transaction such as sales and refunds.

With virtual vendor, once the issuing bank has approved a transaction, the acquiring bank automatically settles it without the merchant having to send a sale

capture request. All approved transactions are automatically settled in time to meet the bank's cut-off times and a statement is e-mailed to the merchant. The following is a summary of the steps that take place during a virtual vendor authorization request:

1) Customer enters payment details in a web form on the merchant's storefront.
2) Payment data is transferred to the merchant through a secure channel.
3) Merchant receives payment data, creates a file of transactions, encrypts the file using PGP, includes the encrypted file in the body of an e-mail message addressed to the acquiring bank, and sends it. The transaction file contains information such as customer's credit card number, expiry date, sales amount, merchant number, and a unique transaction number. This process is done manually by the merchant using a software module called Virtual Client, which allows for entering the customer's credit card details and sending them to the acquiring bank via encrypted mail.
4) The acquiring bank decrypts the message, requests authorization from the bank that issued the consumer's credit card, and if it is approved will instruct the bank to transfer the funds to the merchant's account.
5) The acquiring bank creates an e-mail message, encrypts it and sends it to the merchant. This contains:
    a. Any errors there may be in the authorization request from the merchant,
    b. The result of the authorization request,
    c. The amount to be transferred to the merchant's account,
    d. The transactions held over to the next day for re-trying, and
    e. The transactions that have failed and require manual intervention,
6) Merchant receives the encrypted mail, decrypts it using PGP and may inform the customer accordingly via e-mail.

Just like SET, both the virtual vendor and the BankGate systems require that the merchant and the acquiring bank exchange their public cryptographic keys for encryption purposes, and that they both obtain Digital Certificates through a Certification Authority for authentication purposes.

5.3.4 BankGate™

The BankGate method uses models that allow for immediate authorization and settlement of transactions by banks. For protecting payment data among the parties participating in a BankGate transaction, BankGate provides support for the SSL-3, SET, and its own proprietary security protocols. BankGate has several services available for merchants, shoppers, issuing and acquiring banks. It is mainly composed of three primary software components. For merchants, WebPOS (web point of sale) is a program that emulates the traditional point of sale device across the Internet, performing merchant and bank authentication using Digital Certificates. Data is protected whilst being transmitted from the merchant to the acquiring bank using up to 128-bit encryption. For shoppers, there is WebWallet, which is a program that provides a SET wallet. There is also WebGATE software for acquiring banks, which offers BankGate merchant services, such as authorization and settlement of transaction with the issuing bank. WebGate can be referred to as a payment gateway. Figure 3 shows the program components that form the BankGate system and how they relate to each other.

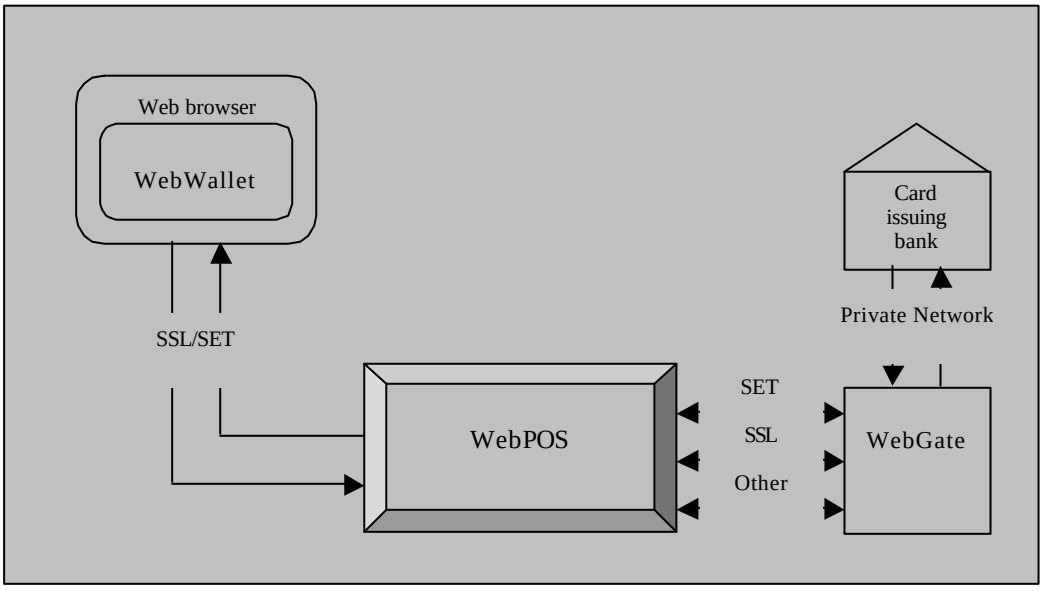

Figure 3: BankGate's component programs

A typical BankGate transaction follows exactly the same steps described in the SET transaction flow. A customer enters his payment details on the WebWallet (cardholder software), which then submits it using SSL, SET or the BankGate proprietary protocols to the merchant software (WebPOS). The merchant forwards the payment

data to the payment gateway (WebGATE) of the acquiring bank, which in turn does the authorization and settlement with the issuing bank, and sends a response to the merchant. The only drawback with the SET (or BankGate) approach is the fact that cardholders need to have Digital Certificates issued to them by the merchant or a Certification Authority. This implies that if a million users use a web store, then a million Digital Certificates have to be issued, which is a practical impossibility. This is one of the major factors that will delay the acceptance of the credit card payment methods based on the SET protocol unless some methods are developed to work around it until such time as Digital Certificates become widely available to consumers.

5.3.5 Alternative Methods for Online Credit Card Payments

As an alternative to credit card payment over the Internet, there are new payment methods being developed, such as electronic cash (E-cash). E-cash is digital money that one can use to make online purchases. Consumers interested in shopping with E-cash need to have special software on their computers that allows them to download money from their bank accounts into their E-cash wallets on their computers. When making a purchase, they exchange this downloaded money with the merchant for the product they want to buy. The merchant then redeems the money at a bank that accepts E-cash deposits. DigiCash [36] is one example of a company, which has developed an E-cash system.

5.4 The WAPS Implementation of the Credit Card Payment Process

The process of commerce-enabling a site can be time consuming and hard to figure out. Three steps were followed when developing the purchasing phase of the Web-based Audio Purchasing system. These were:

- Getting a merchant account with a bank,
- Designing a shopping cart that customers can use to place orders, and
- Setting up payment processing software for securing the payment data

5.4.1 Getting a merchant account

This is a first step towards commerce enabling a Web site. It involves obtaining a virtual merchant number from a bank. At present it is still hard for merchants to get the virtual merchant status, especially for small businesses. The reason for this is that not all banks are ready to provide this service and those that do are bigger banks that are afraid of extending merchant status to small businesses as they consider them to present too much of a risk. These banks are afraid that a at-risk business may not be able to handle any charge backs (credit reversal) that hit its accounts. Thus, if a merchant cannot handle a charge back, the merchant's bank will have to absorb the loss. With online shopping, it is almost guaranteed that charge backs will occur at some point due to a customer claiming the ordered goods never arrived, or were not what the customer ordered. The card issuing banks penalize the acquiring banks that have more than a certain percent charge back of their sales. This percentage varies from banks to banks, but normally it is around 1 percent [39]. This then results in that particular merchant losing its merchant status with the bank and this also makes acquiring banks more reluctant to give out virtual merchant accounts.

The average approval period for a virtual merchant account can take up to six weeks. This is because a thorough investigation into the credit worthiness of the merchant applying for virtual merchant status has to be carried out by the bank. Unlike the normal merchants that are supplied with a credit card terminal by the bank and can thus provide a customer with a receipt to sign on site, virtual merchants cannot provide the buyer with an ink-on-paper proof of purchase. For this reason the banks examine extreme caution in whom they allow to become a virtual merchant. The following are some of the factors that banks consider when processing merchant status approvals:

- The authenticity of the products on sale,
- The average amount per transaction,
- Merchant's projected monthly sales volume,
- How long the merchant has been in business,
- What kind of credit rating the merchant has, and

- The kinds of cards that the merchant wants to accept, e.g. Visa, MasterCard, etc.

Though bigger banks offer credit card merchant accounts, there are still very few that offer virtual merchant accounts at present. One of the reasons for this is the fear of credit card fraud on the Internet, while others are just not sufficiently knowledgeable. Since WAPS is to be used by the International Library of African Music, an Institute of Rhodes University, the University's merchant account number was used when commerce-enabling WAPS. Otherwise ILAM would have had to go through the process outlined above to obtain a merchant account.

5.4.2 Designing a Shopping Cart

When people are shopping over the Web, they typically select a number of items that are then kept in a virtual shopping cart. The shopping cart keeps a continuous list of what the customer wants to buy and how much it costs. Once the customer has finished selecting the items, he simply pushes a button on the shopping cart's interface to place an order. The most popular method for creating a shopping cart is to use the Common Gateway Interface (CGI). With CGI shopping carts, all the shopping information resides on the Web server.

This implies that every time a customer adds or removes an item in the cart, a communication will have to be established with the Web server. This becomes a problem when a customer has a slow connection, as it may take a while to select the items. For this reason the WAPS shopping cart was developed using Java. This enabled the management of selected items on the customer's computer rather than saving them on the Web server, and when the customer decides to place the order, the Java shopping cart sends the order to the merchant only once. A simple framework for the WAPS shopping cart was created and a user interface was attached to it.

5.4.2.1 The WAPS Shopping Cart's Framework

Firstly the attributes of the items that had to go in the cart were selected. These included the title of the song, price, and the quantity. The quantity value was included to avoid multiple instances of the same item in the cart. The following code fragment

shows a simple implementation of the shopping cart item i.e. an object that defines the attributes of the items that are to go to cart. These include a song title, price, and a quantity.

```
class Items {
public String title;
public double price;
public int quantity;

Items (String t, double p, int q) {
 this.price = p;
 this.quantity = q;
}
 //The update method is used when adding items to a cart, It does not perform any checks to
 //ensure that they are similar
public void update (double p) {
  price = price+p;
  quantity++;
 }
  //The delete method is similar to the add method but it removes a certain quantity of items
 public void delete (double p, int q){
  price = (price-p)/q;
  quantity--; } }
```

The next step was to create the actual shopping cart itself. A hash table object for storing the cart items was created. When storing data in a hash table object, a key is associated with each data entry (see Figures 4 and 5).

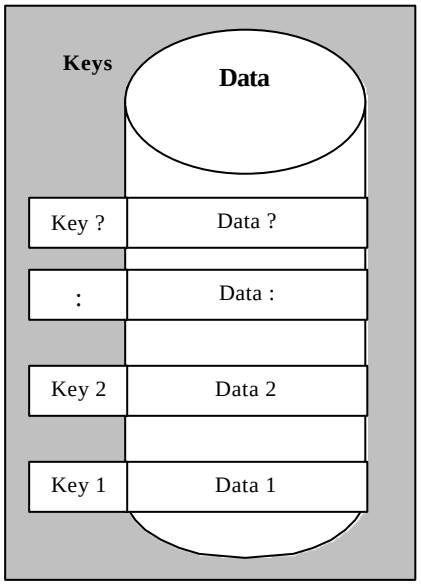

Figure 4: A hash table

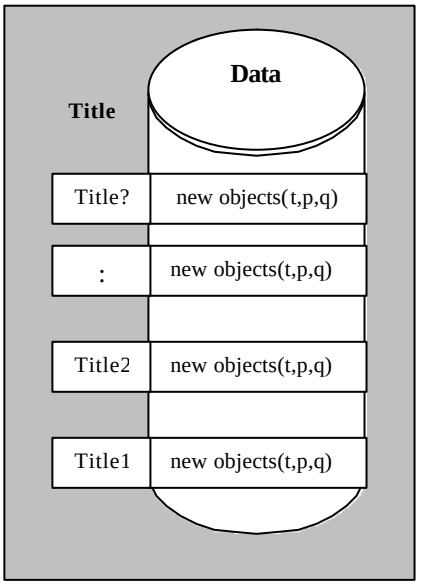

Figure 5: Actual shopping cart

The keys are then used to retrieve the data stored on the hash table. For the WAPS shopping cart, the song titles were used as keys to the data, which was stored as an

object (class Items defined by the code snippet above) with attributes price, quantity, and title. Figure 5 shows the structure of the actual shopping cart. The following code fragment shows the actual shopping cart object implementation:

```
import java.util.*;
class wapsCart{
Objects obn = null;
Hashtable cart;

wapsCart
{
 cart = new Hashtable();
 }
 //This method is for adding items to the cart, It performs checks to ensure that there is no
 //similar items. If the item already exists, it uses the update method of the cart
 //implementation (class Objects) to increment the price and the quantity value of that
 //particular item.
 public void addItem (String t,double p, int q)
 {
  if (cart.containsKey(t)!=true){
  // item does not exist, store it on the hash table
  cart.put(t, new obn(t,p,q));
  }
 // item already exists, update the price and the quantity value
 else
  {
   // retrieve the record
   obn = (Objects)(cart.get(t));
   // update the price an the quantity values
   obn.update (p,q);
  // save the record back
  cart.put (t,obn);
 }
 ……
}
```

Other major methods implemented by the shopping cart object include getTotalCost () for calculating the total amount of the selected items and delete () for deleting an item on the shopping cart.

5.4.2.2 The Shopping Cart's User Interface

Generally, the structure of a Web storefront has two parts: the catalogue (the Information Retrieval Engine in WAPS) and the shopping cart (WAPS shopping cart). When a user is buying musical items, he searches for the available music of interest through the catalog and selects ones he wants. The catalog then sends these items to the shopping cart. All this has to be done through a suitable and easy to use interface. The WAPS shopping cart is composed of two canvases: the main canvas for reviewing the selected items, and the order canvas for placing an order of the selected

items and entering the customer's payment details, such as address, credit card number, e-mail, etc. Figures 6 and 7 show snapshots of the two canvases.

Figure 6: The main shopping cart canvas

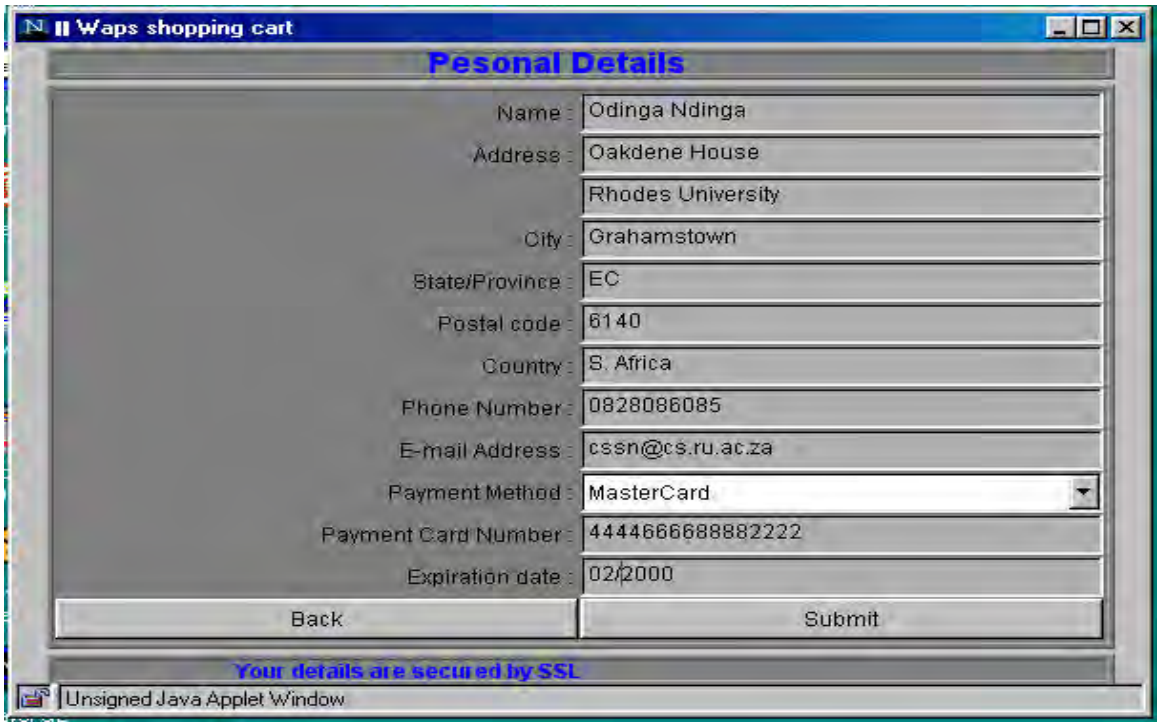

Figure 7: The order canvas

The shopping cart's interface was developed using the standard Java GUI components and layout managers.

5.4.3 Securing Payment Data

A typical WAPS credit card transaction starts with the customer selecting and ordering a musical item from the WAPS strore front. The payment information is sent securely to the merchant and then to the acquiring bank to get authorization for the purchase. The approval or denial is returned to the merchant, which generates the consumer's digital receipt and then the customer is forwarded a Web page with links to the purchased items. The approach that was used when securing the customer's credit card information in WAPS was implemented in two phases. Firstly, the SSL protocol was used when transferring the payment data from the storefront to the merchant. Secondly, both the Virtual Vendor approach and the WebPOS software (BankGate's software program for merchants) described in sections 5.3.3 and 5.3.4 were used for the actual authorization request between the merchant and the acquiring bank. Figure 8 shows the implemented architecure, which forms the WAPS logical payment component. Figure 8 also shows the two phases of implementation.

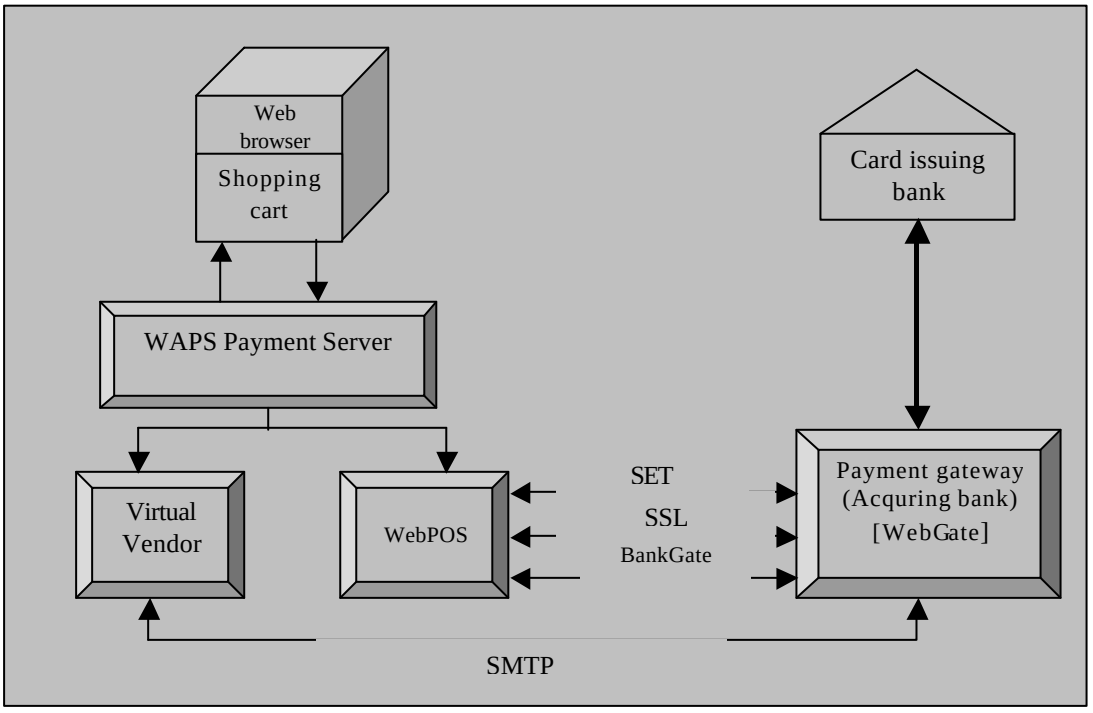

Figure 8: The implemented architecture

As shown on the figure above, the implemented architecture for the credit card payment process integrates two of the WAPS physical components, which are the shopping cart and the payment server, with two external components, which are the payment gateway and the issuing bank. Apart from storing the selected musical items, the shopping cart is also responsible for the transportation of the customer's credit card details to the payment server. The payment server is a multi-threaded stand-alone Java application that executes on the merchant's computer waiting for payment data from the shopping cart over a secure channel of communication. The payment server has been interfaced with the WebPOS software (BankGate's program for merchants) in order to send real-time authorization and settlement requests to the acquiring bank. The following two sections describe the two implementation phases that form the logical payment communication component mentioned in section 1.3.3 of Chapter 1.

5.4.3.1 Phase 1: Consumer to Merchant data transfer

For transferring the customer's credit card details from the storefront to the merchant, a client/server architecture was defined and developed using Baltimore Technology's J/SSL toolkit [40], a Java SSL version 3 class package. This package provides a set of Java classes that mirror the *java.net* socket classes and includes two other socket classes named *SSLSocket* and *SSLServerSocket*. These additional socket classes provide a transparent implementation of the SSL version 3 protocol. These classes behave exactly like their *java.net* equivalents, except that their constructor methods also require a JcipherSuite object among their arguments for providing cipher suite details.

Using this J/SSL toolkit, two Java threads were defined. These were a SSLClient thread named "client" and a SSLServer thread named "server". The client thread was implemented as an inner class of the shopping cart, while the server thread was implemented as the main class of the payment server. Basically, the client thread implements the client side of the SSL protocol, while the server thread implements the server side of the SSL protocol. The server thread is started automatically when starting the payment server and it executes as a daemon thread waiting for SSL connections from clients. When a customer presses the submit button on the order canvas of the shopping cart's interface, the client thread is instantiated. It then sends a data packet requesting an SSL session to the server thread. The server responds by

sending the merchant's Digital Certificate, which it encrypts with its private key. This server's Digital Certificate is then used by the client to prove the server's identity.

This certificate includes a chain of certificates up to the Certification Authority (CA) that the client trusts. Once the client receives the server's Digital Certificate, it decrypts it using the server's public key. The tricky part when setting up an SSL connection is that the client has to obtain a key ring, which is a conventional file containing a database of keys (including the server's public key needed to decrypt the server's Digital Certificate) and the CA's certificate (that issued the merchant's certificate), before an SSL session is created. Since the key ring contains some confidential information, it has to be transferred securely from the merchant to the client thread. In this project, the key ring was encrypted using the cryptographic libraries that are part of the J/SSL toolkit and it was placed on the merchant's Web server. The client thread loads the key ring from the Web server using the URL class provided by the Java class library. The key ring is then decrypted by the client thread. The following code fragment shows how a client thread loads the key ring from the Web server using the URL class.

```
……..
public void loadKeyRing(){
 try{
  byte keyring[] = new byte [ring.length]
  for (int i=0; i<ring.length; i++)
  {
   ByteArrayOutputStream out = new ByteArrayOutputStream ();
   URL url = new URL (getURL ()+keyring[I]);
   InputStream fin = url.openStream();
   int n;
   while ((n=fin.read(keyring))>=0)
   out.write(keyring,0,n);
   fin.close;
   byte[] content = out.toByteArray();
  supportFactory.addCACertificate(content);
 }}}
```

When the client has finished authenticating the server, an SSL handshake begins. During handshake, the client suggests to the server a list of SSL cryptographic algorithms that it is able to support. The server goes through the list, picks up the strongest algorithm that is common to both, and an SSL session is established. The client retrieves all items that are on the shopping cart and the payment details entered on the order canvas, and sends them through the established secure connection to the server. The server gathers the ordered items with the credit card details and gets ready

to create an authorization request with the acquiring bank and to deliver the ordered items.

5.4.3.2 Phase 2: Merchant to Bank data transfer

This covers the transfer of the payment data from the payment server to the payment gateway of the acquiring bank. Both Virtual Vendor and the BankGate credit card payment systems were used for this. As to which payment approach to use at a particular point in time, the merchant has to specify this on a configuration file that is read by the payment server during start-up.

- **Using the Virtual Vendor method**

When this payment approach is being used, the payment server performs two independent operations: Firstly, once the payment server has received the customer's credit card details and the list of the ordered items, it generates an HTML page containing links that point to where the customer can download the ordered musical tracks. This HTML page is then sent to the shopping cart through the SSL connection. The client then uses the *showDocument* method of the URL class package, to display the HTML page on a separate browser window, thereby allowing the customer to start downloading the purchased items. Secondly the payment server creates a transaction file. This file contains a merchant number, merchant ID, customer's credit card number and expiry data, and the total sales amount. The payment server executes PGP from the command prompt to encrypt the transaction file. The following code fragment shows how PGP is executed within the payment server:

```
……
public void encrypt (String filename)
{
 try{
  Runtime rt = Runtime.getRuntime();
  // run pgp from command prompt
  Process pro = rt.exec("pgp –eatw "+filename+" " +publicKey);
 …..
 }
 catch(IOException e)
 {}}
```

Having encrypted the transaction file, the payment server constructs an e-mail addressed to the acquiring bank, attaches the encrypted transaction file, and sends it to the acquiring bank. The acquiring bank decrypts the transaction file using its private

key pair, and forwards the authorization request to the issuing bank for credit authorization and settlement. The issuing bank sends the response to the acquiring bank, which then sends it to the merchant by PGP encrypted email. If the authorization request was approved, the acquiring bank automatically sends a request to the issuing bank for the transfer of funds from the customer's account to the merchant's account.

As can be seen from the above scenario, the merchant does not wait for the bank's response before allowing customers access to the purchased items. This is due to the fact that the implemented approach uses the normal email messages for authorizations and it can take a very long time before a response can be received from the bank. Thus, the Virtual Vendor's message-based payment technique opens opportunities for fraud, since users of the online store can gain access to the musical files while having insufficient funds on their credit card accounts. The following method of payment that was implemented utilizes the BankGate payment approach and it offers an immediate authorization and settlement of requests.

- **Using the Bankgate System**

Also in this approach the payment server performs two operations: First, the HTML page that contains the links to where the ordered items can be downloaded is generated by the payment server. Secondly, the payment server is interfaced with the WebPOS software mentioned in section 5.3.4 for authorization purposes. Unlike the previous message-based payment process, the generated HTML page cannot be sent to the shopping cart until a response is received from the acquiring bank indicating that the authorization request was approved. The WebPOS software used is freely available from http://www.bankgate.com. It is compatible with the SET version 1.0, SSL version 3, and other proprietary BankGate protocols, allowing merchants to choose the type of protocol to use when tranferring data from the merchant to the acquring bank. WebPOS can be interfaced with backend systems that support CGI calls or calls to command line programs.

In our case the payment server was interfaced with WebPOS by using the command line method. The following code fragment shows a typical authorization

request issued by the payment server and it shows how the payment server and WebPOS were interfaced.

```
…..
public void authorize()
{
 // create the authorization request
 String   request="Function=Authorise&Protocol="SET"&BankGateID=bankgateID&
                  CardNumber=ccnumber&CardExpiry=exp_date&Amount=sales&
                  Currency= rands&TransactionID=refNo";
try{
  Runtime rt = Runtime.getRuntime();
  // Execute WebPOS and forward it the authorization request
  Process pro = rt.exec("webPOS  "+request);
  //read the response
  InputStreamReader isr = new InputStreamReader(pro.getInputStream());
  BufferedReader br = new BufferedReader (isr);
  String response = br.readLine();
 }
catch(IOException e){}
```

The bold words are the BankGate argument fields whose values have to be passed to the WebPOS program. For example,

- **Function:** identifies the type of transaction to be sent to the payment gateway, i.e. whether it will be an authorization request or a capture request
- **Protocol:** an argument used for selecting which security protocol is to be used when sending the authorization request to the WebGate (payment gateway) program of the acquiring bank.
- **BankGateID:** is a unique ID number issued by the acquiring bank to the merchant. It is a terminal identification code
- **CardNumber:** customer's credit card number
- **CardExpiry:** credit card's expiry date
- **Amount:** sales amount
- **Currency:** local currency
- **TransactionID:** a randomly generated number that uniquely identifies a particular transaction

When the payment server executes the above sample code, the authorization request is passed to WebPOS, which then uses the selected protocol to send the request to the payment gateway. The payment gateway forwards the request to the issueing bank.

The issueing bank sends the response back to the payment gateway, which forwards it to WebPOS and on to the payment server. If the authorization request is approved, the payment server then sends the HTML page generated earlier to the shopping cart for the customer to download the ordered items. Otherwise the payment server sends a message to the shopping cart to inform the customer that the authorization was denied and hence no access to the ordered items would be given. Just as with the Virtual Vendor approach, when the authorization has been approved by the issuing bank, the acquiring bank automatically issues a credit settlement request without requiring the payment server to issue a capture request.

Summary

The credit card payment security protocols (SSL and SET), and the credit card payment software that implement these protocols (such as the BankGate system) were discussed in this chapter. SSL offers channel security between two parties i.e. client and server. It uses public key encryption to negotiate a session key between a client and a server and uses this key to encrypt the data passing over this channel. The SET protocol not only offers channel security between the parties involved in a payment transaction, but it defines a complete credit card payment methodology by involving the cardholder, the merchant, and the acquiring bank. The approach that was used when commerce enabling WAPS combines both the SSL and the SET protocols to provide a secure transfer of payment data from the WAPS storefront to the payment server and through to the acquiring bank.

As discussed in this chapter, the SSL protocol was used to secure payment data between the shopping cart and the payment server, while the SET protocol was utilized between payment server and the acquiring bank. The main reason for the implemented approach was that SET requires authentication of all parties involved in a transaction. This then means that all customers must obtain Digital Certificates in order to be authenticated by merchants. Thus, in our approach the customer payment details are transferred to the merchant over an SSL connection, and then either SET or SSL can be used to send the authorization request to the acquiring bank depending on which payment method the merchant has decided to use i.e. either e-mail based or BankGate.

Chapter 6

Distributing Music over the Internet

The transition of audio technology from the analog to the digital domain and the rapidly improving hardware and software resources for handling digital audio data has enabled easy illegal copying and manipulation of audio files, especially of those sent over the Internet. This poses a serious challenge to the security and copyright protection of IP based audio streams and to various parties involved in the music production industry. Systems that facilitate the delivery of music over the Internet need to be aware of these challenges and methods that can be used to counteract them. This chapter discusses some copyright protection problems and introduces several proprietary solutions to the problem of unauthorized copying and usage of audio data. Finally some standards that are in place for copyright protection of music transmitted over the Internet will be presented.

6.1 Copyright Problems

The rising popularity of audio transmission on the Internet and the increasing efficiency of audio codecs promotes an ideal environment for music distributors to sell digitized and high quality compressed audio to interested customers over the Internet or dial-in connections (e.g. ISDN). With these emerging possibilities that enhance online music distribution, the intellectual property rights (IPR) of artists need to be protected. Also the current equipment (i.e. multimedia computers with CD-ROM and CD writers) makes it possible to produce digital copies of music tracks or video files without much problem and without the knowledge of the copyright holders. With these new technologies, digital audio data or any multimedia data can be duplicated without any loss of quality, where the copy becomes identical to the master and the process of copying does not affect the quality of the master itself. This leads to the possibility of uncontrolled copying, (reproduction) and selling of unauthorized copies.

This impacts music producers and distributors who stand to lose a lot of money. On the other hand, the rising popularity of the Internet commerce has forced the music distributors to consider it as one of their delivery mediums. This has led to the design and development of web-based systems, such as WAPS, that allow selling of high quality compressed music (e.g. ISO/MPEG Audio Layer III files, or any other high quality compression technique) via the Internet. But, the questions that automatically arise when services of this nature are launched are:

- How can digital multimedia data be sold without losing track of how the data is used?
- How can additional information (e.g. composer, artist, publisher, distributor, etc) be attached to the multimedia data?

The answers to the two questions lie in the hands of the technology developers and the music distribution industry at large. At the end of the day it is the distributor's decision that counts i.e. whether they decide to use the technology developed to protect the copyright of their music or not. However, several systems have been developed to answer these questions. These include: digital audio watermarking [41], Fraunhofer's Multimedia Protection Protocol [42], FASTAudio [43], and the Secure Digital Music Initiative [44], to name a few.

6.2 Intellectual Property Management Systems

6.2.1 Audio Watermarking

A digital watermark is an invisible identification code that is permanently embedded in a multimedia data file. In the case of music the term watermarking describes a method of transmitting arbitrary data along with a piece of music. Basically, watermarking is used to embed information (e.g. information about the artist, distributor, publisher, etc) into an audio track. Watermarking provides a way to carry additional information along with the music even if it is uncompressed. Figure 1 illustrates how watermarking is implemented within digital audio data.

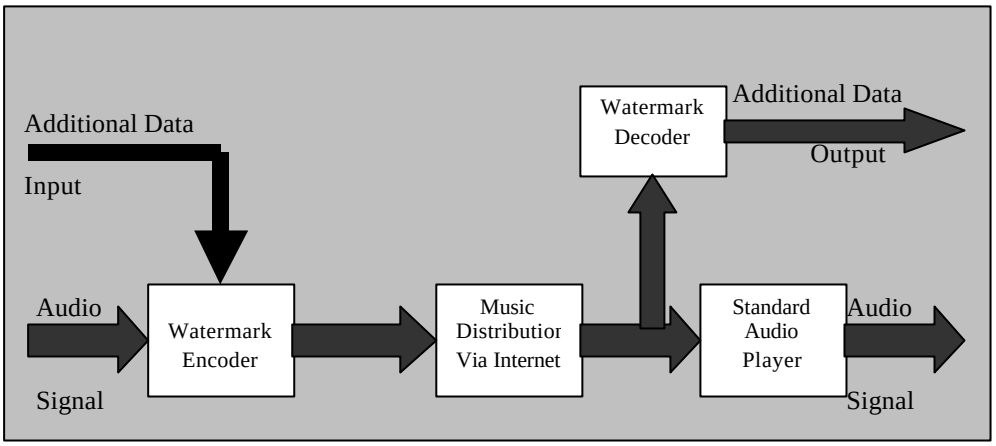

Figure 1: Watermarking in music

From Figure 1 above, the watermark encoder embeds the identification code into the original audio data, the original data being digital in nature (i.e. a digital sound file or sound coming from a sound device). The watermarked file is then transmitted over the network. On the receiving end, the watermark decoder extracts the identification code.

A digital watermark should be:

- Hidden and invisible.

  A watermark should be perceptually invisible and its presence should not interfere with the work being protected.

- Robust against signal processing operations.

  Robustness ensures that no one is able to remove or tamper with an embedded watermark without destroying or at least degrading the music.

- Directly connected to the music, not in the header.

The above requirements can be achieved by combining two fundamental techniques: "Psychoacoustics" and "Spread Spectrum" modulations [45]. The first ensures the inaudibility of the additional data by modeling properties of the human auditory system. The spread spectrum modulation provides high robustness against various signal-processing attacks by spreading the information over the entire time-frequency plane. Although digital watermarking is relatively new as a means of protecting intellectual property, the theories and technologies around it have been around for a while. These include: computer-based steganography (cryptography), spread-

spectrum communications, and noise theory. There are mainly two different categories of watermarking technology tools available [43]. The first is based on fingerprinted binary information, and the other is based on watermarking techniques developed at NEC research and the University of Catholique de Lourain (Belgium), which identifies documents by hidden number (fingerprints). There are various systems that implement audio watermarking. These include: SysCoP (System for Copyright Protection) [46], developed at the Fraunhofer Institute for Computer Graphics; Digimarc from Digimarc Inc. (Poland) [47]; and Argent from DICE [48]. These systems can encode additional identification information such as the author's name, publisher, etc.

6.2.2 FASTAudio

The Fast and Adaptive Security Technique for Audio over IP is a solution for securing audio transmission in a number of IP based applications. FASTAudio has been developed by the Fraunhofer Center for Research in Computer Graphics (CRCG). FASTAudio is an audio IP streaming technology that is adaptive and content-based. With FASTAudio, adaptive security functions are applied to selected portions of an audio stream depending on the network conditions and security requirements. Hence, any user can play the lower quality portions of a FASTAudio-encoded stream using any audio decoder, while only authorized users with FASTAudio decoders can play the audio stream at all levels of quality.

FASTAudio uses encryption technology on a significant portion of the audio stream for providing adjustable security. Once the audio data has been encrypted, it can then be securely transported over a network. At the receiving end, these significant portions are first decrypted and then combined with the remaining audio portions into the original audio stream. For further protection of copyright of the audio after decryption, the audio stream is digitally watermarked using the SysCoP audio watermarking systems. SysCoP (System for Copyright Protection) [46] is a file-based mechanism that applies a copyright marker (watermark) to a multimedia file that cannot be deleted without destroying the contents of the file.

Although FASTAudio is not a standard for securing the delivery of high quality audio over the Internet, it offers one possible solution of guaranteeing copyright protection,

especially for streaming audio applications. Some of the possible applications of FASTAudio include Music on Demand, Video on Demand, and Internet Radio. FASTAudio represents one of the simplest audio delivery methods and it offers a solution for secure and flexible audio streaming over IP, including multicast/broadcast transmission.

6.2.3 The Multimedia Protection Protocol

Developed by the Fraunhofer-Institute for Integrated Circuits (Fraunhofer IIS), the Multimedia Protection Protocol (MMP) introduces a convenient way to ensure copyright protection for all kinds of digital data. MMP was especially constructed for the copyright protection of compressed multimedia data e.g. MPEG Layer III files or H.263 video clips. For producing MMP files, Fraunhofer IIS has developed an MMP library that can be used to wrap (protect) multimedia files and unwrap them to use the contents. This library can then be inserted into every multimedia encorder and player.

When a customer asks his distributor for a multimedia file e.g. a piece of music, the distributor can use an MMP encoder (wrapper) to protect the requested track with MMP. During the MMP encoding process, an MMP file is produced and is personalized, meaning that only the target customer's MMP compliant multimedia player can use the data. If, for example, the MMP file is played by another person's player other than the one for which it was intended to, the file becomes useless. Hence, no one but the target customer would be able to use the contents of the file i.e. listen to music. MMP is basically a secure envelope wrapped around the multimedia data for its protection. The inside of the envelope is then ciphered to prevent unauthorized access. This envelope is represented by a header and followed by the MMP file. The MMP header carries information such as:

- The provider and distributor,
- The target customer for whom the file was composed, and
- Additional data describing the contents e.g. author's name, artist, composer, ISRC, information on the copyright holder, etc.

Figure 2 shows the principle structure of an MMP file. When protecting the contents of a multimedia file, both the header and some parts of the multimedia contents are ciphered.

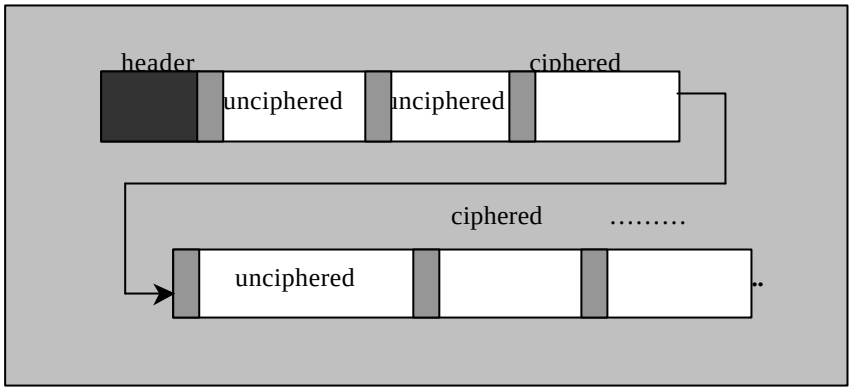

Figure 2: Structure of an MMP file

An MMP file is enciphered by using two different keys: one key, K_H, is used to encipher the MMP header and the other key K_B is used to protect the contents of the MMP file. The ratio at which how much data is to be left un-ciphered is adjustable to the needs of the distributor. More ciphering would obviously produce more load on the computer that wraps the data into the MMP envelope, while on the other hand increasing the safety of the MMP file. The MMP approach has promoted an efficient and a convenient way for music distributors to sell their music securely on the Internet without fear of copyright infringements. MMP is part of the ISO MPEG-1 Layer III software developed at the Fraunhofer IIS (WinPlay3 for Windows and MacPlay3 for Macintosh). Thus, if a distributor intends distributing his music in MPEG-1 Layer III format, all he has to do is envelope the music into MMP files using an MMP encoder. These files can then be unwrapped by the target customers using any MMP compliant player.

6.3 Ethics

The Internet has on many occasions proven to be a good testing ground for the applicability of existing and new laws. As new technologies develop, there is a continuous struggle to determine the legality of various applications of that technology, which often involves groups with varying goals. One such technology is the distribution of music over the Internet. There are several standards for transmitting

music over the Internet, each of which puts into focus various issues of the legality of such actions. Various groups have also come to the forefront of the debate involving the distribution of music over the Internet. One typical example of such a group is the Secure Digital Music Initiative (SDMI).

The SDMI was announced at the end of 1998 by the Recording Industry Association of America and Japan (RIAA and RIAJ) and the International Federation of the Phonographic Industries (IFPI). SDMI is the music industry's effort to ensure copyright protection of music over the Internet. Through this initiative the record and technology solution providers hope to create a specification that can be extensibly embedded within any online music delivery technology to ensure it is secure. The major driving force for this initiative has been the widespread use of formats like MPEG Layer III for pirating music. So, the SDMI was launched in order to establish a secure way of trading and distributing music (and other multimedia content) over the Internet.

Summary

Though the technological advances might seem to suggest that the distribution of CD quality audio over the Internet is a solved problem, there are still barriers that have to be overcome before there can be widespread access to vast music catalogues. Furthermore, music that has already been distributed online is vulnerable in the light of these advances. Hence, systems and services have to be built, which will guarantee that the music copyright owners do not lose complete control of their wares. It is very difficult to build a completely foolproof solution, but there is a great need for technology providers to design systems that will maximize the incentives for customers to play by the rules. There is however a belief that the kinds of systems described in this chapter do enable consumers to use music in new ways. These systems also enable music distributors to distribute their music more confidently and securely without fearing for copyright infringements. The distribution of high quality audio over the Internet has revolutionized the entire music industry and has enabled content owners (or music distributors) to engage in entirely new kinds of services.

Chapter 7

Conclusion

The primary ways of ordering music have been through the mail, from the music store, or on the Internet. Purchases made through the mail and Internet are partially convenient because they do not require individuals to leave their homes. However, once a purchase has been made via these means, a consumer has to wait between two and four weeks for the goods to be delivered. Additionally, if ordered by mail or Internet, consumers are unable to listen to samples of the music prior to their purchase. Hence, they may prefer to get in their cars, drive to the nearest music store, make a purchase, and leave with the goods in hand. The growth of the WWW has provided new opportunities for distributing and selling recorded music on the Internet. This has enabled music distributors to utilize the Internet as an important delivery medium, i.e. for advertising, ordering, or promoting music as well as delivering digitally compressed music.

The electronic transfer of music over the Internet provides a more convenient and accessible alternative to music consumers. It has the potential to allow individuals to browse, listen to, purchase, and receive music at their residence in a relatively short time period.

7.1 Thesis Overview

The main goal of this thesis was to investigate the technological requirements for an ideal Web music store and suggest an optimal way for setting it up. The basic purpose of an electronic music store is exactly the same as that of a physical store i.e. to bring potential buyers and sellers together, whereby:

- Sellers offer their goods and buyers order them, and
- Buyers make a payment and sellers deliver the goods.

In the above actions, the two parties have specific security requirements, namely integrity, confidentiality, and availability. Essentially, an online music store allows

users to sit at their computers, log onto a Web site, search for music of their interest, listen to music samples prior to purchase, select and purchase items, and download the digitally compressed CD-quality songs to their hard-drives. Putting together an enterprise-level application for an online music store involves design and integration of various technologies that play specific roles in a distributed computing environment. Five Internet-related technologies, which form the core of the model for WAPS, were identified, discussed, and analyzed in this thesis. These include various methods for accessing remote databases, Internet audio codecs, streaming audio protocols, payment security protocols, and the Intellectual Property Management and Protection (IPMP) of multimedia data over the Internet.

7.1.1 Methods for Accessing Remote Databases

Since the Internet is inherently distributed in nature, a distributed topology is a prerequisite for building Internet applications that access Web databases remotely. As discussed in Chapter 2, a 3-tier architecture approach is the foundation upon which such applications are built. This approach has several added advantages over the traditional 2-tier approach. These include: reduced network load; clear separation of user interface control and data presentation from the application logic; re-definition of storage strategy that doesn't influence clients; business objects and data storage are brought together as close as possible, reducing network load; dynamic load balancing: if bottlenecks in terms of performance occur, the server process can be moved to other servers at runtime. The 3-tier architecture is composed of a client-tier, an application server-tier, and a data storage-tier. The interaction between the application server and the data storage tiers is handled by database protocols, such as ODBC or JDBC.

The question remained of how to manage the interaction between the client and the application server tiers. This question was used in Chapter 2 as the driver for an analysis of various techniques, provided by Java programming language, that can be used to facilitate data communication between client and the application server tiers. However, the results also apply to the more general field of distributed applications. Four approaches, using different protocols and mechanisms to enable interaction between a client (Java applet) and a remote corporate server application were identified and analyzed. The approaches based on HTTP/CGI and those that

implement straight IP sockets suffer from significant shortcomings, such as data marshalling, performance, and maintainability. In Chapter 2, it was shown that the RMI mechanism provides a very efficient and lightweight communication between remote Java objects and seemed to be the preferred solution for distributed applications that are completely in Java.

On the other hand, CORBA with its support of a range of major programming languages, provides a useful set of fundamental services for distributed systems, including Naming, Trading, and Event services. CORBA seems to be the choice when it comes to the larger scale distributed applications or to the access of legacy applications not implemented in Java. Both the RMI and the CORBA approaches were tested for preliminary performance results by means of a prototype of a distributed audio database management and purchasing system, developed in this project. As outlined in Chapter 1, the system is based on a client/server architecture where audio documents (text, audio, still images, etc) are stored and accessed in a manner transparent to its client applications. Access to audio information is performed by querying a relational database and the set of audio items are integrated and transparently distributed to the users. For the test, several queries were performed, with each query performed three times and the average performance time measured. The queries consisted of issuing SQL statements to select each and every field from the database, where between 50 and 100 records were returned from each query.

The performance results for the RMI and CORBA approaches showed that CORBA's performance was faster than that of RMI. The reason for this is that RMI uses Java object serialization for passing results between client and server applications, while CORBA does not i.e. its uses normal data types when passing results. The latest release of RMI over IIOP by JavaSoft provides a core ORB including an IDL compiler, which uses a lightweight protocol for invocation of methods on remote Java objects by non-Java clients, and vice-versa. This has created the possibility of integrating RMI and CORBA, so that either of these approaches can be chosen where they are most appropriate.

7.1.2 Internet Audio Codecs and Streaming Audio Protocols

The Internet does not offer guaranteed bandwidth and there is always a variable delay involved when transmitting audio files in real-time. Because of this unknown overhead, an audio file's bandwidth must be between 6 and 56 kbps if it is to be streamed across the Internet, with subjective quality kept as high as possible within the given bit rate. There are basically two solutions to the bandwidth problem for applications that utilize audio on the Internet. These include careful selection of suitable audio codecs and appropriate transport protocols. Chapter 3 presented various audio codecs that can be used in low band connections, and it presented the results of blind subjective tests, evaluating the audio quality of 5 industry standard, and widely used codec families (which constituted 13 codecs) against a limited bandwidth reference. The study of these codecs is unique, since this was the first time that all of these codecs have been compared in a single formal test.

The codecs tested operated at bit rates between 8 kbps and 56 kbps. The tests examined the quality of the codecs against increasing bandwidth and were conducted in accordance with methodologies outlined in the ITU-R Recommendation BS.562.3. Three test materials were selected and encoded in each of the 13 codecs. These included an acoustic guitar; a rock piece with a good mix of synthesizers, bass and drums; and a female voice speech. Eighteen men and two women then listened to and evaluated the quality of the samples produced by the 13 codecs. The listeners were given a 56 kbps MPEG-1 Layer III file as a reference, and were asked to rank the quality of the samples against it. The 56 kbps MPEG-1 file was used as a reference instead of the original recording because it is closer in quality to the compressed samples. This encouraged the listeners to use the full grading scale given, from 1 to 5.

The results of the tests showed that the codec families are clearly delineated with respect to quality. The ranking of the individual codecs with respect to quality is (in order of merit):

1) RealAudio/44 and TwinVQ/40
2) MPEG-1 LII/56, MPEG-1 LIII/40, and MsAudio/40
3) MPEG-1 LII/24, MPEG-1 LIII/16, and RealAudio/16
4) MsAudio/16 and TwinVQ/16

5) MPEG-1 LIII/8, MsAudio/8, and TwinVQ/8

Overall, the RealAudio and TwinVQ codecs operating at 40 kbps had the best sound, while others (except MPEG-1 Layer III, MsAudio, and TwinVQ, all operating at 8 kbps and rated as "Poor") performed between "Fair" and "Good" on the quality scale. These tests only considered audio quality with respect to bit rate. Other factors such as computational complexity, delay, sensitivity to bit errors, and compatibility with existing systems, usually need to be considered in the selection of a suitable compression algorithm for a particular application.

As mentioned earlier, the various audio compression schemes, which markedly reduce audio file sizes and lower bandwidth requirements, present one solution to real time streaming for applications that stream audio over Internet. The other alternative is the use of streaming audio protocols. Chapter 4 presented several standard and proprietary multimedia transport protocols suitable for the transmission of real-time media on the Internet. In Chapter 4, it was also indicated that streaming media technology developers basically have to choose between two packet-transport protocols, neither of which is particularly effective for real-time data delivery. Some products rely on the User Datagram Protocol (UDP) to send packets as quickly as possible with no guarantee of delivery. Others use the Transmission Control Protocol (TCP), ensuring that all packets arrive in the proper order. However, TCP introduces transmission overhead, which can slow data transfer when congestion occurs. Thus, using either transport, packets frequently do not reach the destination player in time to put together smooth audio playback, especially on low band Internet connections. Hence, specialized protocols, essential for the success of streaming media, needed to be developed. Some of these protocols, presented in Chapter 4, include RTP/RTCP, RTSP, and the Real Network's proprietary PNA protocol.

Some experimental tests using these protocols were carried out in the presence of real-time play out requirements. These testing activities involved the transmission of a 20 kbps RealAudio stream from a producer to a consumer through various network environments. The main goal of these tests was to evaluate the applicability of the HTTP, RTSP, and PNA protocols, in order to satisfy real-time play out requirements with respect to the transmission of compressed low bit rate audio

streams. These tests were conducted within different networking environments, which included an Ethernet based LAN, a dedicated WAN connection, and the Internet. The parameters of interest were the bandwidth requirements and the achievable bit rate. Attention was restricted to RealAudio since, as seen on the qualitative tests from Chapter 3, it guarantees a better quality in audio reproduction at a higher compression rate compared to other compression schemes, such as TwinVQ, MsAudio, MPEG-1.

There were no problems using all tested protocols on both the LAN and the dedicated connection/WAN environments. There was no substantial delivery delay, and on both the LAN and WAN environments the bandwidth was widely sufficient to put out smooth audio playback without any noticeable gaps between packets. In the Internet environment, the performance of the protocols was greatly influenced by the available bandwidth. Though the performance of HTTP on average was slightly higher than that of the RTSP and PNA protocols, noticeable gaps could be heard when the link became saturated with other network traffic. Overall, the RTSP and PNA protocols offered a much more controlled and smoother playback, and allowed one to move back and forth during playback of the audio stream, a feature that is not possible with HTTP.

7.1.3 Payment Security Protocols

Security is the primary concern of people and businesses that commit to using the Internet for conducting business transactions. An Internet transaction is conducted between two parties, and each party needs to have a certain level of trust in the other. For electronic transactions to be the primary mechanism for conducting business, the level of trust needs to be as high as, if not higher than, that of a person-to-person transaction. Most importantly, the parties involved in an electronic trade also need to have a high degree of trust in the medium used for information transfer, namely the Internet. Several payment security protocols, and payment systems that utilize these were introduced in Chapter 5. These included the SSL and SET protocols, and the Virtual Vendor and the BankGate systems. These payment security protocols provide important end-to-end security features in a way that persists even if payment information passes over more than one secure channel. These features include:

- Digital signatures on transaction data sent by customers to merchants, from merchants to payment gateways, and from payment gateways to merchants.

- Non-repudiation of transaction data sent by customers to merchants, from merchants to payment gateways, and from payment gateways to merchants.

- Secondary encryption, allowing information to be encrypted by its originator (for example, a customer), sent off to another participant (for example, a merchant), but decrypted only by a third-party (for example, the payment gateway).

These functions are required by electronic payment systems in order to fulfill the needs of all participants (customer, merchant, and payment gateway). These needs involve:

- Keeping both the customer credit card information and the order information private.

- Digitally signing all communications between all parties for the purpose of guaranteeing authorship, and to adding non-repudiation to the communications (i.e. the customer should not be able to deny having made an order).

7.1.4 Issues and Concerns about Distributing Music over the Internet

Though technological advances have furthered the ease with which music can be distributed over the Internet, as well as increased the quality of music being transmitted, there are still many issues that need to be addressed concerning the distribution of high quality audio on the Internet. Chapter 6 introduced some of the technological standards and groups such as the SDMI, who are at the forefront of the debate involving the distribution of music over the Internet. These debates not only reflect the technology itself, but the interest of those who stand to gain and lose it. At

the bottom line the protection of the rights of composers, artists, and music publishers, is a key concern of music distribution over the Internet.

There are plenty of individuals hoping to gain tremendous revenue opportunities from network music distribution, and all the players in the electronic commerce value chain including musicians, recording artists, distributors, marketing groups, and Web developers, want to benefit, as technological advances improve the quality of network music distribution. These opportunities, however, are contingent upon important public policy advances that ensure the development of a secure and business-friendly environment. For example, no businesses will invest in Internet music distribution unless there exist reasonable guarantees that they will be appropriately paid for their services and goods. Thus, in order for this to happen, public policy advances need to be applied in the following areas: cryptographic security, licensing mechanisms, copyright protection, and secure payment systems. This implies that any system that facilitates Internet music distribution has to incorporate most (if not all) of these technologies.

7.2 End-product

The Web-based audio purchasing system, developed in this project, proposes a novel way of integrating various Internet-related technologies, such as audio codecs, security protocols, multimedia transport protocols, and methods for accessing remote databases, into an appealing and reusable online music store. The design of WAPS is based on a distributed topology. The system is low cost, since it has been built using standard and widely used technologies. Its modular structure enables music distributors to upgrade and maintain it with relative ease. The system is also applicable to the real world, and has been tested by the International Library of African Music (ILAM), of Rhodes University. The following are some of the requirements, mentioned in Chapter 1, which we felt the system has fulfilled:

- Providing flexible online access to a large music database by implementing the industry standard distributed architectures, RMI and CORBA.

- Supporting ad-hoc queries for users to find music material of interest based on bibliographic information.

  This was achieved by using the traditional "alphanumeric" technique for retrieving music information from the music database utilizing RMI, CORBA, and the JDBC communication methods.

- Providing support for the standard and the most widely used audio codecs and streaming audio protocols, so as to facilitate real-time preview of short audio excerpts provided by the online store.

- Providing a secure online payment mechanism, achieved by implementing a combination of both the SSL and SET protocols in order to secure payment information from the customer to the merchant, from the merchant to the payment gateway, through to the bank and then back to the merchant.

The fulfillment of the above requirements has led to an analysis of various tools and protocols involved, and also to the development of an online application that allows users to search for music of their choice from a database, receive excerpts from its collection on demand, to order securely, and to have their music delivered in digital form via the Internet.

7.3 Future Directions

The following points constitute some of the areas or technologies that will improve music distribution over the Internet.

- **Distributed Databases**

A flexible, distributed client/server architecture is a good solution to the problem of servicing processes on the Internet depending on the network load. This load also affects access to databases, that is, the transfer of data. A simple solution to this is the idea of a distributed database. A database distributed over several hosts, each one located in different geographical locations, is a viable solution. The database servers would then synchronize their contents during times of low

network load, and clients would then be able to choose their "closest" server. Thus catalogue information would be duplicated in different geographical locations.

- **Content-based Retrieval Techniques**

A large and well-organized audio database is a very important component in an online music store. However, the issue of searching the database using the "alphanumeric" or text-based technique only is a major limitation due to the natural complexity of sound or music, and good alternatives are based on content-based retrieval techniques. These would enable one to search a database by specifying acoustic or perceptual attributes that are obtained directly from sound analysis.

- **Multiparty Payment Systems**

Currently, there is still a need to establish a standard micro payment system that is flexible and secure enough to handle network music transactions. A sophisticated micro payment system would ensure that music distributors, authors and artists are properly compensated for their services and goods. Such a micro payment system would provide the technology to back-up intellectual property rights and copyright infringements. Hence, in order to fully encourage fair electronic transactions, payment systems must move towards an open, multiparty model in which Internet music distribution would be based on voluntary compliance and enforceable law, whereby all parties involved (including customers, merchants, owners, artists, producers, etc) benefit.

In conclusion, the advantages of electronically selling, transporting, and downloading music over the Internet are numerous. Music distributors and artists are able to reach a much broader audience and decrease their production costs. Consumers on the other hand, benefit from the ability to easily compare prices of various distributors, with the convenience of home-based shopping and faster delivery of the product. At present, these advantages are slowly being realized by interested parties, especially the larger music producers such as Sony, due to the various issues of concern highlighted in this thesis. Hence, further technological advances are necessary to ensure that:

(1) Costs are reasonable for both the consumer and the provider,

(2) There are adequate incentives to encourage the use of this resource including copyright protection, and
(3) Secure environments are provided for both the payment and transmission products.

In addition, when providing networked music commerce, appropriate and effective technologies need to be selected and coordinated to produce an environment that further facilitates the achievement of the above goals. Under these conditions, networked digital music distribution is likely to become a major player in the music industry.

References:

[1] C. McGrath and B. Fowler, "Multimedia Databases", ONLINE - http://ils.unc.edu/~fowll/multimedia/index.html.

[2] S. McCline, "Object databases vs. Relational databases", IDC Bulletin No. 14821E, August 1997, ONLINE - http//:www.cai.com/products/jasmine/ analyst/ide/1482E.htm.

[3] A. Amir et al, "Key to Effective Video Retrieval: Effective Cataloguing and Browsing", ACM Multimedia 98, ONLINE - http://www.acm.org/sigm/MM98/ electronic_proceedings/ponceleon/index.html

[4] J. Patrick van Metre, "Common Gateway Interface", ONLINE - http://ei.cs.edu.vt.edu.

[5] Distributed Technologies GmbH, "3- and n-tier Architectures", 1998, ONLINE - http://www.corba.ch/e/3tier.html.

[6] JavaSoft, "JDBC™ Guide: Getting Started", Sun Microsystems Inc., ONLINE - http://java.sun.com/products/jdk/1.1/docs/guide/jdbc/getstart/intro.doc.html.

[7] JavaSoft, " Java Servelet API", Sun Microsystems Inc., ONLINE - http://java.sun. com/products/servlet/index.html.

[8] P. Mohseni, "Exploiting distributed Java Computing with RMI", NC-World Magazine, ONLINE - http://www.ncworldmag.com, January 1998.

[9] OMG, "CORBA Overview", Object Management Group Publications, ONLINE - http://www.omg.org/corba/whatiscorba.html.

[10] Jakob Nielsen, "Response Times: The three important limits", ONLINE - http://www.useit.com/papers/responsetime.html.

[11] TechMetrix Reaserch, “Internet architectures and performances”, ONLINE - http://www.techmetrix.com, November 1998.

[12] JavaSoft, “idl2java compiler”, Sun Microsystems Inc., ONLINE - http://java.sun.com.

[13] D.Yen Pan, “Digital Audio Compression”, Digital Technical Journal Vol. 5 No. 2, pp. 1-12, Spring 1993.

[14] K. Lampert, “Methods to Reduce the Bandwidth Requirements of MPEG-1 Layer II Audio Data for Transmission Speeds of less than 28.8 kbps”, MSc thesis, University of Miami, 1997.

[15] T. Stachowiak, “Multiplatform, Desktop Videoconferencing Systems for the Internet”, MSc thesis, Graduate School of Syracuse University, December 1997.

[16] B. Feiten et al, “Dynamic Scallable Audio Internet Transmission”, Presented at the 104th AES Convention, Amsterdam, May 1998.

[17] FhG-IIS, “Frequently Asked Questions”, ONLINE - http://www.iis.fhg.de /departs/ amm/layer3/sw/index.html.

[18] A. P. Smith, “Perceptual Audio Coding”, ONLINE - http://www.ug.ecs.soton.ac.uk/ ~sap296/media/perceptual_audio_coding.htm.

[19] Fraunhofer IIS-A, “MPEG-2 AAC”, ONLINE - http://www.iis.fhg.de/amm/ techinf/aac/index.html.

[20] Fraunhofer IIS-A, “MPEG-4”, ONLINE - http://www.iis.fhg.de/amm/techinf/ mpeg4/index.html.

[21] MP3’ Tech, “TwinVQ”, ONLINE - http://freeflight.cockpit.be/mp3tech/ vqf.html.

[22] RealNetworks, "RealAudio G2 Music Codec", ONLINE - http://www.real.com.

[23] Dolby Laboratories Inc. Press Realease, " Dolby introduces audio format for the Internet", ONLINE - http://www.dolby.com/press/imadnet.html.

[24] ITU-Recommendation BS.562.3, "Subjective assessment of sound quality" ONLINE - http://ext-www-proxy.itu.ch/itudoc/itu-r/rec/bs/562-3.html.

[25] K. Lee and W. Suh, "Real-time Audio on the Internet", University of Pennsylvania, ONLINE - http://www.seas.upenn.edu/~ksl/Classes/TCOM500/InternetAudio/index.html, September 1996.

[26] Gilbert A. Soulodre et al, "Subjective Evaluation of State-of-the Art 2-Channel Audio codecs", AES preprint 4740, AES 104th convention, May 16-19, 1999, Amsterdam.

[27] A. Covell, "Streaming Audio and Video on the Internet", Network Computing, CMP Media Inc., ONLINE - http://techweb.cmp.com/nc/712/712corpnetSTREAMING.html.

[28] C. Liu, "Moltimedia over IP: RSVP, RTP, RTCP, RTSP", ONLINE - http://www.cis.ohio-state.edu/~jain/Cis788-97/ip_multimedia/index.html, January 1998.

[29] JavaSoft, "Java Media Framework API", Sun Microsystems Inc., ONLINE - http://java.sun.com.

[30] K. Laik and M. Baker, "Measuring Bandwidth", Stanford University, ONLINE - http://mosquitonet.Stanford.edu/~laik/pro…blication/infocom/199/html/netliner.html, 1999.

[31] Fraunhofer-CRCG, "SLICE: Secure Light – Weight Information Commerce", Fraunhofer Center for Research in Computer Graphics, ONLINE - http://www.crcg.edu/projects/slice.html.

[32] K. Lamond, "Credit Card Transactions – Real World and Online", ONLINE - http://www.virtualschool.edu/mon/ElectronicProperty/klamond/ credit_card.htm.

[33] IBM, "Java Network Security", IBM International Technical Support Organization, November 1997.

[34] Visa and MasterCard, "Business Description: Secure Electronic Transaction Specification", version 1.0, May 1997, ONLINE - http://www.visa.com/cgi-bin/vee/nt/econmm/set/main.html.

[35] Virtual Vendor CC, " Virtual Vendor Transaction Management Service", ONLINE - http://vvendor.co.za.

[36] DigiCash, Digicash home page: ONLINE - http://www.digicash.com.

[37] Netscape, "SSL Version 3.0", ONLINE - http://www.netscape.com/newsref/ std/SSL.html.

[38] BankGate™, "Electronic commerce payment processing software for the Internet Merchant", ONLINE - http://www.bankgate.com.

[39] On the Internet Magazine (Oti), "Commerce Enabling a Web site", issue 5, July/August 1998, pp. 18 – 23, ONLINE - http://www.oti.co.za

[40] Baltimore, "J/SSL Version 1.0", Baltimore Technologies Inc., ONLINE - http://www.baltimoreInc.com.

[41] Ingemar J. Cox and Matt L. Miller, "A review of Watermarking and the Importance of Perceptual Modeling", proceedings of Electronic Imaging'97, February 1997.

[42] N. Rump, " Copyright Protection of Multimedia Data: The Multimedia Protection Protocol (MMP)", Fraunhofer – IIS, Erlangen, October 1996, ONLINE – http://www.iis.fhg.de/amm/techinf.

[43] Fraunhofer-CRCG, "FASTAudio: Fast and Adaptive Security Techniques for Audio over IP", Fraunhofer Center for Research in Computer Graphics, ONLINE – http://www.crcg.edu/projects/audio.html.

[44] N. Rump et al, "The Secure Digital Music Initiative", Fraunhofer – IIS, Erlangen, February/March 1999, ONLINE - http://www.iis.fhg.de/amm/.

[45] Fraunhofer-IIS, " Intellectual Property Management and Protection (IPMP)", ONLINE - http://www.iis.fhg.de/amm/techinf/ipmp/index.html.

[46] Fraunhofer-ICG, " SysCop: The System for Copyright Protection", Fraunhofer Institute for Computer Graphics, Darmstadt, Germany, ONLINE - http://www.iis.fhg.de/.

[47] Digimarc, ONLINE - http://www.digimarc.com/.

[48] DICE, ONLINE - http://www.digital-watermark.com/.

[49] V. Jacobson, "Pathchar - a tool to infer characteristics of Internet paths", ONLINE - ftp://ftp.ee.lbl.gov/pathchar/msri-talk.pdf, MSRI, April,1997.

[50] PGP, ONLINE - http://web.mit.edu/network/pgp.html.