A novel approach to local multimedia sharing

Master’s Thesis in Computer Systems Engineering

Tim Motmans & Sander Bel
A novel approach to local multimedia sharing

Master’s Thesis in Computer Systems Engineering

School of Information Science, Computer and Electrical Engineering
Halmstad University
Box 823, S-301 18 Halmstad, Sweden

May 2011
Preface

It all started in August 2010 when Sander got the idea to participate in an Erasmus project in Grenoble, France. However, after a while, this seemed not possible because the courses were given in French and they were only Bachelor degree. Since Sander was obliged to attend Master degree courses together with the Master thesis project, Grenoble was not suitable for his kind of studies.

While Sander was still trying to find a well-suited place to study when the new school year started, Tim got word of his idea to study abroad. Since Tim already went abroad 2 years ago to study in Seinäjoki, Finland for his Bachelor degree and had great experiences with it, he was really interested to study abroad again. Tim even wanted to attend his Master degree Erasmus project in the same city.

Since we were colleagues in the same class, the idea came up that we both could study abroad in Seinäjoki. But after contacting the Seinäjoki school of Technology, which is a faculty of the University of Applied Sciences, again some problems came up. This time courses were given in English, but there were no Master degree courses available. Our international supervisor Patrick Colleman and we concluded to search for another place.

Finally, after contacting the Högskolan I Halmstad located at the southwest coast of Sweden and investigating all possible courses we could follow, this school was the best place to end our Master studies in Electronics-ICT. The school offered a lot of Master degree courses related to our studies and field of interest, which also could be attended in English, and on top of that Sweden was a country we have never been to. After signing a lot of documents, we were ready to leave to Halmstad at the 15th of February 2011.

Now that our stay is almost at its end, we have to admit that we do not regret anything. We have met a lot of nice people and made a lot of friends. The both of us also got to know a lot about the Swedish culture and we have even skied.

There are a lot of people we want to thank for making this adventure unforgettable. First of all we want to thank our partner during the project. We never had a fight, made a lot of fun together, and we could cooperate very fine during the whole project.

Secondly we want to thank Matthias, the third Belgian guy who went to Halmstad, but did another project, and all other international students living near our house for all the good time we had together.

Our parents are the ones who deserve the most credit because they made it all possible to go on Erasmus, a once in a lifetime experience, we can tell you!

We did our traineeship at the Högskolan I Halmstad so we would like to thank the Högskolan I Halmstad, especially Tony Larsson. He made it possible to do such an interesting project and supported us very well while researching and writing our thesis.

Finally we want to thank everyone at the Katholieke Hogeschool Kempen in Geel, Belgium. This school made this all possible and made the arrangements to go on Erasmus, especially Agnes Dilliën, international coordinator, and Patrick Colleman, our Belgian supervisor.

In case we forgot someone, thank you too, we really want to thank all people who made our trip this excellent!

Tim Motmans & Sander Bel
Halmstad University, May 2011
A novel approach to local multimedia sharing
Abstract

Sharing locally stored media files like music, videos and pictures has not been user-friendly for a long time now. Nowadays people, when they know where the media is stored, have to use the complicated network shares or external storage solutions like USB sticks, hard drives or even CD/DVDs to share media across different users.

When the users do not know where the media is stored, they have to use Internet-based peer-to-peer applications like LimeWire, KaZaa or the Gnutella-network, which requires searching and downloading the media first, before being able to actually make use of it.

But what if you do not have an Internet connection, do not want to mess around with external storage solutions nor want to wait while downloading or copying from a (network) device, but still want to make use of the media stored on another computer system?

Indeed, nowadays there is not any easy solution that provides a very user-friendly, fast and responsive, flexible and stable solution for this. This problem brought us to our research question: “Is there a very easy solution for sharing or watching media throughout the local network?”

After some research we stumbled upon some state-of-the-art technologies, which came very close to what we wanted to achieve, however, still having quite some drawbacks, not suitable as a solution for the problem mentioned above. We decided to innovate and tried to find a solution without any drawbacks while still being very user-friendly.

We achieved quite good research results showing that:

- Using a client-only network was the most efficient and flexible way to provide a stable network structure;
- Java was the best programming language to provide a cross-platform application;
- For compatibility with the media sharing itself an object-oriented based indexing storage structure, like db4o, yielded the best flexibility and speed in comparison with SQL or other technologies;
- The streaming of the media could be achieved best by making use of Java VLC libraries.

The user-friendliness of the demo application that we created was also very good, only a few clicks are sufficient to share your media across the network, no need to bother about user rights and so on.

We can conclude that our research can be the base of a very successful innovative media sharing system and strongly believe, with some more adjustments in the future, that it has potential to become a very popular application along the media sharing industry.
A novel approach to local multimedia sharing
# Contents

**PREFACE** ............................................................................................................................................... 1

**ABSTRACT** ........................................................................................................................................ 3

**CONTENTS** ....................................................................................................................................... 5

**LIST OF ABBREVIATIONS** .................................................................................................................. 9

## 1 INTRODUCTION ................................................................................................................................ 13

1.1 **APPLICATION AREA AND MOTIVATION** .................................................................................. 13

1.2 **PROBLEM STUDIED** ................................................................................................................... 13

1.3 **APPROACH CHOSEN TO SOLVE THE PROBLEM** ..................................................................... 14

1.4 **THESIS GOALS AND EXPECTED RESULTS** ............................................................................. 14

## 2 BACKGROUND ................................................................................................................................... 15

2.1 **DIRECTORIES** ............................................................................................................................. 15

2.1.1 **Directory services** ................................................................................................................... 15

2.1.2 **Hierarchical database model** .................................................................................................. 16

2.1.3 **Directory objects** .................................................................................................................. 16

2.1.4 **Comparison with databases** .................................................................................................. 17

2.1.5 **Conclusion** ............................................................................................................................ 18

2.2 **DATABASES** ................................................................................................................................. 19

2.2.1 **Database management systems** ............................................................................................ 19

2.2.2 **Data models** ........................................................................................................................... 20

2.2.3 **Global structure** .................................................................................................................... 20

2.2.4 **Database objects** .................................................................................................................. 21

2.2.5 **Conclusion** ............................................................................................................................ 21

2.3 **PEER-TO-PEER NETWORKS** ...................................................................................................... 22

2.3.1 **Architecture** .......................................................................................................................... 23

2.3.2 **Structured systems** ............................................................................................................... 25

2.3.3 **Unstructured systems** ........................................................................................................... 26

2.3.4 **Indexing and resource discovery** ........................................................................................... 27

2.3.5 **Advantages and disadvantages** ............................................................................................. 27

2.3.6 **Conclusion** ............................................................................................................................ 28

2.4 **ORB** ................................................................................................................................................ 29

2.4.1 **Introduction** ............................................................................................................................ 29
### 3 SOLUTION IDEA TO BE INVESTIGATED

#### 3.1 CHOOSING THE RIGHT NETWORK STRUCTURE

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.1</td>
<td>Introduction</td>
<td>39</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Master-client based network</td>
<td>39</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Clients-only based network</td>
<td>42</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Conclusion</td>
<td>43</td>
</tr>
</tbody>
</table>

#### 3.2 CHOOSING THE PROGRAMMING LANGUAGE

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1</td>
<td>Introduction</td>
<td>44</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Approaches</td>
<td>44</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Challenges</td>
<td>46</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Conclusion</td>
<td>47</td>
</tr>
</tbody>
</table>

#### 3.3 STORING THE INDEXED LISTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1</td>
<td>Introduction</td>
<td>48</td>
</tr>
<tr>
<td>3.3.2</td>
<td>db4o</td>
<td>48</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Conclusion</td>
<td>50</td>
</tr>
</tbody>
</table>

#### 3.4 DISCOVERING OTHER CLIENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1</td>
<td>Introduction</td>
<td>51</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Broadcasts</td>
<td>51</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Multicasts</td>
<td>52</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Conclusion</td>
<td>52</td>
</tr>
</tbody>
</table>

#### 3.5 STREAMING THE MULTIMEDIA FILES

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.1</td>
<td>Java streaming libraries</td>
<td>53</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Java binding for VLC media player</td>
<td>53</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Example code of Java bindings for VLC media player</td>
<td>53</td>
</tr>
<tr>
<td>3.5.4</td>
<td>General setup of streaming server</td>
<td>54</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Media options example: HTTP</td>
<td>54</td>
</tr>
</tbody>
</table>
4 DETAILED DESCRIPTION OF THE INVESTIGATED SOLUTION ........................................... 57
  4.1 PHYSICAL OR HARDWARE STRUCTURE .................................................................. 57
  4.2 LOGICAL OR SCHEMATIC STRUCTURE ................................................................. 58
    4.2.1 Class diagrams ......................................................................................... 58
    4.2.2 Threading ................................................................................................. 60
    4.2.3 Implementation of db4o ........................................................................... 64
    4.2.4 Implementation of multicasts ................................................................. 68
  4.3 BEHAVIOURAL OR FUNCTIONAL DESCRIPTION .................................................. 70
    4.3.1 Search function ....................................................................................... 70
    4.3.2 Compression of the database .................................................................. 72
5 TEST OR ANALYSIS OF SOLUTION .............................................................................. 75
  5.1 PROTOTYPE SETUP .......................................................................................... 75
  5.2 INDEXING ........................................................................................................ 75
  5.3 DB4O .............................................................................................................. 76
  5.4 CLIENT DISCOVERY ....................................................................................... 78
  5.5 USER-FRIENDLY GUI .................................................................................... 78
  5.6 CROSS-PLATFORM COMPATIBILITY .............................................................. 79
6 CONCLUSIONS ......................................................................................................... 81
7 FUTURE WORK .......................................................................................................... 83
  7.1 ID3 TAGS ........................................................................................................ 83
  7.2 STREAMING ................................................................................................... 83
  7.3 DOCUMENTS ................................................................................................ 84
8 REFERENCES ............................................................................................................ 85
9 BIBLIOGRAPHY ........................................................................................................ 87
10 APPENDIX ............................................................................................................... 89
  10.1 CLASS HIERARCHY OF JAVA BINDINGS FOR VLC MEDIA PLAYER .................. 89
  10.2 CODING EXAMPLE OF ID3 TAGS WITH JAUDIOTAGGER ............................ 93
### List of abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAC</td>
<td>Advanced Audio Coding</td>
</tr>
<tr>
<td>AD</td>
<td>Active Directory</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASP</td>
<td>Active Server Pages</td>
</tr>
<tr>
<td>CD</td>
<td>Compact Disc</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>DBS</td>
<td>Database System</td>
</tr>
<tr>
<td>DHT</td>
<td>Distributed Hash Table</td>
</tr>
<tr>
<td>DLNA</td>
<td>Digital Living Network Alliance</td>
</tr>
<tr>
<td>DMC</td>
<td>Digital Media Controller</td>
</tr>
<tr>
<td>DMP</td>
<td>Digital Media Player</td>
</tr>
<tr>
<td>DMPR</td>
<td>Digital Media Printer</td>
</tr>
<tr>
<td>DMS</td>
<td>Digital Media Server</td>
</tr>
<tr>
<td>DMR</td>
<td>Digital Media Renderer</td>
</tr>
<tr>
<td>DN</td>
<td>Distinguished Name</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial-of-Service</td>
</tr>
<tr>
<td>DSL</td>
<td>Digital Subscriber Line</td>
</tr>
<tr>
<td>DVD</td>
<td>Digital Versatile Disc (or Digital Video Disc)</td>
</tr>
<tr>
<td>eD2k</td>
<td>eDonkey Network</td>
</tr>
<tr>
<td>ERM</td>
<td>Entity Relationship Model</td>
</tr>
<tr>
<td>FLAC</td>
<td>Free Lossless Audio Codec</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HDTV</td>
<td>High-definition Television</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>ICT</td>
<td>Information and Communication Technologies</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Full Form</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPsec</td>
<td>Internet Protocol Security</td>
</tr>
<tr>
<td>IPTV</td>
<td>Internet Protocol Television</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>JIT</td>
<td>Just-In-Time</td>
</tr>
<tr>
<td>JMF</td>
<td>Java Media Framework</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>M-DMC</td>
<td>Mobile Digital Media Controller</td>
</tr>
<tr>
<td>M-DMD</td>
<td>Mobile Digital Media Downloader</td>
</tr>
<tr>
<td>M-DMP</td>
<td>Mobile Digital Media Player</td>
</tr>
<tr>
<td>M-DMS</td>
<td>Mobile Digital Media Server</td>
</tr>
<tr>
<td>M-DMU</td>
<td>Mobile Digital Media Uploader</td>
</tr>
<tr>
<td>M-NCF</td>
<td>Mobile Network Connectivity Function</td>
</tr>
<tr>
<td>MKV</td>
<td>Matroska</td>
</tr>
<tr>
<td>MIME</td>
<td>Multipurpose Internet Mail Extensions</td>
</tr>
<tr>
<td>MIT</td>
<td>Massachusetts Institute of Technology</td>
</tr>
<tr>
<td>MIU</td>
<td>Media Interoperability Unit</td>
</tr>
<tr>
<td>MPEG TS</td>
<td>Moving Picture Experts Group Transport Stream</td>
</tr>
<tr>
<td>MS-WMSP</td>
<td>Windows Media HTTP Streaming Protocol Specification</td>
</tr>
<tr>
<td>NAS</td>
<td>Network Attached Storage</td>
</tr>
<tr>
<td>NOS</td>
<td>Network Operating System</td>
</tr>
<tr>
<td>OO</td>
<td>Object-Oriented</td>
</tr>
<tr>
<td>OODBMS</td>
<td>Object-Oriented Database Management System</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>OM</td>
<td>Object Model</td>
</tr>
<tr>
<td>ORM</td>
<td>Object-Relational-Mappers</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>PGM</td>
<td>Pragmatic General Multicast</td>
</tr>
<tr>
<td>PHP</td>
<td>Hypertext Pre-processor</td>
</tr>
<tr>
<td>POSIX</td>
<td>Portable Operating System Interface for Unix</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational Database Management System</td>
</tr>
<tr>
<td>RM</td>
<td>Relational Model</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-time Transport Protocol</td>
</tr>
<tr>
<td>RTSP</td>
<td>Real Time Streaming Protocol</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Internet Protocol Suite</td>
</tr>
<tr>
<td>TDM</td>
<td>Time Division Multiplexing</td>
</tr>
<tr>
<td>TTL</td>
<td>Time-To-Live</td>
</tr>
<tr>
<td>TV</td>
<td>Television</td>
</tr>
<tr>
<td>UDP</td>
<td>User Diagram Protocol</td>
</tr>
<tr>
<td>UPnP A/V</td>
<td>Universal Plug and Play Audio/Video</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>VHS</td>
<td>Video Home System</td>
</tr>
<tr>
<td>VLC</td>
<td>Video LAN Client</td>
</tr>
<tr>
<td>Wi-Fi</td>
<td>Wireless Fidelity</td>
</tr>
<tr>
<td>WMP</td>
<td>Windows Media Player</td>
</tr>
<tr>
<td>WMV</td>
<td>Windows Media Video</td>
</tr>
<tr>
<td>WORA</td>
<td>Write Once Run Anywhere</td>
</tr>
<tr>
<td>XBMC</td>
<td>Xbox Media Centre</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XUL</td>
<td>eXtended Unified Language</td>
</tr>
<tr>
<td>XVT</td>
<td>eXtensible Virtual Toolkit</td>
</tr>
</tbody>
</table>
A novel approach to local multimedia sharing
1 Introduction

Media, as a conveyer of different forms of information, has always been a very important part of people’s lives for as long as we can remember. In the beginning it was more like a solid object such as books, newspapers or silver film based storage media. However, with the evolution of the human kind, also the media has evolved.

The content of media nowadays can be described as movies, music and pictures. Some decades ago, and even at this moment of writing, the media is mostly stored on CD’s, DVD’s, VHS’s, books, etc. As you can see, these are solid objects that you can exchange between people. Nowadays though, in most cases, the media is stored on a computer or laptop.

The problem that comes to mind now is the sharing possibility of this media. If you are not working with a centralised storage location, sharing multimedia nowadays will not be an easy task. In decentralised storage cases, we want to be able to share the media in the local network in a very easy way.

1.1 Application area and motivation

Currently, sharing decentralised media in a local network is a big problem. Every client in the network should be able to search for all the shared media from all other clients or hosts. In this way there is no need for an Internet connection nor exchanging media with others by using external hard disks or USB drives.

Computer hosts need an easy to use application, which will allow them to exchange media of their interests in a fast and easy way.

1.2 Problem studied

Because of the lack of a solution to the local media sharing possibilities, this thesis will describe the possible state-of-the-art solutions and existing applications to local multimedia sharing. To resolve this main problem, we will come across some sub problems as described below:

- Discovering and administration of the other participants
- Indexing media files
- Storing and maintaining the indexed media file lists
- Structure of the network
- Streaming the media
- Choice of the programing language
1.3 Approach chosen to solve the problem

Always a good idea is to research which state-of-the-art technologies already exist before trying to implement or create a solution yourself. This is exactly what we did, but moreover, like most people forget, we already started thinking from the user’s expectations too. Doing such a thing was very helpful as we came across sub problems we would never have thought about before. Flexibility, stability, speed and the most important requirement user-friendliness, are good examples of such user expectations.

For example, based on the user requirements mentioned above together with the research of state-of-the-art technologies, we could conclude that a client-only network structure was the best solution for providing the network flexibility and moreover reducing the needed network traffic. More information about this sub problem can be found in section 3.1.

Another good example is the choice of the programming language, described in section 3.2, to implement this project’s prototype. We had the idea to make the prototype very user-friendly. Not only because of an easy and simple graphical user interface, but also by thinking out of the box and providing cross-platform compatibility between Mac-, Windows- and Linux-users. Note that the latter operating system is not tested yet, but because of the similarities between Mac and Linux platforms, it could already work just fine.

1.4 Thesis goals and expected results

Our thesis’ goal is to provide well-founded solutions for the problems we came across in section 1.2, together with an implementation of a prototype, which will be referred to as ‘wShareLocal’ from now on and which can be adjusted in the future, to further extend the functionalities or ease of use of the application.

After the implementation process, our goal is to measure and test the prototype’s characteristics like scalability, speed, flexibility, ease of use, etc.

A comparison between the different technologies like the programming language, Directories, LDAP, SQL, a master-client network structure and the solutions used in our prototype application are interesting too. We also want to make clear why we have chosen certain technologies for certain problems and compare the results to the related research we have done.

Note that our scope stays limited to the topics mentioned above. Content copyright issues and other problems related to the multimedia author’s rights, will not be discussed. Though, a possible solution for this problem would be paying the author providing the multimedia directly, if requested, which could be implemented in the prototype application in the future using well-known PayPal technologies.
2 Background

2.1 Directories

As introduction to directories, we start with a short definition. In this case the term directory is related to a computerised database system for handling of directory information.

A directory is used for storage as a repository or database of information, which in most cases is optimised for reading performance and not so much for writing performance [1]. This is caused by the fact that the information that is stored, does not update frequently in comparison to the number of data reads. In this way the most commonly uses of a directory supports searching and browsing through the data on top of simple data lookups.

Some example usages of directories are web directories, network information services or hierarchical object oriented directories. The web directories allow you to access a categorised listing of other websites. Network information services, on the other hand, are used to provide a structured and named solution in a network. Domain Name System, DNS, is an example of this. The latter described usage of directories is the most important one in this thesis. It relates to the object oriented storage of information. Good examples are LDAP, X.500 and Microsoft Active Directory [2].

2.1.1 Directory services

The software system that makes it possible to store, update and read directories is called a directory service. It takes care of the organization of the stored data in the directories that it manages and provides interaction between the directory and the requesting parties.

To better understand the principle of directories, it is easier comparing it with something well known. You can by example compare a directory with a dictionary. When you search for a word in a dictionary, you get more information about this word. This is exactly what a directory stands for. It stores information about a lot of objects, so when you search for a certain entry, you get pieces of information about this subject. Actually directories help us to eliminate aimless searching for the information we seek.

Directories may have a very specific scope, being very specialised in one certain area, thus supporting only a small set of node and/or data types. Though they can be very broad too, supporting a lot of different types of data. In DNS by example, the nodes are domain names and the data items are IP addresses, thus a very narrow scope. With a Network Operating System, on the contrary, the nodes represent a lot of resources because the OS has to manage users, computers, printers and so on.

The simplest form of a directory service is a naming service. This service maps the names of network resources to the correct network address. When a user makes use of the name service, he does not need to remember the network address, but just needs to know the name of the network resource he wants to access. Objects that are stored in the directory server are the network resources with their name and network address as attributes. Moreover the information in the objects can be secured by using permissions to access them.
The namespace of a network is defined by a directory service. A namespace in this context can be interpreted as the term that is used to hold one or more objects as named entries. The naming and identifying of the network resources is determined by a set of rules. These rules are called the directory design process. A very important rule specifies that the names must be unique and unambiguous. In LDAP the name is called the distinguished name, DN, and is used to refer to a collection of attributes and relative distinguished names, which create the name of a directory entry.

Furthermore, it is important to realise that a directory service is a shared information infrastructure. This structure is used for locating, managing, administering, and organizing common items and network resources, which can include a broad variety of objects, like folders, files, users, groups, devices and etc. A directory service is an important component of a Network Operating System, NOS, such as Microsoft Windows or Novell NetWare. In the more complex cases, a directory service is the central information repository for a Service Delivery Platform. For example, looking up computers using a directory service might yield a list of available computers and information for accessing them.

2.1.2 Hierarchical database model

An important characteristic of directories is the fact that it is hierarchical. The directory concept is originated from an old database model, called the hierarchical database model [3]. As the name implies this database structure is crucial. The database model resembles a tree, thanks to the parent-child relationships. Every parent has one or more children, but a child has only one parent. In this way data is stored in entities and every entity has attributes, which carry the information about that entity. The data is stored in a one-to-many relationship and not in a table, like being used in regular database models.

2.1.3 Directory objects

After speaking about the structure, we can also discuss what is contained by the structure. The structure of directories is a very important item to understand, because this is what defines how suitable directories are for the problem you are facing. To make it easy to keep track of the different parts of directories, we will sum them up:

- **Entry**
  An entry is the most essential item of a directory. Since a directory is made up by different entries, this is the basic element of a directory. It is up to you to choose what you want to store, an entry can be all possible things, and there are no limitations.

- **Attribute**
  An entry on its turn consists of attributes. In this way it is possible for an entry to contain information, so when you search for an entry, you are able to see of what this entry consists. Attributes can be everything you want and can contain every kind of information you want. Moreover, it is possible to add as much attributes as you need to provide the entry with required information.

- **objectclass**
  This is a special case of attribute. The objectclass specifies what set of rules the entry needs to follow. With this, you can conclude that every entry must be made of an objectclass attribute, aside from the other attributes.
• **Container**

Beside the items to hold information in a directory, a directory also exists of objects that create the hierarchy of the directory structure itself. For these goals a directory uses containers. In this way you are able to divide entries in different containers and separate them from each other. With containers the structure becomes more clearly and hierarchical, which is one of the biggest advantages of directories. This makes the structure more readable and easier to maintain.

• **Schema**

After using containers, your entries are divided into different compartments. But to specify from what your directory data must be made up, we use a schema. A schema defines a set of rules that the contents of a directory must constrain.

Since directory services are based upon a client and server structure, you need to take into account replication and distribution. They have very distinct meanings in the design and management of a directory service. Replication is used to indicate that the same directory namespace is copied to another directory server, so you have an exact copy of it. This approach is used for redundancy and throughput reasons. The same authority maintains the replicated namespace. The term distribution can be explained by the fact that it is possible to have multiple directory servers that hold different namespaces. The different servers need to be aware of each other and of the other namespaces, so they are interconnected to form a distributed directory service. In this case, different authorities can maintain each namespace.

With the basics of directories in mind, you can understand that this system has some very important advantages when work needs to be combined. In this way you have a centralised directory that stores all the information about employees, for example. Many applications and services can take advantage of this system. The only problem with directories is that it is an unknown territory for many people. If a centralised data storage service is needed, everybody immediately thinks about setting up a Relation Database Management System, RDMS [4]. In many cases this will be a good solution, though in a lot of cases it will even be better to implement a directory. Like mentioned above a directory is optimised for reading performance and is very useful in situations where a hierarchical solution is needed.

2.1.4 **Comparison with databases**

To keep mistakes out of the picture and to show the difference between databases and directories, a comparison between the two state-of-the-art technologies will be done. In this way you can also notice that in many cases it is more suitable to use directories, an approach that is usually overlooked, like mentioned above.

In general there are two differences between the two technologies:

• Many reads, only few writes/updates
  
  Data stored in directories is optimised for reading, so there must be a high amount of read operations in comparison to writes or updates.

• Hierarchical
  
  The structure is not applicable in many cases, so mainly databases are the right choice for a centralised information storage solution.
Database solutions do not work with object classes, they work with tables to distinguish different records (or entities in directories), and so it is interesting to know what the difference with an object class is:

- An object class defines the rules the entries need to follow. In a table, the columns define how the records need to be structured.
- It is possible to have an empty attribute, so you can avoid completing every attribute, which is comparable to NULL values in databases.
- Attributes can be multi-valued. In this way it allows multiple naming attributes at one level, which is rather not preferred in databases. Directories, however, are allowed to have a multi-valued attribute, for example computer name, domain name and IP address.
- The naming convention with attributes and object classes are standardised throughout the industry and formally registered with the IANA for their object ID. A lot of software companies build their applications with this knowledge and thanks to this they can benefit of existing software applications, thus yielding faster development.
- Directory services are often a central component in the security design of an IT system because a lot of harmful data is stored by it. Administrators have to carefully implement the access control to the directory.
- Object classes are separated into namespaces within a directory, which is quite comparable to the different databases within a database system. In every database the data stored can be different than the data stored in another database, creating data separation.

2.1.5 Conclusion

The utilization of directories is not applicable to this thesis. The research however, to find out the exact meaning of directories is significant. It is closely related to databases, a section that will follow in the next pages, has the ability to store data in a directory structure and has optimised reading performance. The downside is the framework of directories since the only way to set up a directory service is to create a server to host the information. This is not suitable for small hosts as in this thesis since this breaks the flexibility requirement.
2.2 Databases

The global description of a database, in short DB, is an organised collection of data, a similar meaning as the definition of a directory. In many cases this data is used for multiple applications and is typically in digital form. The data is mainly organised to model a certain object and an application will then produce views shown by such a collection, which can help to follow and manage these aspects. The term database also refers to the way the clients view its content since the physical and logical materialization of its data. Because of this general description, many systems' data collections comply with this definition. However, the term database usually is used with a collection of data, which is maintained by a general-purpose database management system, a DBMS. These database management systems are in most cases complex software systems that need many usage requirements, and the databases that it maintains have the same complexity.

2.2.1 Database management systems

The system that is in charge of the database is called a database management system, which is a software system that allows storing and updating the database, as well as reading information from it. Such information is retrieved from the computational results done by the DBMS on the stored data in the database. Due to the fact that the structure of such a database is very complex, it needs to be handled with its database management system. Any other attempt is very likely to result in the corruption of the database. As a global model, the concept can be described as follows: a database provides the overlaying application with data, while its respective database management system is the tool to use and maintain the database. Database management systems are sold as computer software products, the database itself, of course, has to be provided by the user. Each database management system can support a database type, which is chosen by the manufacturer and specific for that kind of database management system. A couple of popular general-purpose database management systems are Microsoft’s Access and SQL Server, Oracle’s DBMS and the open source MySQL DBMS. Each one of these database management systems supports thousands of databases all over the world. It is not possible to port databases across different database management systems, but it is possible to use standards like ODBC (Open DataBase Connectivity) or, the more popular, SQL (Structured Query Language) [3]/[4]. With these standards, database management systems can inter-operate at a certain level, while each database management system controls a database of its own database type. A state-of-the-art general-purpose database management system is designed in a way that it is able to work together with as many as possible different applications. It needs to be very versatile in this way, but it also needs to be easy to understand for developers so that they are not scared by the complexity of the system.

Another important characteristic of database management systems is response time. It needs to provide effective run-time execution to properly support a high amount of end-users in terms of performance, availability, and security. The last characteristic is heterogeneous. It is not very common, but a database can be heterogeneous in type, where each database type within is controlled by its respective supporting database management system. The combination of a database and its respective database management system is sometimes referred to as a database system or DBS.
2.2.2 Data models

To suit to a certain standard, databases are typically organised according to general data models. These models emerged in the late 1960s and became genuine standards. Some models that are worth noting are:

- Relational model, RM.
  The above mentioned database management systems utilise this model.

- Entity-relationship model, ERM.
  This model is utilised primarily to design databases.

- Object model, OM.
  The main advantage of this model is that it has more expression-power than the relational, but on the other hand is it more complicated and less utilised for common applications.

It is possible to view a database within different data models that are mapped between each other. An example of this is the mapping between the entity-relationship and relational model. This mapping example is commonly used in the database design process and supported by many database design tools. It is often contained within the database management system itself.

From the three data models noted, the last one, the object model is the most interesting for this thesis. In the next section we will have a quick reference to the approach used in this case. Later on, the complete explanation of db4o, which uses the object model, will be given.

2.2.3 Global structure

In the following paragraph an overview of the global structure will be given, starting off with the lowest layer, the database itself. Databases are namely built and maintained via database management systems. These database management systems are on its turn controlled by humans, by example a database administrator, or by software systems, by example the Oracle E-Business Suite. The creation of database management systems, then again, is done by database management system manufacturers. The manufactures also develop the tools needed for database administrators. Now we come to a very important part: the database languages.

Databases need to be built and maintained. This is done by using database languages to modify them, as well as retrieve needed information from their contents. Such languages are in fact types of programming languages or extensions of existing programming languages adjusted specifically to manipulate databases. The languages used for retrieving information are also called query languages, which is most often used to refer to those database languages. Each language is data-model-specific. Some examples are SQL, the most widely used database language, and the object-oriented programming (java with db4o). This last approach is chosen in this thesis, because of the advantages concerning flexibility and scalability.
2.2.4 Database objects

Databases are made up of different objects, these will be summed up, as been done within section 2.1.3:

- **Table**
  
  Every database is made up of different tables. With these it is possible to separate the different data sets since every table can store different unrelated data. Note that the usage of tables is only available in databases. In directories you are able to divide the different entries, but you cannot really separate them from each other.

- **Record or row**
  
  In directories we talked about entries, in databases we speak about records or rows. Every row contains information about a specific entity. Every record in a table is unique.

- **Column**
  
  Columns make it possible to store different kinds of information about a record in a table. This can be compared with attributes in directories. It is possible to create as many columns as you want.

- **Relation**
  
  Between different tables you can create relations. These relations make it possible to avoid double data and link different tables with each other. The relationships that are created exist between the columns of the involved tables. You can use values of one table in the columns of another table. In directories you do not have this possibility since you work with a hierarchical database not containing tables nor relations.

2.2.5 Conclusion

Databases are a very common solution to store, update and read data. After reading the above summary of databases, it is clear why the concept is used in so many cases. The system is very versatile and many different variants exist.

Luckily for our thesis, one of these variants was very suitable. The choice we made is based upon several requirements. The first requirement is the network structure that could only be built up by clients/peers/nodes who are equipotent. This requirement already excluded most of the database management system types. The second requirement is that our prototype needs to be very flexible. In case of a SQL DBMS this requirement fails because this approach is to demanding on performance and is not easy to set up. The last requirement that applies to this state-of-the-art technology is the connectivity over the local area network. Since databases are designed to be assigned remotely, this was an easy fit.

Like mentioned in the exposition of the data models, we have chosen for an object model. The database management system is object-oriented, and more specific: db4o. More information about this technology will be given in section 3.3.2.
2.3 Peer-to-peer networks

This state-of-the-art technology will be discussed because these kinds of networks are used to share data in a flexible manner. The principle of peer-to-peer (P2P) networks will be explained first. In this way it is easier to understand the choice for this technology.

Peer-to-peer computing or networking is a distributed application architecture that shares and divides tasks or workloads between peers. Peers can be described as participants who are equally privileged and equipotent in the application. These peers together form a peer-to-peer network.

Peers make a part of their resources, such as network bandwidth, disk storage or processing power, available to other network participants. In this way you do not need central coordination by servers or stable hosts. Peers both supply and consume resources, in contrast to the traditional client–server model where only the servers supply, and clients consume.

![Image of peer-to-peer network](image)

**Figure 2.3: An example of a peer-to-peer network**

The first applications that made the peer-to-peer concept popular, are file sharing systems like Napster. The architecture has inspired new structures and philosophies in many areas of human interaction.
2.3.1 Architecture

Peer-to-peer networks can be divided into several subdivisions:

- Structured peer-to-peer networks
  - Distributed Hash Tables, (DHT)
- Unstructured peer-to-peer networks
  - Pure peer-to-peer networks
  - Hybrid peer-to-peer networks
  - Centralised peer-to-peer networks

These different network types will be explained in the following pages. As you can see, DHTs [5] will be explained along with the structured peer-to-peer networks because this is a very important and commonly used approach. For unstructured peer-to-peer networks we can subdivide the methods in three main divisions, the important difference is the way the architecture of the network is constructed.

In most cases you can assume that peer-to-peer systems implement an abstract overlay network, which is built at the Application Layer, on top of the native or physical network topology. Such overlays are used for indexing and peer discovery and make the P2P system independent from the underlying physical network topology. Data exchange is typically done directly over the underlying Internet Protocol (IP) network. Anonymous peer-to-peer systems, on the other hand, are an exception. They implement extra routing layers to hide the identity of the source or destination of transmitted queries.

In structured peer-to-peer networks, peers (or resources) are organised to follow specific criteria and algorithms, which cause overlays with specific topologies and properties. They typically use distributed hash table-based indexing.

Figure 2.3.1: Decentralised peer-to-peer concept (left), Centralised peer-to-peer concept (right)
In contrast to structured peer-to-peer networks, we have the unstructured peer-to-peer networks. They do not provide any algorithm to organise or to optimise the network connections. Unstructured peer-to-peer networks can be mainly divided into three architecture models. First we have pure peer-to-peer systems. Here, the entire network consists of equipotent peers and there is only one routing layer because there are no preferred nodes with a special infrastructure function. Secondly we have hybrid peer-to-peer systems, which allow the existence of infrastructure nodes with special functions. These special nodes are often called super-nodes. As the latter unstructured model, we have centralised peer-to-peer systems. Here a central server is used for indexing functions and to bootstrap the entire system. Although this architecture has similarities with the structured one, the connections between peers are not determined by any algorithm. The first prominent and popular peer-to-peer file sharing system, Napster, was an example of this latter peer-to-peer model [6]. In Napster, the regular nodes make use of a central server to find data from other nodes. Gnutella and Freenet, on the other hand, are examples of the decentralised model, these systems do not use a central server, only nodes or super-nodes exist. KaZaa is in between the model used by Napster and the one used by Gnutella, they use the hybrid model.

Peer-to-peer networks are typically used to connect nodes via ad hoc connections. Data, including digital formats like video files and real time data like telephony traffic, is passed by using peer-to-peer technology.

To understand the concept of nodes and to be sure there will be no mistakes when you compare this system with the client-server system you will be given a short explanation about this difference. A pure peer-to-peer network, by example, does not make the distinction of clients or servers but it only has the notion of equal peer nodes that simultaneously function as both clients and servers to the other nodes on the network. In some cases a node will act as a server to another node that in this case will act as a client, for by example sharing an audio file. A model that arranges its network like a pure peer-to-peer network differs from the client–server model, where communication usually only exists from and to a central server. A typical example of a server-client communication, that does not use the peer-to-peer model, is the Hypertext Transfer Protocol, HTTP, service in which the client and server programs are distinct. The clients request information from the server, and the servers respond to these requests, so that the clients will get what they requested from the server, it will never be the other way around.

The peer-to-peer overlay network consists of the participating peers, which will act as network nodes. Links exist between nodes that know each other. By example: if a participating peer knows the location of another peer in the peer-to-peer network, then there is a directed edge from this one to the other node in the overlay network. We can classify the peer-to-peer networks as unstructured or structured based on how the nodes in the overlay network are linked to each other.
2.3.2 Structured systems

Now we go more into detail to the different peer-to-peer networks. Structured peer-to-peer networks generally utilise a consistent protocol. In this way the network is able to ensure that any node can efficiently route a request to a peer that has the desired data, even if the file is extremely rare. This guarantee is not applicable with unstructured systems, but we will go more into detail with that in the section of unstructured systems. To be able to fulfil this guarantee, it is necessary to have a more structured pattern of overlay links. By far, the most common type of structured peer-to-peer networks is the distributed hash table, in which a variant of consistent hashing is used to do the assignment of the ownership of each file to a certain peer. This is done in a way that resembles the traditional hash table's assignment of each key to a particular array slot.

First it is very crucial to understand that DHTs are used for decentralised systems, so without using a centralised server of some kind. They can be interpreted as a class of distributed and decentralised systems that provide the possibility to lookup data similar to a hash table. Pairs of (key, value) are stored in the DHT, and any participating node is able to efficiently retrieve the value associated with a given key. The responsibility for maintaining the mapping from keys to values is distributed among the nodes. With this concept a change in the set of participants, when by example a participant disconnects, causes a minimal amount of disruption. This benefits the scalability of DHTs. The network can range from a small amount to extremely large numbers of nodes because DHTs are able to handle the frequently adding, removing and disconnecting of nodes.

The infrastructure formed by DHTs can be used to build peer-to-peer networks. Important and probably well-known distributed networks that use DHTs are BitTorrent's distributed tracker, the Storm botnet and the Coral Content Distribution Network [5][6]. Because of the importance and versatility of DHTs, there are also some prominent research projects going on, like the Chord project at MIT.

![Distributed Hash Tables Example](image)

**Figure 2.3.2: Distributed hash tables example**

DHT-based networks [5] have been widely utilised for accomplishing efficient resource discovery of computer resources from multiple administrative domains. These domains can then easily work together to reach a common goal. This kind of collaboration is called grid-computing systems. DHTs are a good use in these systems, as it aids in resource management and the scheduling of applications. The activity to discover resources involves the search for the appropriate resource types that match the user’s application requirements. With the recent adjustments in the domain of decentralised resource discovery, the existing DHTs are extended with the capability to organise data with multiple dimensions and even to route queries.
2.3.3 Unstructured systems

The second architecture subdivision of peer-to-peer networks is unstructured peer-to-peer networks. The main characteristic of these networks is that the overlay links are established arbitrarily. Such networks can be easily constructed. When a new peer wants to join, by example, the network can copy existing links of another node for the new node and then over time it can form its own links. The main disadvantages are the amount of traffic an unstructured network produces and it cannot guarantee that the query will be resolved. Example: if a peer wants to find a desired piece of data in the network, the query has to be flooded through the whole network to find as many peers as possible that share the data. If in the case the desired data is very rare, the query will have a very low success rating. Popular content is likely to be found at several peers and any peer searching for this content will have the same search results. Since there is no connection between a peer and the content it manages, there is no guarantee that flooding the network will find a peer, which is holding the desired data. Like mentioned above, flooding also causes a high amount of signaling traffic in the network and as a result such networks typically have very poor search efficiency. Many of the popular peer-to-peer networks are unstructured, because they are more flexible.

Unstructured peer-to-peer networks can still be divided into more specific architecture methods and this based upon the roles of the peers in the network:

- Pure peer-to-peer networks
  In pure P2P networks, peers are all equal. They merge the roles of clients and server, acting as a server when requests are received and as a client when sending requests. In such networks, there is no central server managing the network, neither is there a central router. An example of pure peer-to-peer Application Layer networks designed for peer-to-peer file sharing is Freenet.

- Hybrid peer-to-peer networks
  There also exist hybrid peer-to-peer systems, which distribute their clients into two groups: client nodes and overlay nodes (also sometimes called super-nodes). Typically, each client is able to act according to the demands of the network at that moment and can become part of the respective overlay network used to coordinate the whole peer-to-peer structure. The separation between the two different nodes is done to address the scaling problems that occurred in the early pure peer-to-peer networks. The ‘better’ nodes are used to distribute the traffic in a more structured and scalable way. An example of such networks is Gnutella (after v0.4).

- Centralised peer-to-peer networks
  The last method is another type of hybrid peer-to-peer networks and is called centralised peer-to-peer networks. These networks use central servers or bootstrapping mechanisms together with peer-to-peer for their data transfers. We use the term ‘centralised’ due to the fact that they can only work with their central servers. An example for such a network is the eDonkey network (eD2k).
2.3.4 Indexing and resource discovery

In the old days, peer-to-peer networks duplicated the resources across each node in the network, which were configured to carry that type of information. In this case the main advantage is that it allows local searching and it is much faster, but the downside is that it requires much more network traffic.

Nowadays, peer-to-peer networks use central coordinating servers and directed search requests, so you do not flood the network with queries. The typical use of the central servers are listing potential peers (by example Torrents), coordinating their activities (by example Folding@home), and searching for files (by example Napster). First peer-to-peer networks started off with flooding the queries across the participating peers for decentralised searching. Nowadays, they use more efficient and directed search methods, including DHT and supernodes.

To give you an idea how peer-to-peer networks are implemented at the moment of writing, we can conclude that in many cases, peer-to-peer systems use more powerful peers (super-nodes) as servers and client-peers are connected in a star-like topology to a single super-node.

2.3.5 Advantages and disadvantages

Clients provide resources in a peer-to-peer network. These clients may include storage space, computing power and bandwidth. So as more nodes arrive, the load on the system increases, but also the total capacity of the system increases. In contrast with a typical client–server architecture, clients only cause extra load on the system and do not share their resources. In this case, when more clients join the system, fewer server resources are available to serve each client.

For pure peer-to-peer systems it is not only the capacity that increases, but also the robustness of the network. This is caused by the distributed nature of peer-to-peer networks that also increases robustness, by enabling peers to find the data without relying on a centralised index server. In this case, there is no single point of failure in the system, so you do not need to worry about redundancy.

Security of peer-to-peer networks needs to be taken into account too. In case of most network systems, unsecure and unsigned codes may expose unwanted security threats, like remote access to files on a victim's computer, which can even compromise the entire network. An example of this can be found with the FastTrack network (used by KaZaa). Anti-peer-to-peer companies managed to introduce faked chunks into downloads and downloaded files. Mostly MP3 files became unusable afterwards or even contained malicious code. However, this happened already some time ago. Consequently, the peer-to-peer networks of today have enormously increased their security and they support file verification mechanisms. Most networks are nowadays resistant to almost any type of attack, thanks to chunk verification, modern hashing and different encryption methods, even when major parts of the respective network have been replaced by fake or nonfunctional hosts.
2.3.6 Conclusion

The peer-to-peer framework looks very promising because of the discovery of peers and the maintaining of the connections between them. The big interest area is pure peer-to-peer networking, because this is the most flexible solution in our case. We want to build up a network structure with all peers, who are able to fulfill the role of a client as well a server. There is no need for a server, because in this case every peer maintains its own data and when another peer requests data, they will respond to the request if the requested data is available. No server is needed and so no failover or redundancy must be taken into account. The big problem that we came across in pure peer-to-peer networks is that there is no guarantee that the request for information can be answered. If we then take a look at the structured network architectures, we sadly enough also find a big downside. In these peer-to-peer networks there is need for an online Internet connection and one of our thesis’ requirements is the ability to function without any Internet connection.
2.4 Orb

2.4.1 Introduction

After some research we found an existing application that also implements media sharing across a network. The application is called Orb [7] and comes close to what we want to realise in our project, but has quite some drawbacks as you will see discussed later.

When Orb is installed on your pc you are able to:

- Stream music to you notebook, smartphone or to the best speakers in the house
- Stream movies to your laptop or TV
- Stream live and recorded TV shows on your laptop or smartphone
- Stream photos on any remote computer
- Share your media (music, videos, photos and documents) through a free Orb account

Note that for all services to work properly, a few requirements have to be met, which will be discussed in the section below.

2.4.2 Application requirements

The application’s functionalities are very promising, but to be able to use all this useful functionalities a few important requirements have to be met:

- You need an ‘always on’ home computer where all your media has to be centralised in order to share your resources.
- You need a very reliable and fast internet connection.
- You need to create a free account at Orb’s servers to be able to access your media from another computer or device.

In the next section this application’s requirements will be compared to the prototype’s requirements we have in mind. It will be a good overview of the pros and cons of both applications.
2.4.3 Advantages and disadvantages

When comparing Orb with our application’s structure and goals, there are some advantages and disadvantages about both:

<table>
<thead>
<tr>
<th></th>
<th>Orb</th>
<th>wShareLocal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network structure</td>
<td>via the internet via Orb’s servers, so media availability all over</td>
<td>via the local network so media is only available within</td>
</tr>
<tr>
<td></td>
<td>the world.</td>
<td>the local network area.</td>
</tr>
<tr>
<td>Media support</td>
<td>for music, videos, pictures, etc. are well implemented.</td>
<td>can be implemented by using VLC libraries, which support all</td>
</tr>
<tr>
<td></td>
<td></td>
<td>media file structures.</td>
</tr>
<tr>
<td>Free application</td>
<td></td>
<td>Free application</td>
</tr>
<tr>
<td>Public sharing</td>
<td>via a flash widget is present.</td>
<td>Public sharing is implemented by default.</td>
</tr>
<tr>
<td>Cross-platform</td>
<td>compatibility is provided via separate versions for each operating</td>
<td>compatibility is provided by using the Java Virtual Machine,</td>
</tr>
<tr>
<td></td>
<td>system.</td>
<td>one version is sufficient to provide cross-platform</td>
</tr>
<tr>
<td></td>
<td></td>
<td>compatibility.</td>
</tr>
<tr>
<td>An ‘always on’</td>
<td>internet connection is required since all data has to be send to an</td>
<td>No internet connection required, only a local network</td>
</tr>
<tr>
<td></td>
<td>Orb master server, which clients in the same network will connect</td>
<td>connection to share between different users.</td>
</tr>
<tr>
<td></td>
<td>to.</td>
<td></td>
</tr>
<tr>
<td>The connection between</td>
<td>client and Orb’s master server has to have a high throughput, a low</td>
<td>Local network connections are less prone to high latencies</td>
</tr>
<tr>
<td></td>
<td>latency and a high reliability, which can be expensive.</td>
<td>or low throughput issues.</td>
</tr>
<tr>
<td>Problems at Orb’s</td>
<td>No master server required, all media is decentralised on all clients,</td>
<td>All media is centralised so if the server computer fails, all</td>
</tr>
<tr>
<td>master server will</td>
<td>a problem only results in unavailability of certain shares of one</td>
<td>shares are inaccessible.</td>
</tr>
<tr>
<td>result in a loss of</td>
<td>client.</td>
<td></td>
</tr>
<tr>
<td>all functionality</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>because all traffic has to pass through it for all Orb’s clients.</td>
<td></td>
</tr>
<tr>
<td>An ‘always on’ home</td>
<td>An ‘always on’ home computer is not needed, all media is shared</td>
<td>An ‘always on’ home computer is required, which makes</td>
</tr>
<tr>
<td>computer has to be</td>
<td>automatically when clients enter the network.</td>
<td>sharing less user-friendly.</td>
</tr>
<tr>
<td>present, which will</td>
<td></td>
<td></td>
</tr>
<tr>
<td>keep using power even</td>
<td></td>
<td></td>
</tr>
<tr>
<td>when clients are not</td>
<td></td>
<td></td>
</tr>
<tr>
<td>accessing the shares.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All media is</td>
<td>Media is decentralised over all clients, which yields smaller</td>
<td>Only the main application is required, required media</td>
</tr>
<tr>
<td>centralised so if the</td>
<td>footprints when having problems.</td>
<td>players are included in the distribution prototype.</td>
</tr>
<tr>
<td>server computer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fails, all shares are</td>
<td>Media is decentralised, no need for ordering or sorting it, the</td>
<td></td>
</tr>
<tr>
<td>inaccessible.</td>
<td>application automatically detects media files and shares them.</td>
<td></td>
</tr>
<tr>
<td>Media has to be</td>
<td></td>
<td></td>
</tr>
<tr>
<td>centralised first,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>which can take a lot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>of time and effort.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A device with a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>browser or a streaming</td>
<td></td>
<td></td>
</tr>
<tr>
<td>media player is</td>
<td></td>
<td></td>
</tr>
<tr>
<td>required.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creation of an Orb</td>
<td>Plug and play policy, no accounts required.</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.4.3: Advantages and disadvantages of Orb compared to wShareLocal
2.4.4 Conclusion

As stated in the previous section, Orb still has quite some drawbacks that can easily be solved or implemented in our prototype, or another application. Orb has a big advantage too though: media is available all over the world if everything is working as planned. However, this does not weigh up against all other advantages wShareLocal could offer while Orb cannot. The conclusion is clear and simple: Orb is not a good solution for solving our big research question.
2.5 Windows Media Sharing

2.5.1 Introduction

Since many people nowadays have a lot of music, video, and picture files in Windows Media Player’s library, there might be times when you want to play media on another computer, Xbox 360 or a networked digital media player in your home. By using Windows Media Player and a private network, you are able to stream your favourite media to anywhere in your home, even to rooms that do not have a computer. Even when people log in to your computer with their own user accounts, they are able to play all media files in Windows Media Player’s library.

By sharing your media with Windows Media Sharing [8], you can do all these things, however not outside of your private network. For example, it is also not possible to use media sharing to download music from the Internet or to stream video on a website.

2.5.2 Application requirements

Windows Media Sharing’s functionalities are very promising again, but to be able to use all this useful services a few important requirements have to be met:

- You need a wired or wireless private network.

- You need a device known as a networked digital media player, a digital media receiver, or another computer running the same version of Windows Media Player. Networked digital media players are hardware network devices that connect to your private network and allow you to browse and play content from your Windows Media Player’s library, even if you stored the media on a computer in another room.

- You need to configure media sharing, which is not enabled by default.
• Several firewall ports on your computer must be open, however if you are using Windows Firewall, Windows’ default firewall application, turning on media sharing will automatically open the required ports for sharing your media.

• The IPsec (Internet Protocol Security) service has to be disabled.

2.5.3 DLNA

Windows Media Sharing makes use of DLNA’s, the Digital Living Network Alliance’s, upcoming technology, which is a standards-based technology to make it easier for consumers to use, share and enjoy their digital photos, music and videos [9].

DLNA is a non-profit collaborative organization established by Sony in June 2003, and nowadays has even more than 250 member companies in the mobile, consumer electronics, PC and service provider industries. In January 2011, more than 9000 different devices already obtained the “DLNA Certified” status, indicated by a logo on their packaging and confirming interoperability with other DLNA devices.

In fact Windows Media Player is a perfect example of “DLNA Certified” software, which is sold directly, however in this example included in the Windows operating system, to consumers through retailers, websites and mobile application stores. Windows Media Player integrates a former non-DLNA device like a computer into the personal DLNA ecosystem, which helps bringing content such as videos, photos and music stored on DLNA Certified devices to a larger selection of consumer electronics, mobile and PC products.

There are different classes of DLNA Certified devices, which can be separated as: home network devices, mobile handheld devices and home infrastructure devices. The subdivision of the first class, the most important one related to this example, can be described as follows:

• Digital Media Server (DMS): These devices store content and make it available to networked Digital Media Players (DMPs) and Digital Media Renderers (DMRs). Examples include PCs and Network Attached Storage (NAS) devices.

• Digital Media Player (DMP): These devices find content on Digital Media Servers (DMSs) and provide playback and rendering capabilities. Examples include TVs, stereos and home theatres, even wireless monitors and game consoles.

• Digital Media Renderer (DMR): These devices play content received from a Digital Media Controller (DMC), which will find content from a Digital Media Server (DMS). Examples include TVs, audio/video receivers, video displays and remote speakers for music.

• Digital Media Controller (DMC): These devices find content on Digital Media Servers (DMSs) and play it on Digital Media Renderers (DMRs). Examples include Internet tablets, Wi-Fi enabled digital cameras and Personal Digital Assistants (PDAs).

• Digital Media Printer (DMP): These devices provide printing services to the DLNA home network. Generally, Digital Media Players (DMPs) and Digital Media Controllers (DMCs) with print capability can print to DMPs. Examples include networked photo printers and networked all-in-one printers.
In fact Windows Media Player (WMP) can be classified as a Digital Media Server because the media is stored and can be shared to other DLNA Certified devices. However the application has a double functionality as it also supports playback and rendering capabilities of media stored on other DLNA devices, which makes it classifiable as a Digital Media Player.

Other devices, as mentioned in the previous paragraph can be in the home network class, but can also be mobile handheld devices, or home infrastructure devices, which can be separated as follows:

**Mobile Handheld Devices**

- **Mobile Digital Media Server (M-DMS):** These wireless devices store content and make it available to wired/wireless networked Mobile Digital Media Players (M-DMPs), Digital Media Renderers (DMRs) and Digital Media Printers (DMPrs). Examples include mobile phones, especially smartphones, and portable music players.

- **Mobile Digital Media Player (M-DMP):** These wireless devices find and play content on a Digital Media Server (DMS) or Mobile Digital Media Server (M-DMS). Examples include mobile phones, again smartphones, and mobile media tablets designed for viewing multimedia content.

- **Mobile Digital Media Uploader (M-DMU):** These wireless devices upload content to a Digital Media Server (DMS) or Mobile Digital Media Server (M-DMS). Examples include digital cameras and smartphones.

- **Mobile Digital Media Downloader (M-DMD):** These wireless devices find and download content from a Digital Media Server (DMS) or Mobile Digital Media Server (M-DMS). Examples include portable music players and smartphones.

- **Mobile Digital Media Controller (M-DMC):** These wireless devices find content on a Digital Media Server (DMS) or Mobile Digital Media Server (M-DMS) and send it to Digital Media Renderers (DMRs). Examples include Personal Digital Assistants (PDAs) and smartphones.

**Home Infrastructure Devices**

- **Mobile Network Connectivity Function (M-NCF):** These devices provide a bridge between mobile handheld device network connectivity and home network connectivity.

- **Media Interoperability Unit (MIU):** These devices provide content transformation between required media formats for home network and mobile handheld devices.

To summarise: DLNA is a very good upcoming technology. In practise you are able to choose media on your DLNA Certified device and decide on which other DLNA device you want to play or watch it when the file structure is supported. With Windows Media Sharing, the user is also able to login with its own user account to your DLNA device to browse through the media you shared. However quite some requirements have to be met in order to make use of this functionality, as described before, together with creating the necessary user accounts on the device where the media needs to be accessible. Note that this process is not always user-friendly.
In the next section all advantages and disadvantages of Windows Media Sharing, in fact all DLNA Certified applications, will be compared to the goals of our solution, as been done before in the section describing Orb.

2.5.4 Advantages and disadvantages

As mentioned in the previous section, Windows Media Player, or Windows Media Sharing, is just an example of an application using the DLNA technology. Comparing the goals of wShareLocal to Windows Media Player would not be a well-founded comparison because nowadays there are different clients available based on DLNA technology: Skifta for Android, the first certified DLNA software, J. River Media Centre, Windows Media Player 11 and 12, XBMC, AwoX mediaCTRL for iPhones and so on.

That’s why we have chosen to provide a better comparison and summarise all DLNA based applications with the prototype we had in mind.

Both applications will make use of a local network structure, so media is only available within the local network area and cannot be accessed from the outside. This is quite a drawback for both applications.

When looking at the media file structure support of both applications, we were very surprised of the huge limitations of the DLNA technology. All over the Internet, DLNA is found to be an excellent solution for sharing your media between a lot of compatible devices like mobile phones, TVs, the Xbox, the PlayStation 3, etc. However in practise, nowadays the sharing possibilities are very limited since DLNA can also be defined as the former UPnP A/V, but now with restrictions.

The UPnP A/V is an UPnP specification that is used by many audio- and video players or servers to provide an easy way to share and stream multimedia content, but sadly was never really well defined. It uses broadcast messages, making massive usage of SOAP requests over HTTP to send info in XML form and it requires up to 3 services to dialog between a client or player and a server only to make the server provide an HTTP URL to the client to know where to stream from.

First the DLNA technology used the UPnP A/V protocol but, judging it was way to versatile, it was better to harden it quite a bit. Using plain UPnP A/V, a server can share any kind of file from binary files to audio- and video files. Even text files, Word or PowerPoint presentations could be shared. All it does after all is sharing a file through HTTP with its associated MIME type, just like any HTTP server like Apache would do. The client or player simply had to determine whether or not it could handle the stream.

But the Digital Living Network Alliance decided that only a numerous number of files are worth being shared. As a result, they restricted the protocol so that only some specific file extensions, but also containers and audio/video codecs combinations can be used.
Below you can find the only supported stream characteristics, as you can find on the official website of DLNA:

<table>
<thead>
<tr>
<th>Media Formats</th>
<th>Required Formats Set</th>
<th>Optional Formats Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imaging</td>
<td>JPEG</td>
<td>GIF, TIFF, PNG</td>
</tr>
<tr>
<td>Audio</td>
<td>LPCM (2 channel)</td>
<td>MP3, WMA9, AC-3, AAC, ATRAC3plus</td>
</tr>
<tr>
<td>Video</td>
<td>MPEG2</td>
<td>MPEG1, MPEG4, WMV9</td>
</tr>
</tbody>
</table>

Table 1. DLNA Media formats for Home Devices

Table 2. DLNA Media formats for Mobile/Handheld Devices

Figure 2.6.4 a: DLNA supported stream characteristics

As you can see, many media formats are purely dropped because no one uses them in the industry, which is quite simple to understand. However no free audio codecs like Ogg/Vorbis or FLAC are supported, nor is Windows Media Video 10, also known as VC-1, which tends to replace the WMV9 codec more and more. More interestingly, neither AVI nor Matroska (MKV) containers are supported, which just means that 99% of video files shared over the Internet are not DLNA compliant, and thus cannot be streamed.

Now we will discuss the tricky part. Supposing you have a multimedia file that fulfil these constraints, like a HDTV trailer downloaded from Apple.com, which is H264 video based with AAC audio encapsulated with a MOV/MP4 container. It looks like a supported media file, but unfortunately, it is not. As mentioned before the audio- and video codecs combinations are limited too, which are stored in DLNA profiles. A file, to be compliant with DLNA, has to comply with one of these profiles. As said, it depends on the audio- and video codec combination, but also on resolution, bitrates and other information.

If we use the previous example, the file could have matched because both A/V codecs and the container is valid, though, when the resolutions exceeds 720x576, the MP4 container is judged inadequate and thus only the content within the MPEG TS container can be streamed.

How to stream your file then? Well, this is quite simple. For each file, the DLNA server, most likely a DMS, has to demux the file first in order to retrieve its A/V codec and container information in order to know whether or not it complies with an existing profile. If it does not comply and you still want your media to be streamed, the server has to remux your media file, best case ever, or remux and re-encode it on-the-fly which is the worst case ever because it consumes a lot of CPU power, which is by definition, not available. Since a lot of users may be interested in streaming your media, your computer will have a very high workload, which can lead to unplayable local media content, thus has to be avoided. It also makes it impossible to implement this technology on embedded devices, which were the target of DLNA inventors.
In contrast with our prototype, which will probably make use of Java VLC libraries supporting all possible media file structures, DLNA is really flawed in supporting widely used media formats by home users, not the industry. In order to be working properly, these users should first re-encode all their media in a DLNA compliant format, which is not user-friendly at all. Having discussed the media file support, we can now look at other pros and cons.

Figure 2.6.4 b: VLC logo

An advantage of both implementations is that there is quite some DLNA Certified software available for free, such as Windows Media Player 11 or 12, which is included in the Windows operating system, Cyber Media Gate, available for Windows, Mac and Linux, Yahoo Music Jukebox, only supporting music files, etc. However, wShareLocal, the name of our prototype, was planned to be a free application too.

Again an advantage of both applications is that public sharing is possible. Though with most DLNA servers and clients, public sharing has to be configured since new clients joining the network must first be allowed by the server in order to access the media resources. On the contrary, our prototype does not need configuring public sharing as all shared directories are automatically shared along clients in the same local network.

Cross-platform compatibility is also well supported along both solutions. For DLNA clients a wide variety of cross-platform applications already exist. wShareLocal is implemented in Java and thus is also multi-platform compatible.

Both implementations also make use of local network connections, which are less prone to high latencies or low throughput issues, even if the network is provided wirelessly. Both are very flexible too since there is no need to centralise the media, nor to have a master server, which yields small footprints when having problems.

An advantage of DLNA clients over wShareLocal is that it is already widely supported, even on mobile devices and TVs, which is quite impressive, however in order to work, all participating devices have to be DLNA Certified.

Finally to conclude the advantages and disadvantages, we stumbled upon an important and useful functionality missing in almost every DLNA client. Users are able to browse the media, however they were almost never able to do a search through all resources, which really would make the application user-friendlier while adding a great functionality too. Note that wShareLocal is designed to search through all shared media resources from the start, the database structure is even optimised for it, as you can see discussed in section 3.3.

2.5.5 Conclusion

The media support of DLNA Certified software is a major drawback, which is tried to be solved by transcoding, on-the-fly converting a potentially unsupported encoding into a supported one for the receiving device. Moreover the lack of search functionality is really annoying and not user-friendly at all. However, DLNA is already widely implemented and working pretty well if the media file structure is supported. It is a quite good standard considering the annoying drawbacks, but we tried to implement a solution with the least number of disadvantages. That is why we decided to continue our search for a better solution and leave DLNA’s technologies for what it is.
A novel approach to local multimedia sharing
3 Solution Idea to be investigated

3.1 Choosing the right network structure

3.1.1 Introduction

The network structure of the prototype or the final application is an important choice. It needs to be researched thoroughly because it has a big impact on the scalability, flexibility and speed of the application itself. Users, most of the time, do not have to know or do not care which structure is used. They are only interested in having a solid network base to share the multimedia between different clients.

Figure 3.1.1: Network structure

We started off to research state-of-the-art network structures as used in peer-to-peer applications previously discussed in section 2.3. Afterwards we started looking at the master-client based network structures, as you will see discussed below.

3.1.2 Master-client based network

Since most peer-to-peer based applications use a master-client based network structure over the Internet, we investigated this approach first.

A master-role in this kind of network structure is to hold or store all information where clients are interested in. All clients participating in this particular network generally provide this information, which can be indexed lists of media, to the master. The client providing information to a master can also be called the client’s-role.

Masters have to be quite powerful computer systems with a stable connection to all other clients. High throughput and low latencies are requirements too since all clients request for information to the master, which has to be able to respond quickly at all times. Not only quickly responding to requests, but also executing commands the clients request for, such as the processing of data or encoding a movie file, has to be completed very fast.

A master-client network structure would look like this:

Figure 3.1.2 a: Master-client network structure
As you can see, the master holds all information in his database, while the client is requesting for information. The master will have to process the request and send back the appropriate information as displayed in the left figure below. Note that in practise probably more clients are connected to a master server, which can all send requests at the same time. A good example for this is a user, serving the client’s role, doing a search request to Google.com, being the network master. The figures are simplified and only display one client to ease the explanation.

When the client acquires the master’s information package, like a search result, it is able to do another request to the same server, which is exactly what will happen when trying to stream a media file, related to our research question, as shown in the figure on the right shown above.

However to keep in mind flexibility, scalability and speed, which are the important requirements for our prototype’s network structure, the choice of the master computer is very important. A possible use case will be discussed in the next paragraph.

When our prototype application would be launched, it could search on the local network whether other clients are already present, which requires a network broadcast. This broadcast message will contain a request to all active clients at that time. However the master is the only one obliged to acknowledge the request by replying to it. Either a smart master selection algorithm can be implemented, or not. If no active master in the current network is found, the new client will automatically become the local network master.

Since a new client broadcasts its presence or an existing master acknowledges a request, it can also provide information about its capabilities. Such information can be exchanged using the results of speed tests, the availability of free space, latency test results and others. When a new client receives this information package, does the same tests meanwhile and gets better results, a smart algorithm can conclude that the new client may better fulfil the master-role in the network structure.

The first scenario is that the new client is indeed faster than the current network master. A transfer between the client and master will then have to take place. This will not be a network-friendly task, especially with high numbers of active clients. First the former master will have to notify all clients that a new master will be assigned, note that this can cause a lot of network traffic, and afterwards the bulky database of the master has to be transferred over the network. Note that when doing so, requests to the master cannot be replied to anymore, which is the worst case, or they will have a much higher delay because of the higher network load during the transfer.
In a network with a lot of clients coming and going, this smart algorithm might cause a lot of overhead because of the need to switch the master-role between clients, however in a rather static network, this algorithm can be advantageous too since it can provide more responsiveness and speed to the network structure.

The second scenario is that the new client is slower than the computer fulfilling the current master-role. In this case the new client will only reply to the master by sending his indexed media file list. Search requests, even for local files, will always have to be sent to the master server since clients do not store a database their selves.

But what happens if the clients update their indexed media file lists? Each time they will have to notify and resend the new list to the master server, which on his turn has to notify all clients that the shared media has changed. Users have the possibility to still get search results, even when meanwhile a client changed his shared media files, because the transfer and notifications of updated lists can take a while in order to be fully processed or because the master has implemented a search cache.

At this point of research we stumbled upon some important sub questions: What is the best way to store the indexed file lists as a master? Into databases, directories or object-oriented database structures? But even more important: Why should the indexed lists be centralised if the multimedia itself is decentralised? What if the master decides to stop working? What about failover standby master computers and how much should there be? Is there not a way to reduce network traffic? Do we really need a master-client structure? Is this approach scalable and flexible for new users joining the network?

The answer of the first question can be found in section 3.3, but the answer on the more important question is rather a question again: Is it not better to decentralise the indexed media lists too? This approach will be discussed in the following section.

Answering the question what would happen if a master decides to stop is easier. When closing the prototype, the master would have to notify all other clients that he is about to leave the network, which will require network traffic. After this, a client has to take over the master-role and get a copy of the master’s database, which as mentioned before can have negative consequences. It would be better to have a client acting as a standby or failover master in the network, which will automatically take over when the ‘main’ master decides to stop, but how many standby masters would be needed then? Also note that standby masters have to be updated with recent indexed lists too. Having 2 masters, an active- and a failover master, will result in twice the needed network traffic, which is not efficient. To solve the question of how many standby masters should be present, well… this question cannot be answered correctly since masters are able to stop at the same time too, thus all clients in the local network should be failover masters to have a stable structure. Obviously this causes massive network traffic, overhead, and is not a good solution at all.
3.1.3 Clients-only based network

A client-only network is simply a network without any master-role. Related to our project, this approach can greatly reduce the network overhead needed to assign the master-role or doing search requests. Even no failover masters are required, and clients can join or leave the network whenever they want, which will be discussed later. Note that clients in this kind of structure need to store their own indexed lists, which provides faster search results since search queries can now be executed locally. This will also be discusses later on.

When a new client joins the network, here a wireless network, it broadcasts its presence so that other clients know a new client joined the network.

Afterwards the existing clients will have to send their indexed file lists to the new client joining the network, which can be a unicast since they learned the client’s IP address from the broadcast of presence package send before.

Figure 3.1.3 a: A client broadcasting its presence

After this, the new client sends his indexed file list to all other clients, being a multicast. Now the process of a new client joining the network is complete and the new client is able to search for a specific file in his own database, which now contains data from all clients sharing media in the network. From this information, it will get the location of the media file requested for. Finally a stream between the two clients can be set up to easily make use of the media while not needing to download or wait for it.

Note that this structure was intentionally the idea for our prototype. However, as you will see discussed later, it changed a little bit.

Figure 3.1.3 b: Different steps of a client joining the client-only network
3.1.4 Conclusion

As described above, a master-client network structure would be very useful in rather static networks, especially in combination with a smart master selection algorithm. However, it strongly limits the flexibility and scalability since one master has to be able to handle all requests and streams to every client. A master-client structure also requires a lot of network traffic for the necessary handshakes between the participating users, called overhead.

A client-only network structure however, provides the same services while reducing the network traffic and distributing the workload along all participating clients, even search results will be faster. In general this approach is more preferable as a choice for the network structure.
3.2 Choosing the programming language

3.2.1 Introduction

Choosing a programming language for the whole project is an important choice. It determines which platforms are supported, the ease of implementation and, most important, the available support.

Since cross-platform compatibility and support is very important nowadays, we looked into the cross-platform programming approaches. In order for software to be considered cross-platform, it must be able to function on more than one computer architecture or operating system.

Figure 3.2.1: Programming languages

Note that different operating systems have different APIs, which are sets of rules and specifications that software programs can follow to communicate with each other. They serve as an interface between applications by facilitating their interaction similar to the way the user interface facilitates interaction between humans and computers. Linux uses a different API for applications than Windows does. However there are big similarities between the Linux OS and the Mac OS APIs.

Yet there are different ways of implementing a cross-platform application, which will be discussed in the next section.

3.2.2 Approaches

There are 4 main possibilities to realise this kind of programming. First of all one can create multiple versions of the same application using different source code sets. This approach is straightforward, yet is more expensive in development costs and time. Also note that using different source code sets, which are most of the time written by different specialised programmers, would result in more problems considering bug tracking and fixing on different platforms. The smaller the programming team though, the quicker the bug fixes tend to be.
Secondly you can depend on pre-existing software that hides the differences between different platforms, called the abstraction of the platform or platform agnostic solutions. Using this approach the application itself will be unaware of the platform it is running on. The most important examples which provide such capabilities are:

- Java in combination with the Java Virtual Machine (JVM), which is a well-known and supported programming language, also making an application possible to be implemented on other devices than personal computers, such as mobile phones (smartphones) and tablets. Currently, applications written in this programming language can be executed on Microsoft Windows, Mac OS X, Linux and Solaris operating systems, even Android has built-in support for Java.

- C#, pronounced C Sharp, in combination with Mono. C# is a Microsoft .NET programming language only supported in current Windows releases, which makes it quite limited in scalability between platforms. However Mono, an open source version of the Microsoft .NET framework, eliminates this limitation. Note that when trying to execute C# applications on other platforms than its native one, the Mono framework has to be installed properly first, which makes it less easy for the average user to install or execute a C# based application on another platform. Also note that Mono is an open source project, which does not at all include all functionality provided in the ‘original’ Microsoft .NET framework yet. A lot of work still has to be done to fix this limitation.

- C++ in combination with Juce, an application framework written in C++ used to write native software on numerous systems like Windows, POSIX, Mac OS X, etc., with no need to change the source code. This could be a good solution for the problem, but since we both have do not have good experience in this language and it is a rather low-level programming language, we ought to leave it as it is.

- C or C++ in combination with XVT, a cross-platform toolkit for creating desktop applications in Windows, Linux and Mac-based platforms. For the same reason as status above, we kept this solution at our sides.

Then we have the possibility to combine already known programming languages to create the cross-platform application itself. This may sound difficult, but a good example to explain this approach is the Firefox Web Browser in combination with the Mozilla framework. This framework uses abstraction, as mentioned above, to provide low-level components along with different source code sets to implement platform-specific features, for example to build the Graphical User Interface (GUI) of an application. Concerning the portability, the framework makes use of XUL (eXtended Unified Language), CSS and JavaScript for extending the browser. Well-known cross-platform applications based on this approach are:

- Netscape Navigator, a well-known web browser
- Mozilla Firefox, a well-known web browser
- Mozilla Thunderbird, a well-known email client
- Songbird Media Player, a Mozilla framework based media player
- Miro, an internet TV application

Since the Mozilla framework has to be properly installed on every computer system in order to provide this cross-platform compatibility, we do not have any experience with this approach, and creating a user-friendly GUI by using different code sets is not easy to debug, as stated in the first approach, we avoided this approach too.
The last approach to provide cross-platform compatibility is by using web-based scripting languages such as PHP, Java-applets, JavaScript, ASP, etc. Web applications are typically described as cross-platform because they are accessible from (almost) any web browser, even on most recent mobile phones and tablets. Using a web application, sadly, requires a master-client system network structure, which proves to be less efficient as discussed in section 3.1.2. Moreover the implementation of network- and disk-based I/O while providing an easy to use user interface will be a very hard task with only the availability of JavaScript, CSS and HTML. Also note that disk and network access is usually for desktop applications, but by default this is restricted for browser based applets or websites, which quite annoying since this is exactly what our prototype needs. Using PHP or other server-based web languages also require a webserver, which will only take up more resources and will make it harder for the user to install and execute the application. Note that it also decreases the flexibility of the application since the webservers always has to be started and stopped when the prototype application would start. Moreover installing a webserver is not a straightforward task for the average user, which was the last argument to not make use of this approach.

3.2.3 Challenges

As described above, there are several ways to implement cross-platform applications, however there are quite some issues that have be kept in mind:

- The testing procedure may be considerably more complicated as different architectures can have slightly different behaviours, or even bugs.

- The lowest common denominator factor is very important. This means that developers better use the subset of features which are available on all platforms, which can hinder making use of more advanced features of a platform.

- Different platforms have different user interface (UI) conventions. Good examples are the minimise, maximise and close buttons in Mac OS X and Windows.

- The code of Virtual Machines’, like JVM, Mono and Juce, has to be translated to native code, which can be a performance drawback. JIT (Just-In-Time) compilation can speed up this process, but even using such techniques, a little overhead may be unavoidable.

- Installation of multi-platform programs can be harsh. A good example here are the DMG, MSI and RPM based installer structures in Mac OS X, Windows and Linux distributions. Yet making use of InstallBuilder or IzPack, 2 applications which can easily create native installed packages, can fix solve this challenge.
3.2.4 Conclusion

Keeping all information discussed before in mind, the best programming language for our prototype would be Java because:

- Java was conceived with the concept of WORA or ‘write once, run anywhere’, which is done using the JVM.
- The Java Virtual Machine is well-implemented and even features automatic exception handling, which provides information about the error independent of the source, making debugging a lot easier.
- The JVM is distributed along with a big set of standard class libraries that implement the JAPI or Java Application Programming Interface.
- The JVM can execute CLASS or JAR files and moreover nowadays JIT compiling is used to achieve greater speed and overcome the overhead mentioned in the previous section.
- Applications written in Java have an automatic garbage collection system while executing, which keeps the needed resources down and eliminate the need to dispose created objects or pointers.
- Java is a well-known programming language, lots of code examples and libraries are available for extending the standard functionalities.
- Our knowledge of Java much better in contrast to C(++), however similar to C#.
- Support for applications written in Java is implemented by default in the most operating systems, or can be installed very easily via Java.com, so there is no need to install application frameworks in contrast to Mono, Juce or all other frameworks.
- JVM separates from the operating system thus debugging different platforms yield the same results or bugs nor security issues will exist due to the change of architecture.
- Creating the GUI is quite simple and will be identical on every platform since Java uses libraries distributed in the JVM, which are the same for every operating system.

The advantages are quite obvious. This is why we chose to already start implementing a prototype based on our research as soon as we knew Java as the programming language.
3.3 Storing the indexed lists

3.3.1 Introduction

Keeping section 2.1 and 2.2 in mind, we concluded that the use of a database technology would be preferable over directories since the requirement for using directories, which were only optimised for reading, was to create a bulky directory server to host the information that is not suitable for small networks as in this thesis. Though databases are optimised for storing, updating and reading data and have a smaller footprint on the performance requirements, which justifies its popularity.

Figure 3.3.1: Database structure

Sadly most database solutions still need to setup a server, like a SQL server, which most of the time is not an easy task for the average user. Our prototype had to be very user-friendly so actually we were stuck on finding the optimal storage solution.

Luckily, after some more research we stumbled upon db4o, an open source object-oriented database system [10] natively available for Java and the .NET framework, which was very promising from the first sight already.

3.3.2 db4o

db4o is an object database management system developed and distributed by Versant Corporation and stands for ‘database for objects’. It differs from ordinary databases, like SQL, since it has much more expression-power and can not only store strings, integers, booleans and decimals in rows and columns, but in fact any object you like.

Figure 3.3.2: db4o logo

Software programs using different data persistence technologies are an integral part of the contemporary information handling solutions. More than often such systems are implemented with the help of an object-oriented programming language (Java, C#, etc.) such as in our prototype, and a relational database management system (Oracle, MySQL, etc.). This implementation originally contains a mismatch between relational and object worlds, which is often called object/relational impedance mismatch. The essence of the problem is in the way the systems are designed. Object systems consist of objects and are characterised by identity, state, behaviour and encapsulation whereas the relational model consists of tables, columns, rows and foreign keys, and is described by relations, attributes and tuples.
The object-relational mismatch has become enormously significant with the total adoption of OO (Object Oriented) technology. This resulted in the development of Object-Relational Mappers (ORM), such as Hibernate, EclipseLink or Entity Framework [11]. This solution cures the symptoms of the object relational mismatch by adding a layer into the software stack that automates the tedious task of linking objects to tables. However, this approach creates a huge drain on system performance, drives up software complexity and increases the burden on software maintenance, thus resulting in higher cost of ownership. While the mapper solution may be feasible in large, administered datacentre environments, it is prohibitive in distributed and zero-administration architectures such as those required for embedded databases in client software, mobile devices, middleware or real-time systems.

Significant side effects of the object relational mismatch manifest themselves in unnecessary system overhead with bloated footprints and runtime performance issues. Of course, there is also overall time to market delays due to poor developer productivity. The overhead still exists in ORM because under the covers, the runtime is still query driven. The more complicated your models, the more problematic keeping changes synchronised with the internal mapping will be.

Primary performance issues come from the fact that despite being called a "relational database", a RDBMS does not store direct relations. Relations are resolved at runtime by performing set based operations on primary-foreign key pairs. This means the application has to constantly re-discover data relationships at runtime resulting in immense CPU consumption for something that should be an inherent part of your application model. Further, because discovering these relations over and over again requires continual access to index structures and data to perform the set operations, contention is much higher within database internals leading to poor scalability of individual database processes. Furthermore, lack of direct storage of relations cause the application design to become query driven instead of object driven.

Using an object database (OODBMS), the relations are a fundamental part of the storage architecture. So, application design is model driven. You do not have to suffer any performance overhead for discovering a relationship. The relationships are just there and immediately accessible to the requesting thread. This makes the internal structures much simpler and therefore less contention exists with data requests. The result is that individual processes become more scalable under concurrency, it reduces the cost of development, support and versioning and thus generally, the overall system costs.

A big advantageous difference between db4o and SQL, Oracle or other technologies is that it does not require using queries anymore. Instead it uses so-called ‘native queries’, which are less error-prone since they do not require typing statements anymore. With native queries, you can simply create an object, and put it into the database by calling the .store() method of a db4o (embedded) server or client. Native queries also provide more security. Only the application which instantiated the db4o server or client can access the database in runtime, thus no security leaks for so called SQL injection. More details implementing native queries will be discussed section 4.2.3.

db4o is also very flexible. No need for setting up servers, completing complicated installations or doing other difficult procedures. You just need to include one db4o precompiled JAR file, call the .openFile(), .openServer() or .openClient() methods and close the session by calling the corresponding .close() function. Again, more details about these procedures can be found in section 4.2.3.
Moreover db4o also includes indexing, which makes native queries returning results faster, a big set of monitoring tools to test the performance, an implemented backup system and even supports the replication of databases among different hosts. This OODBMS is also designed to be cross-platform compatible, which is exactly what we need in our prototype. The only requirement for this extension to run is a Java- or .NET framework and an application, which makes the calls to the database.

### 3.3.3 Conclusion

It is quite obvious that we will use db4o to implement our prototype. It is free, well-supported, cross-platform compatible, secure, packed with a lot of monitoring tools, user-friendly and fast: the ideal technology to use as our Database Management System.
3.4 Discovering other clients

3.4.1 Introduction

In order for a client to access the shared media of other clients, first he has to know which clients are participating in the current network. Using broadcast messages can do this, however this is not an efficient way since it unnecessarily loads the network.

Using multicasts and setting up an area of interest by joining a particular multicast-group solves this problem and is preferable over broadcast messages.

Figure 3.4.1: Broadcasting

3.4.2 Broadcasts

Broadcasting refers to a method of transferring a message to all recipients simultaneously and can be performed in high- or low level operations. An application broadcasting Message Passing Interface is a good example of a high level operation, while broadcasting on Ethernet is a low level operation.

Figure 3.4.2: Broadcast example

In computer networking, which is our case, broadcasting refers to transmitting a network packet that will be received by every device on the network, which can be seen in the figure shown above, the red dot being the broadcasting device while the green ones are the recipients.

The scope of a broadcast is limited to a specific broadcast domain, which can be configured by adjusting the Time-To-Live (TTL) parameter of the packet that will be broadcasted.

Broadcasting a message is also in contrast to unicast addressing, which a host sends datagrams or packets to another single host identified by a unique IP address.

A limitation is that not all networks support broadcast addressing. For example neither X.25 nor Frame Relay have a broadcast capability. Moreover any form of an Internet-wide broadcast does not exist at all. Broadcasting is largely confined to Local Area Network technologies, most notably Ethernet and Token Ring, the latter being less familiar.

Both Ethernet and Internet Protocol Version 4, IPv4, use a 255.255.255.255 broadcast address to indicate a broadcast packet. Token Ring however uses a special value in the IEEE 802.2 control field of its data frame.

In those kinds of network structures, the performance impact of broadcasting is not as large as it would be in a Wide Area Network, like the Internet, however it is still there and better can be avoided.

Broadcasting can also be abused to perform a DoS-attack, a Denial-of-Service attack. The attacker then sends fake ping requests with the source IP-address of the victim computer. The victim’s computer will then be flooded by the replies from all computers in the domain.

The successor of IPv4, Internet Protocol Version 4, IPv6 also does not implement the broadcast method to prevent disturbing all nodes in a network when only a few may be interested in a particular service, instead it relies on multicast addressing, which will discussed in the next section.
3.4.3 Multicasts

Multicast addressing is a conceptually similar one-to-many routing methodology, but differs from broadcasting since it limits the pool of receivers to those that join a specific multicast receiver group. It is the delivery of a message or information to a group of destination computers simultaneously in a single transmission from the source automatically creating copies in other network elements, such as routers, however only when the topology of the network requires it.

As shown in the figure above, you can see that only the ‘interested’ green hosts, which joined the multicast receiver group, will receive the multicast information packets sent by the red multicast host. However the other ‘uninterested’ yellow hosts, not inside the multicast group, will not be tampered at all.

Multicast [12] is most commonly implemented in IP multicast in IPv6, as mentioned in the previous section, applications of streaming media and Internet television. In IP multicast, the implementation of the multicast concept occurs at the IP routing level, where routers create optimal paths for datagrams, or multicast packets, sent to a multicast destination address.

IP multicast is a technique for one-to-many communication over an IP infrastructure in a network. It scales to a larger receiver population by not requiring knowledge of whom, or how many receivers there are in the network.

Multicast uses the network infrastructure very efficiently by requiring the source to send a packet only once, even if it needs to be delivered to a large number of receivers. The nodes in the network take care of replicating the packet to reach multiple ‘interested’ receivers, however only when necessary.

The most common transport layer protocol to use multicast addressing is UDP, which stands for User Datagram Protocol. By its nature, UDP is not reliable because messages may be lost or delivered out of order. PGM, Pragmatic General Multicast, has been developed to add loss detection and retransmission on top of IP multicast.

Nowadays IP multicast is widely deployed in enterprises, stock exchanges and multimedia content delivery networks. A common enterprise use of IP multicast is for IPTV applications such as distance learning and televised company meetings.

Sadly no mechanism has yet been demonstrated that would allow the IP multicast model to scale to millions of transmissions together with millions of multicast groups and thus, it is not yet possible to make fully-general multicast applications practical.

Another drawback is that not all Wi-Fi Access Points support multicast IP, however this number is increasingly quite fast and is facilitating the WiCast Wi-Fi Multicast that allows the binding of data to geographical locations.

3.4.4 Conclusion

Keeping all information above in mind, it is quite clear that using IP multicast, or in short just multicasts, is the best technology to discover other clients in our client-only network structure. However we should keep in mind that not all Access Points support this kind of technology. Since its popularity is growing so fast, we strongly believe we should still use this new technology since it is much more efficient than ordinary broadcast packages.
3.5 Streaming the multimedia files

For sending multimedia over the local area network, there exist a couple of possibilities. One of the better solutions is called streaming. The concept of streaming can be explained very easily. On one side you have a client who is hosting a multimedia file and on the other side you have a client who wants to receive the multimedia file. In this case the hosting client sets up a network stream from the multimedia file over, by example, HTTP and the receiving client opens this stream. With this concept it is not possible to transmit a file, like you do for example when you are downloading a file from the World Wide Web. Though it is possible to receive the stream and watch the streamed media. The big advantage of streaming is the fact that you do not need to receive the multimedia file completely, you just open the broadcasted stream and you are able to watch it.

3.5.1 Java streaming libraries

The streaming of multimedia can be done by several possibilities that exist the current day. For the programming language we chose, Java, so the choice became more limited. Some examples are Xuggler, Java Media Framework (JMF) or vlcj. The latter one being the best choice for our thesis.

Vlcj [13] has the big advantage that it is based upon the very popular VLC Player from VideoLAN. This VLC Player is very versatile, stable and is loved by many people on the internet. Thanks to its popularity, the online community of vlcj is very active and up-to-date. Furthermore the possibilities of the vlcj libraries are very widespread and easy to use with, known that there exist a lot of tutorials and example code.

3.5.2 Java binding for VLC media player

Vlcj or the Java bindings for the VLC Media Player will be explained in this section. Because of the versatility of the vlcj library, we will only focus on the streaming ability and do not expand the explanation about the general structure of the Java VLC library.

The options of streaming with vlcj are numerous, you can stream just about anything with these Java libraries. This can be simply explained since vlcj wraps libvlc, which is the library of the original VLC Media Player. The syntax and the streaming options that you use with vlcj are defined by libvlc.

A good practice is to use the VLC QT application to get the required streaming commands. The reason behind this is that when you choose to stream a multimedia file, as a result a dialog box will pop up and show you the streaming commands you will need. The resulting streaming command, which you received from the dialog box, can then be passed as media player options to vlcj in order to play your media files.

3.5.3 Example code of Java bindings for VLC media player

With a small example, the idea behind the streaming of vlcj will be clearer. We start off with the server-side of the streaming and then we continue with the client-side, which is much shorter and easier.
3.5.4 General setup of streaming server

So for the streaming server, start by obtaining an instance of a HeadlessMediaPlayer:

```java
String[] libVlcArgs = {"arguments of VLC"};
MediaPlayerFactory mediaPlayerFactory = new MediaPlayerFactory(libVlcArgs);
HeadlessMediaPlayer mediaPlayer = mediaPlayerFactory.newMediaPlayer();
```

After setting up the HeadlessMediaPlayer class, you need to start the media to want to stream.
So the next step is play your multimedia file, and like mentioned above you will pass the the crucial media options:

```java
String[] mediaOptions = {"media options for VLC"};
mediaPlayer.playMedia("<media file>", options);
```

For your own convenience, it can save you a lot of work if you use the following code. With this approach, you just need to pass the media options once. So when you play a lot of multimedia files, you do not need to pass the media options every time, if of course you want to use the same media options every time.

```java
mediaPlayer.setStandardMediaOptions(mediaOptions);
```

3.5.5 Media options example: HTTP

From the different options available at the time of writing, we choose HTTP to be the better standard in this case. Some other examples are Windows Media HTTP Streaming Protocol Specification (MS-WMSP), Real Time Streaming Protocol (RTSP), Real-time Transport Protocol (RTP), User Data Protocol (UDP) and IceCast.

Because we are determining a standard protocol, both the server and the client need to support this protocol. In this way we first continue setting up the server and then we finish off with a simple coding example of the client.

So firstly we have a very simple example of HTTP multimedia streaming, though it also has the capabilities to define more streaming options.

Streaming server:

```java
mediaPlayer.playMedia("<media file>",
    ":sout=#duplicate{dst=std{access=http,mux=ts,dst=<IP:portnr>}}");
```

Client, who wants to open the stream:

```java
mediaPlayer.playMedia("<IP:portnr>");
```
3.5.6 Test setup of Java bindings for VLC media player

The above coding sample is easy to use and to show an example of the usage, this section will give you a test setup of a possible solution.

For this solution we used the standard VLC Media Player to stream a multimedia file over the local area network using the HTTP protocol. To open this stream we use the Java bindings, vlcj, for VLC Media Player.

The VLC Media Player is streaming a MP4 multimedia file that is encoded with H.264 for the video and MPEG AAC for the audio. To emulate the local area network, the setup consists of a hosting machine with a virtual guest machine, using Microsoft Virtual PC. The file will be streamed on the host machine, while the stream will be opened on the Virtual Machine.

To open the stream, we make use of the test media player, which is constructed of the vlcj library. This Java application will be running on the Virtual Machine.

![Figure 3.5.4 a: Test setup of streaming with vlcj, opening the stream using HTTP](image)

In the figure above, you can see the setup of our approach. At the left it is showing the VLC Media Player on the host operation system that is streaming the multimedia file and on the right the Virtual Machine with the Java application (based upon vlcj) that will open the stream using HTTP.

The following picture shows the Java application that has successfully opened the stream and playing the multimedia file that is streamed by the VLC Media Player.
3.5.7 Conclusion

With a short introduction of streaming we can conclude that this is a good solution for providing a flexible way to exchange multimedia throughout the local area network. We concluded to use Java bindings for VLC Media Player because from the different libraries available it was clear to see that this is a very solid solution and moreover, easy to use. The test setup shown above shows how easy and fluent the library is in usage.
4 Detailed Description of the Investigated Solution

4.1 Physical or Hardware Structure

As for the physical or hardware structure of our prototype, the investigated solution exists of an amount of computers (desktops or notebooks) connected via the local area network.

![Network Diagram](image)

The different participants will be running the Java application using the Java Virtual Machine and in this way the choice of the host’s platform is completely up to the participant. The different connected computers will be sharing data and having connections to the other computers. In this way they are able to search for the shared media of the others, including their own.

The local area network is a prerequisite of the investigated solution. The structure is not aware of the underlying layers like the physical media connections. Before setting up the application you need to be able to confirm that the TCP/IP network is working properly while moreover being able to support multicast addressing, which will be discussed in section 4.2.4.
In order to give a quick overview for understanding our prototype we thought it would be best to provide a class diagram. With a class diagram you can see the relations between different classes, which are the yellow arrows, and it significantly eases the need for complicated explanations.
Figure 4.2.1 b: Class diagram 2
4.2.2 **Threading**

In computer science, since threading can have a lot of meanings, a thread of execution is the smallest unit of processing that can be scheduled by an operating system. It generally results from a fork of a computer program into two or more concurrently running tasks, such as processing image files or compressing files. The implementation of threads and processes differs from one operating system to another. However, a thread is always contained inside a process [14].

![Figure 4.2.2 a: Threading](image)

Multiple threads can exist within the same process and share the process’ resources such as memory, while different processes running on the same system, do not. In particular, the threads of a process share the process’ instructions and its context, which are the values that its variables reference at any given moment. To simplify we can state that multiple threads in a process are like different students reading off the same book and following its instructions, though not necessarily from the same page.

Threading, or multithreading, generally occurs by TDM on single processors. In this case the processor switches between different threads very frequently such that the user perceives the threads as running at the same time. On a multiprocessor or multi-core system, the threads or tasks will actually run at the same time, with each processor or core running a particular thread or task.

There are quite some advantages from threads over processes. Threads share the process’ resources, but are also able to execute independently. It provides useful abstraction of concurrent execution but moreover, when it is applied to a single process, it can enable parallel execution on a multiprocessor system, which is much faster than executing tasks serially. Since threads share their address space within the process and new address space does not need to be assigned when creating new threads, context switching between threads in the same process is typically faster than context switching between processes.

Another advantage, even for single-CPU systems, is the ability for an application to remain responsive to input. In a single-threaded program, if the main execution blocks on a long running task, the entire application can appear to freeze. By moving such long running tasks into a worker thread, that runs in parallel, or concurrently, with the main execution thread, it is possible for the application to remain responsive even while executing tasks in the background.

It all seems pretty flawless, however in some cases, it is not. Programmers have to be careful and need to avoid race conditions. Threads will often need a rendezvous to process the data in a correct order. They may also require mutually exclusive operations, to prevent common data from being simultaneously modified, or read while in the process of being modified.

Having only one drawback, mentioned in the previous paragraph, while the rest of the arguments all being advantages, we decided to implement threading in our prototype. In total we used 5 threads, since the main execution path can be considered as a thread too. In chronological order, as implemented in our prototype, a summary of our threads is described below:

- Main thread, starts immediately when the application is launched
- Thread_Index, executes when the user choose his directories to share
- Thread_ParseLocalIndices, runs when the GUI is started
• Thread_BroadcastPresence, runs when ParseLocalIndices completed
• Thread_GetConnectedClients, runs when BroadcastPresence started running

The main thread cannot be discussed thoroughly because it is automatically being handled by the JVM and starts running when the prototype, or in fact every application, starts. A process cannot exist without a thread. It always needs one main thread that provides tasks to the process. When a process without a thread would be able to exist, this process would be useless since it would not have any tasks assigned to it.

The Thread_Index however, which starts when the user has decided which directories to share after pressing the ‘Scan directories’ button as shown in the figure below, had to be implemented and controlled by ourselves. The code behind the Scan directories button looks as follows:

```java
private void startIndexing(java.awt.event.ActionEvent evt) {
    saveSettings();
    GUI_IndexProgress guiIndexProgress = new GUI_IndexProgress(this, lisSharedDirectories.getModel());
    [CODE OMITTED]
}
```

As you can see, the settings, being the shared directories, are first being saved by calling the method saveSettings(). After this, a new Graphical User Interface will be instantiated by passing the shared directories to it. This IndexProgress GUI will keep track of the progress of the Thread_Index, which will be started in the constructor of the IndexProgress GUI, as shown below:

```java
public class GUI_IndexProgress extends javax.swing.JFrame {
    private final Thread_Index thrIndex;
    private final javax.swing.Timer timUpdateTimer;
    public GUI_IndexProgress(Frame parent, ListModel lisModel) {
        [CODE OMITTED]
        thrIndex = new Thread_Index(lisModel);
        timUpdateTimer = new javax.swing.Timer(1000, new java.awt.event.ActionListener() {
            @Override
            public void actionPerformed(java.awt.event.ActionEvent ae) {
                if (thrIndex.booRunning) {
                    progBar.setValue(thrIndex.bytProgress);
                }
            }
        });
    }
}
```

![Figure 4.2.2 b: Settings GUI showing status before starting Thread_Index](image-url)
Analysing the code, you can see that the Thread_Index thrIndex is being instantiated right from the start, even before the class’ constructor is called, together with a swing.Timer timUpdateTimer, which is used to periodically check the progress of the thread. However you have to keep in mind that thrIndex did not start yet, as you will see discussed below.

In the public constructor of the GUI, you will see this statement: “thrIndex = new Thread_Index(lisModel);”, this is where the Thread_Index will be eventually started since the constructor of it looks like this:

```java
public Thread_Index(javax.swing.ListModel lisModel) {
    this.lisModel = lisModel;
    thrRunner.start();
}
```

Note that thrRunner in this class has already been instantiated as a Thread-object with:

```java
private final Thread thrRunner = new Thread(this, "Thread_Index");
```

Finally, when thrRunner.start() will be executed, the program will automatically start executing the following code:

```java
@Override
public void run() {
    try {
        System.out.print("Thread: Index running... ");
        [CODE OMITTED]
        System.out.println("Done!");
    } catch (Exception ex) {
        System.out.println("ERROR: The indexing did not succeed! Reason: " + ex.toString());
    }
}
```

As you may think, implementing threads is not that easy and it brings complicated structures to the code of the prototype, however, the result and the advantages attached to it are really worth it. The result of all code discussed above looks like this:

![Figure 4.2.2 c: Result of threading in combination with a GUI](image-url)
While the thread is doing its work and indexing all media files in the specified drives or directories, the prototype remains responsive and meanwhile shows the progress of the indexing procedure together with some interesting statistics. When the indexing is done, the public boolean, booRunning, will be set to false, which will make the timUpdateTimer and thrIndex stop running. After this, the Main GUI will be created and new threads are started.

Since we already explained the implementation of threads by Thread_index, the remaining ones will be discussed more briefly. Thread_ParseLocalIndices for example is implemented a little different. This thread starts automatically when the main GUI is created and checks all indices stored in the database in order to be shared to other users. The syntax looks like this:

```
thrParseLocalIndices = new thread_ParseLocalIndices(
    cliLocalClient.strServerIP, cliLocalClient.intPortNumber,
    cliLocalClient.strUserName, cliLocalClient.strPassword);
[CODE OMITTED]
thrParseLocalIndices.start();
```

As you can see, it requires quite some arguments to be able to put the checked indices into the shared db4o database, which will be discussed in the next section.

The other threads: Thread_BroadcastPresence and Thread_GetConnectedClients are implemented in a similar way using the .start() method together with swing.Timers which periodically check the progress of the running thread.

BroadcastPresence multicasts the client’s presence over the network by periodically sending UDP packets containing information about himself and credentials needed for db4o.

GetConnectedClients, as the name is quite self-explanatory, handles the discovery and administration of connected clients and provides the functionality to get a list of all connected clients in the network.

Threads were really useful to help implementing our prototype. They made it possible to do useful work in the background, without the need the user has to wait or will be annoyed by an application that would looked frozen while executing.
4.2.3 Implementation of db4o

The implementation of db4o, the OODBMS discussed in section 3.3.2, was quite easy. By using Netbeans as our IDE environment, it was sufficient to download the all-in-one JAR file from db4o.com and add it to our prototype’s project as follows:

![Image of Netbeans project properties](image)

You just needed to go to the project’s properties, browse to libraries, then press Add JAR/Folder, search for the db4o JAR you downloaded from the website and finally add it to your project.

However, in order to make use of it, one more thing was required. In every class file, especially in GUI_Main where db4o is administered, the following lines of codes are needed:

```java
import com.db4o.ObjectContainer;
import com.db4o.ObjectServer;
import com.db4o.ObjectSet;
import com.db4o.cs.Db4oClientServer;
import com.db4o.query.Predicate;
```

Importing could also be done by using `com.db4o.*`, `com.db4o.cs.*` and `com.db4o.query.*`, however this approach would import too many objects, which would not be used after all. That's why we choose to only import the necessary objects, which helps in improving the performance of the prototype.
ObjectContainer is an object that provides the ability to be instantiated as a session to access a db4o database. ObjectServer is an object that holds db4o server capabilities and ObjectSet is an object, close to an arraylist of specified objects, which the results of native queries are stored in. Db4oClientServer has to be imported since it provides methods to initialise or instantiate all objects mentioned above, in particular the methods used to create a db4o server and client. Finally the Predicate defines which object to search for while using native queries, which will be discussed later on.

To start with, we will discuss the use of db4o as a local database. This approach is used to store the indexed media files created in Thread_Index, mentioned in section 4.2.2, together with the shared directories or settings of the prototype.

The code to write the settings to a local db4o database is as follows:

```java
private void saveSettings() {
    try {
        System.out.print("Saving settings... ");
        new java.io.File("wShareLocal.wsl").delete();
        final com.db4o.ObjectContainer db4oSettings =
            com.db4o.Db4oEmbedded.openFile("wShareLocal.wsl");
        db4oSettings.query(new
            com.db4o.query.Predicate<javax.swing.ListModel>() {
                @Override
                public boolean match(javax.swing.ListModel limModel) {
                    db4oSettings.delete(limModel);
                    return true;
                }
            });
        javax.swing.ListModel lisModel = lisSharedDirectories.getModel();
        db4oSettings.store(lisModel);
        db4oSettings.commit();
        db4oSettings.close();
        System.out.println("Done!");
    } catch (Exception ex) {
        System.out.println("ERROR: The settings could not be saved!
            Reason: " + ex.toString());
    }
}
```

Analysing the code carefully it can be seen that the current settings together with all indices will be deleted each time the settings will be saved since they are stored in the “wShareLocal.wsl”-file. However if this will fail, which would be quite exceptional, a native query will be executed to delete the existing settings. Those queries always start by using:

```java
ObjectContainer db4oSettings =
    com.db4o.Db4oEmbedded.openFile("wShareLocal.wsl");
```

This creates a session db4oSettings, connected to the local database, which can be used to execute queries. The actual query looks like this:

```java
db4oSettings.query(new com.db4o.query.Predicate<javax.swing.ListModel>() {
    @Override
    public boolean match(javax.swing.ListModel limModel) {
        db4oSettings.delete(limModel);
        return true;
    }
});
```
A novel approach to local multimedia sharing

Notice that the Predicate ListModel is used here to specify that the query should only return objects of this kind. By using the overriding function match(), which will return true when it found an object of the same kind, it will delete the existing settings in the database, which are stored as a ListModel object. Note that the results of the query here are not stored in an ObjectSet, which is not needed when only deleting objects in database.

Finally the new settings, contained by lisModel, are simply stored using:

```java
db4oSettings.store(lisModel);
```

Afterwards, we still need to tell the database that the new data can be committed and since we first opened the file, we need to close it too:

```java
db4oSettings.commit();
db4oSettings.close();
```

Of course it is always wise to surround such a native query by a try-catch statement since things can go wrong too. It simplifies the debugging a lot because the error messages returned by db4o are very self-explanatory.

As you can see native queries are not that hard to implement and, as stated in the section describing db4o, it provides more security than SQL statements.

Since now we had a local database storing the local indexed media files, we still needed an approach to get this data to other clients participating in the same network. That is the reason why we also implemented a db4o server and a db4o client. A db4o server was needed for other clients to browse through the shared media of one particular client and the way doing so, was by using a db4o client connecting to this particular server.

However in order for this approach to work, all clients should not only send their presence via multicasts, they also should exchange information about accessing the database among other clients, which will be discussed in section 4.2.4.

As discussed in the section describing threads, ParseLocalIndices periodically parses all indices stored in the local db4o database, checks their availability and afterwards puts them into a db4o server to share them across the network. The code to implement this functionality is pretty similar to db4o’s local database administration, as discusses above, however now needing credentials.

Supposing credentials are successfully sent between different clients, the ParseLocalIndices thread uses the following code to create a session to browse through the local database:

```java
private final com.db4o.ObjectContainer db4oLocal =
    com.db4o.Db4oEmbedded.openFile("wShareLocal.wsl");
```

However doing the same to create a session for the server database db4oShared:

```java
private final com.db4o.ObjectContainer db4oShared;
```

This variable will be instantiated in the constructor of the ParseLocalIndices thread:

```java
public Thread_ParseLocalIndices(String strServerHost, int intPortNumber,
                                 String strUsername, String strPassword) {
    db4oShared = Db4oClientServer.openClient(strServerHost, intPortNumber,
                                            strUsername, strPassword);
    thrRunner.start();
}
```

As you can see credentials are needed to access the server database, which was created in the GUI_Main class, and need to be passed to the ParseLocalIndices thread. In the run() method of the thread, the important work will be done. The code will be discussed again since it is a good example about native querying.
Detailed Description of the Investigated Solution

The following code will execute a query on the local database only returning MediaObjects, which are indexed media files, which still exist on the local computer and store them in an ObjectSet obsMediaObjectsLocal:

```java
@override
public void run() {
    System.out.print("Thread: ParseLocalIndices running... ");
    ObjectSet<Class_MediaObject> obsMediaObjectsLocal =
        db4oLocal.query(new Predicate<Class_MediaObject>() {
            @Override
            public boolean match(Class_MediaObject mobObjectLocal) {
                return new File(mobObjectLocal.strFullPath).exists();
            }
        });

    ObjectSet<Class_MediaObject> obsMediaObjectsShared =
        db4oShared.query(new Predicate<Class_MediaObject>() {
            @Override
            public boolean match(Class_MediaObject mobObjectShared) {
                return true;
            }
        });

    db4oLocal.commit();

    boolean booExists = false;
    long lngCurrent = 0;
    double dblTotal = obsMediaObjectsLocal.size();

    for (Class_MediaObject mobObjectLocal : obsMediaObjectsLocal) {
        lngCurrent++;
        strProgress = new DecimalFormat("0").format((lngCurrent / dblTotal) * 100) + "...";
        for (Class_MediaObject mobObjectShared : obsMediaObjectsShared) {
            if (mobObjectLocal.matches(mobObjectShared)) {
                booExists = true;
            }
        }
        if (!booExists) {
            db4oShared.store(mobObjectLocal);
            booExists = false;
        }
    }

    db4oShared.commit();

    booRunning = false;
    db4oLocal.close();
    db4oShared.close();
    System.out.println("Done!");
}
```

The following code will execute a query on the local database only returning MediaObjects, which are indexed media files, which still exist on the local computer and store them in an ObjectSet obsMediaObjectsLocal:
Putting the results of this query into the server database was not that easy. First we needed to check whether the MediaObjects found in the local database already exist in the shared database, because avoiding similar object entries is very important. First we fetched all MediaObjects found in the shared database and then compared the MediaObject found in the local database to them. If the object does not exist yet, it is put into the server database by:

```java
if (!booExists) {
    db4oShared.store(mobObjectLocal);
    booExists = false;
}
```

That’s all, not forgetting to close the open sessions, this is the code required to copy local database objects to the server db4o database. When using the search functionality of our prototype though, a db4o client is needed, but having a rather functional description, this will be discussed in section 4.3.1.

### 4.2.4 Implementation of multicasts

Also mentioned in the section about threading is the task of the BroadcastPresence thread. It was responsible for multicasting the client’s presence over the local network by using multicasts with UDP packets. Analysing the code in the run() method from this class gives a good explanation about how we succeeded in this purpose:

```java
@Override
public void run() {
    System.out.println("Thread: BroadcastPresence running... ");
    try {
        java.net.MulticastSocket mcSocket =
            new java.net.MulticastSocket(8717);
        java.net.InetAddress inaAddress =
            java.net.InetAddress.getByName("230.0.0.1");
        System.out.println("        Multicast: " + strClientName + "/" + strServerIP + "/" + intPortNumber + "/" + strUserName + "/" + strPassword);
        byte[] bytMsg = (strClientName + "/" + strServerIP + "/" + intPortNumber + "/" + strUserName + "/" + strPassword).getBytes();
        mcSocket.joinGroup(inaAddress);
        mcSocket.setTimeToLive(127);
        java.net.DatagramPacket packet =
            new java.net.DatagramPacket(bytMsg, bytMsg.length, inaAddress, 8717);
        while (true) {
            mcSocket.setSendBufferSize(bytMsg.length);
            mcSocket.send(packet);
            Thread.sleep(bytSleepTimerInSeconds * 1000);
        }
    } catch (InterruptedException iex) {
        System.out.println("Thread: BroadcastPresence stopped by user...");
    } catch (Exception ex) {
        System.out.println("ERROR: The BroadcastPresence-thread stopped abnormally! Reason: " + ex.toString());
    }
}
```
As you can see, sending UDP multicast packets are made possible by using a java.net.MulticastSocket object. In our prototype we used the fixed port 8717, as used in its constructor, because most ports below are officially reserved by other applications and we wanted to avoid congestions.

The next line, the inaAddress is a very important object. This object stores an address, which will decide the interest of the client. As you can see in the following line of code:

```
mcSocket.joinGroup(inaAddress);
```

The MulticastSocket object will join the group specified by the inaAddress. As described in section 3.4.3, this actually tells the prototype to join a multicast receiver group so it will only receive multicast packets from the same hosts who joined the same group. However here, it is used to specify the destination receiver group of the multicast packet, which will be the clients participating in the same group.

Afterwards the TTL value for the UDP packet will be set. This value determines how far the packet being transmitted will be able to travel through the network. The meaning of different TTL values can easily be found on the Internet.

Now the datagram packet is being created with the same settings. Note that the data being send over the network has to be an array of bytes, which in our case is:

- The name of the client (Username @ Computername)
- The local host’s IP address, for other clients to connect to the db4o server.
- The local host’s port number of the db4o server, which is chosen randomly and also needed for other clients to connect to the local db4o server.
- The remote username and password of the local client, which will be used for the local client to access other clients’ db4o server. These credentials will be used on other clients, which will grant access to this user in their server db4o database.

After the creation of the datagram, the multicast socket is ready to send the UDP packet over the network. This is how we implemented multicast notifications of client presence in our prototype.
4.3 Behavioural or Functional Description

4.3.1 Search function

As mentioned before in section 4.2.3, when using the search function a db4o client is needed. We found a nifty way for each client to be able to only store his indexed media file list itself, while still being able to search through other client’s multimedia. A code sample out of our project will be shown to help explaining the approach we used.

```java
java.util.ArrayList<String> arrResults = new java.util.ArrayList<String>();

for (Class_Client cliClient : thrClientNotifier.cliConnectedClients) {
    try {
        com.db4o.ObjectContainer db4oShared = null;
        if (cliClient.strClientName.matches(cliLocalClient.strClientName)) {
            db4oShared = Db4oClientServer.openClient(cliLocalClient.strServerIP,
                                                    cliLocalClient.intPortNumber, cliLocalClient.strUserName,
                                                    cliLocalClient.strPassword);
        } else {
            db4oShared = Db4oClientServer.openClient(cliClient.strServerIP,
                                                    cliClient.intPortNumber, strRemoteUserName, strRemotePassword);
        }
        ObjectContainer session = db4oShared.ext().openSession();
        ObjectSet<Class_MediaObject> obsSharedObjects = session.query(new Predicate<Class_MediaObject>() {
            @Override
            public boolean match(Class_MediaObject mo) {
                if (mo.bytCategory == 0 && tbtnPictures.isSelected()) {
                    return mo.strFullPath.toLowerCase().indexOf(txtSearchBox.getText().toLowerCase()) > -1;
                } else if (mo.bytCategory == 1 && tbtnMusic.isSelected()) {
                    return mo.strFullPath.toLowerCase().indexOf(txtSearchBox.getText().toLowerCase()) > -1;
                } else if (mo.bytCategory == 2 && tbtnVideo.isSelected()) {
                    return mo.strFullPath.toLowerCase().indexOf(txtSearchBox.getText().toLowerCase()) > -1;
                } else {
                    return false;
                }
            }
        });
        for (Class_MediaObject mo : obsSharedObjects) {
            arrResults.add(mo.FileName() + "ù" + mo.Extension().toUpperCase() + "ù" + mo.getSize());
        }
        session.commit();
        session.ext().purge();
        session.close();
        db4oShared.ext().purge();
        db4oShared.close();
    } catch (Exception ex) {
        System.out.println(ex.toString());
    }
}
```
As you can see in the second line, there is a for-loop iterating through all connected clients stored in the public variable cliConnectedClients, which are gathered by the thread GetConnectedClients. Since this latter thread has not been described yet, we will do this here at the functional description.

The run() method of this thread periodically checks for incoming multicast UDP packets, in the same way it joined the multicast receiver group discussed in section 4.2.4:

```java
java.net.MulticastSocket mcsSocket = new java.net.MulticastSocket(8717);
mcsSocket.joinGroup(java.net.InetAddress.getByName("230.0.0.1"));
System.out.println("Thread: GetConnectedClients running...");
```

After this code a never-ending while-loop is programmed to analyse the incoming datagram packets. There are 2 cases: either a client just sends its presence or it multicasts that it will leave the network by starting its datagram by ‘STOP’.

In the first case, a client, while it does not exist in the client-list already, will be granted access to the shared db4o database of the local host with the username and password that was included in the multicast message. Because of this, all remote clients are automatically granted access to the local client’s shared database. All remote clients, including the client required to access the local shared database, are Client objects collected in cliConnectedClients containing all necessary credentials. In the second case, when the datagram just starts with ‘STOP’, the user will be deleted out of the list and will lose his access to the server’s database since the next time it comes online, it will have other automatically random assigned credentials.

Ok, so when the search function is called, this for-loop will pass to all clients connected to the network. For each user a new session will be created and native queries are used to fetch the media of interest out of the shared db4o databases. The results will be added into arrResults and will later be displayed in the result window of the main GUI:

![Figure 4.3.1: Search results showed in the main GUI](image-url)
4.3.2 Compression of the database

The prototype even has a smart compression feature built-in. When the application is running, there will be 2 files present in the directory where \textit{wShareLocal.jar} is located: \textit{wShareLocal.wsl} and \textit{wShareLocal.wsl\_}.

The first file, without the underscore, contains the local database and is the result of thread \textit{Thread\_Index}. This information will only be renewed when the user decides to add or remove directories to share. Otherwise, this information will stay persistent and will be used by \textit{ParseLocalIndices}, like mentioned above, to build up the shared db4o server database.

The second file \textit{wShareLocal.wsl\_} is the result of \textit{ParseLocalIndices}, holds the current shared database and will be automatically deleted every time the application shuts down. This file is accessible by other clients with their provided credentials, as mentioned in the previous section.

Thus when the application has shut down, the only file that remains is \textit{wShareLocal.wsl}, which can be a big file when sharing a lot of multimedia. However, since the file should not be accessed by other applications and the users prefer small footprints for applications, we decided to compress this local database file while shutting down. The implementation of this was quite easy and can be seen in the code below:

```java
private void parseLocalDatabase(boolean booZip) {
    if (booZip) {
        try {
            System.out.print("ZIP: Compressing local database...");
            FileInputStream fisFile = new FileInputStream("wShareLocal.wsl");
            byte[] buf = new byte[(int) new File("wShareLocal.wsl").length()];
            fisFile.read(buf, 0, buf.length);

            CRC32 crc = new CRC32();
            ZipOutputStream s = new ZipOutputStream((OutputStream) new FileOutputStream("wShareLocal.wsl\_"));
            s.setLevel(9);

            ZipEntry entry = new ZipEntry("wShareLocal.wsl");
            entry.setSize((long) buf.length);
            crc.reset();
            crc.update(buf);
            entry.setCrc(crc.getValue());
            s.putNextEntry(entry);
            s.write(buf, 0, buf.length);
            s.finish();
            s.close();

            new java.io.File("wShareLocal.wsl").delete();
            new File("wShareLocal.wsl\_`).renameTo(new File("wShareLocal.wsl"));
            System.out.println("Done!");
        }
    }
}
```
As you can see this method is called with a parameter that decides whether the file should be compressed or not. Using example Java ZIP code, which can easily be found on the Internet, the compression and extraction of the local db4o database file was very easy. This method also replaces the original wShareLocal.wsl by the same content, however now compressed with the maximum compression available since the method setLevel(9) is used. 9 is the number to configure the ZipOutputStream to use maximum compression, while 1 is the number required to just store the files and do not apply any compression algorithm.
The next time the prototype will be launched, it will check whether the file \textit{wShareLocal.wsl} already exists. If so the application will directly open the main GUI window of the prototype and will start extracting it. Afterwards, \textit{ParseLocalIndices} will check all indices and put them into the shared db4o database to share the media among other clients.

The result of this feature can be seen in the figure below. The top figure shows the state when the application is running, while the bottom one shows the state when not.

![Figure 4.3.2 b: Files in the prototype's directory when running](image1)

![Figure 4.3.2 c: Files in the prototype's directory while stopped](image2)
5 Test or Analysis of Solution

In this section of the thesis, the test and analysis of the investigated solution will be given. We developed a prototype in Java, which implements the technologies we investigated. Based on this prototype, the test results will be written and an analysis will be done.

For the tests, we have chosen for the main and most important topics of the prototype previously discussed in our thesis.

5.1 Prototype setup

Of course the developed prototype needs to be setup in order to run. In this case the used programming language is Java, so the application consists only of one .jar file. In this way there is no need to install the application, but you need to have the Java Virtual Machine installed.

The next requirement for the setup is to have a router (or layer 3 switch) that supports multicasts and does not block them. The discovery of clients will be done by joining a multicast group, as described in section 3.4.3, and if the clients cannot transmit the multicast throughout the router, the other group members will not receive the multicast. Then they will not be aware of the new client. Together with the need of a multicast compatible router, you need to make sure that the local area network properly working, so that communication between the clients can take place.

The last requirement is to have multimedia files that can be shared among other clients. These files will be indexed and these indices will be stored in the object-oriented database.

In our setup all the requirements are met, so the testing procedure of the application can start.

5.2 Indexing

The multimedia files that need to be shared have to be indexed first in order for others to search in your database for these files. In this case we will be sharing the user directory on a Windows 7 machine as you can see on the following figure:

![Figure 5.2 a: Adding the user directory, this will be indexed in the next step](image)
After adding this directory to the list directories, which will be scanned for indexing, the indexing of this directory can start after pressing the ‘Scan directories’ button.

![Image](image)

**Figure 5.2 b:** The user directory is being indexed

The indexing was completed successfully and we receive the following window of the prototype:

![Image](image)

**Figure 5.2 c:** The indexing is completed successfully

In the next section we can confirm that the data is indexed, when we are checking out the functionality of the database solution, db4o.

### 5.3 db4o

After successfully testing the indexing part of the prototype, we moved on to test the database solution to prove it worthiness.

We kept the application running and to test db4o, we did a couple of local searches. The tests were successful again! On figure 5.3 a, you can see an example of the searches we did.
As shown above you can see that the files are indexed and that for every file the name, extension and the size is given. This can be confirmed by a second test that we can do. We can namely check if the db4o database file is created and if data is residing in it.

In figure above you can see in the Windows Explorer window, aside from the executable .jar file and the lib folder, that the prototype application created two new files. These are the db4o database files as discussed in section 4.3.2. In the Notepad window, you can see that the indexed files are stored in the database. The marked line in Notepad shows the directory of an .mp3 file.
5.4 Client discovery
For this part of the prototype testing, we started up one Windows XP Virtual Machine and another computer, an iMac, on the same network. As a result, we received the other participating clients in the network.

From this moment on, it was possible to do not only local searches, but also searches in the other clients’ databases.

![Image of client discovery in the local area network, with included network performed search](image)

In the figure above, you can also confirm that not only the clients are discovered, but also the network search has been performed successfully.

5.5 User-friendly GUI
User-friendliness is an objective thing to measure, but if you keep your application basic and simple, it will be accessible for a big audience. This is what we did, the initial start-up of the Java application takes 3 steps and when you rerun it afterwards it takes only a double click to start it up again.

To get a general idea about the user friendliness of the graphical user interface, we passed the application to a heterogeneous group of people and asked for their opinion. With a heterogeneous group we indicate that the participants have different backgrounds of studies and different level of computer knowledge.

After a quick survey, we got a general answer that the application was easy to understand. Of course we explained the concept of the application first, so they understood the use of this application. A couple of people who tested the application even responded that with this structure you do not need to mess around with difficult settings and you are just able to run the application without complications.
5.6 Cross-platform compatibility

The last important test that needed to be conducted, is to check if the application can be run on different operation systems. The following figure shows you that it was no problem to run the Java application on Windows XP (left top corner), Mac OS X Snow Leopard (middle bottom) and Windows 7 (right top corner).

Figure 5.6: the prototype running on different operating systems.

It was not only possible to run the application on the different operating systems, it was also no problem for the different participants to interact with each other. So we could do a search on the Mac operating system and search through the databases stored on the hosts using the Windows platform.
A novel approach to local multimedia sharing
6 Conclusions

We can positively conclude all research we have done about this project. We found that there are already quite good state-of-the-art technologies that provide local multimedia sharing like Orb or DLNA Certified software. However they are still flawed in certain areas.

DLNA, for example, lacks the support of commonly used codecs, while Orb needs an Internet connection and an inefficient ‘always on’ home computer where all media has to be centralised first, which is a bulky process.

Finding optimal approaches for those problems is exactly what we have handled in this thesis. We found a good programming language, Java, to implement a demo prototype, we found a great object-oriented database structure to store the indexed media, db4o, and we have chosen a well-founded and flexible client-based network structure. Finally VLC’s Java libraries can handle streaming of media, providing massive media file support without any need to re-encode or worry about the multimedia file structure.
A novel approach to local multimedia sharing
7 Future work

The general goals of our thesis are achieved, though there is still work that succeeds ours and needs to be done. We will explain the future work of our thesis in this section. It can be divided into three main parts.

7.1 ID3 tags

The information of video files or pictures can be extracted by using the name of the file. For music, this information is not sufficient. With music files, especially .mp3 files, you can extract a lot of information when you are making use of ID3 tags.

The ID3 tags that are present in .mp3 files contain information about the music file itself. You can use a lot of different fields, like by example the artist, album, title, release date, genre and rating of the song. After extracting this information, the user is able to use more specific search terms, but to accomplish this you first need to be aware of some things.

First of all, the current db4o is not compatible to store the extra information, so you need to adjust the object you store into the object-oriented database. A good recommendation is to store at least the artist, title and album of the .mp3 file, these are the most common used search criteria.

Another important thing to know is that there are a lot of Java libraries available with the capability to extract ID3 tags out of .mp3 files. After some research we conclude that the Jaudiotagger library was a solid solution to this problem. Jaudiotagger has a lot of support from an online community and the project is still active. It also supports the complete ID3 range, from v1 until v2.4. Furthermore it is also able to extract information out of .flac files, an important and frequently used standard of lossless audio files.

For a coding example we refer you to the appendix section 9.2, in this way you get acquainted with the methods used. You will quickly understand the methods followed to implement this ID3 tag extraction.

7.2 Streaming

Like mentioned in section 3.5, streaming will be used to transmit and receive a multimedia file. We sadly had no more time left to implement streaming, but we did the complete research. In this case next development steps are much easier. The Java library that needs to be used is found and code examples are given in section 3.5.
7.3 Documents

At this moment the focus of this thesis is relying on multimedia files, but this concept can be extended. Apart from sharing pictures, audio and video files, it would be a valuable expansion to add the capability to share documents in the local area network.

For example, when a couple of students are working together on a project. They will be working on separate computers, so the data that they are generating will not be shared. If they had a simple application that would make it possible to share the documents they created or adjusted with the others, it would save them a lot of time since nowadays the fastest solution is to send an e-mail or to set up a shared folder, but then everybody needs to have the user credentials and must be capable to connect to the shared folder.

Of course, implementing the capability of sharing documents relies upon the goals you want to achieve. At the moment of writing, the goals were set upon the development of sharing multimedia, so there is no need to share documents or other file types.
8 References


A novel approach to local multimedia sharing
9 Bibliography


http://www.britannica.com/EBchecked/topic/80540/broadcast-network


http://www.rbgrn.net/content/21-how-to-choose-dlna-media-server-windows-mac-os-x-or-linux


Nair, P. V. (2005). *Choosing the right programming language*. PIT Solutions India and Switzerland.


Shaner L.R., M. *Building the Right Networks for Business Performance*. IMD Inc.


10 Appendix

10.1 Class hierarchy of Java bindings for VLC media player

In this class hierarchy, you can find back all the objects that can be used when implementing a vlcj library in a project. The section about streaming included the use of some classes from this library. For more information about the different classes, we can refer you to the Javadoc available from the GoogleCode website of vlcj, which is also included in the bibliography.

- java.lang.Object
- uk.co.caprica.vlcj.player.AbstractMediaPlayer
- uk.co.caprica.vlcj.player.list.DefaultMediaListPlayer (implements uk.co.caprica.vlcj.player.list.MediaListPlayer)
- uk.co.caprica.vlcj.player.DefaultMediaPlayer (implements uk.co.caprica.vlcj.player.MediaPlayer)
  - uk.co.caprica.vlcj.player.direct.DefaultDirectMediaPlayer (implements uk.co.caprica.vlcj.player.direct.DirectMediaPlayer)
  - uk.co.caprica.vlcj.player.embedded.DefaultEmbeddedMediaPlayer (implements uk.co.caprica.vlcj.player.embedded.EmbeddedMediaPlayer)
  - uk.co.caprica.vlcj.player.headless.DefaultHeadlessMediaPlayer (implements uk.co.caprica.vlcj.player.headless.HeadlessMediaPlayer)
- uk.co.caprica.vlcj.player.AudioDevice
- uk.co.caprica.vlcj.player.AudioOutput
- uk.co.caprica.vlcj.mrl.BaseDvdMrl (implements uk.co.caprica.vlcj.mrl.Mrl)
- uk.co.caprica.vlcj.mrl.DvdMrl
- uk.co.caprica.vlcj.mrl.SimpleDvdMrl
- uk.co.caprica.vlcj.mrl.CdMrl (implements uk.co.caprica.vlcj.mrl.Mrl)
  - uk.co.caprica.vlcj.runtime.windows.WindowsCanvas
- java.awt.Canvas (implements javax.accessibility.Accessible)
  - uk.co.caprica.vlcj.runtime.windows.WindowsCanvas
- uk.co.caprica.vlcj.player.embedded.DefaultFullScreenStrategy (implements uk.co.caprica.vlcj.player.embedded.FullScreenStrategy)
- uk.co.caprica.vlcj.logger.DefaultLogMessageHandler (implements uk.co.caprica.vlcj.log.LogMessageHandler)
- uk.co.caprica.vlcj.filter.ExtensionFileFilter (implements java.io.FileFilter)
- uk.co.caprica.vlcj.filter.AudioFileFilter
- uk.co.caprica.vlcj.filter.PlayListFileFilter
- uk.co.caprica.vlcj.filter.SubTitleFileFilter
- uk.co.caprica.vlcj.filter.VideoFileFilter
- javax.swing.filechooser.FileFilter
- uk.co.caprica.vlcj.filter.swing.SwingFileFilter
- uk.co.caprica.vlcj.mrl.FileMrl (implements uk.co.caprica.vlcj.mrl.Mrl)
- uk.co.caprica.vlcj.binding.LibVlcFactory
A novel approach to local multimedia sharing

- uk.co.caprica.vlcj.version.LibVlcVersion
- uk.co.caprica.vlcj.runtime.x.LibXUtil
- uk.co.caprica.vlcj.player.embedded.videosurface.linux.LinuxVideoSurfaceAdapter (implements uk.co.caprica.vlcj.player.embedded.videosurface.VideoSurfaceAdapter)
- uk.co.caprica.vlcj.log.Log
- uk.co.caprica.vlcj.logger.Logger
- uk.co.caprica.vlcj.log.LogHandler
- uk.co.caprica.vlcj.log.LogMessage
- uk.co.caprica.vlcj.player.embedded.videosurface.mac.MacVideoSurfaceAdapter (implements uk.co.caprica.vlcj.player.embedded.videosurface.VideoSurfaceAdapter)
- uk.co.caprica.vlcj.log.matcher.MatcherLogMessageHandler (implements uk.co.caprica.vlcj.log.LogMessageHandler)
- uk.co.caprica.vlcj.player.MediaDetails (implements java.io.Serializable)
- uk.co.caprica.vlcj.filter.MediaFileFilter (implements java.io.FileFilter)
- uk.co.caprica.vlcj.player.list.MediaList
- uk.co.caprica.vlcj.player.list.MediaListPlayerEventListener
- uk.co.caprica.vlcj.player.list.events.MediaListPlayerEventFactory
- uk.co.caprica.vlcj.player.MediaMeta
- uk.co.caprica.vlcj.player.MediaPlayerEventListener
- uk.co.caprica.vlcj.player.MediaPlayerFactory
- uk.co.caprica.vlcj.player.MediaPlayerLatch
- uk.co.caprica.vlcj.player.ModuleDescription
- com.sun.jna.PointerType (implements com.sun.jna.NativeMapped)
- uk.co.caprica.vlcj.binding.internal.libvlc_event_manager_t
- uk.co.caprica.vlcj.binding.internal.libvlc_instance_t
- uk.co.caprica.vlcj.binding.internal.libvlc_log_iterator_t
- uk.co.caprica.vlcj.binding.internal.libvlc_log_t
- uk.co.caprica.vlcj.binding.internal.libvlc_media_list_player_t
- uk.co.caprica.vlcj.binding.internal.libvlc_media_list_t
- uk.co.caprica.vlcj.binding.internal.libvlc_media_player_t
- uk.co.caprica.vlcj.binding.internal.libvlc_media_t
- uk.co.caprica.vlcj.player.direct.RenderCallbackAdapter (implements uk.co.caprica.vlcj.player.direct.RenderCallback)
- uk.co.caprica.vlcj.mrl.RtpMrl (implements uk.co.caprica.vlcj.mrl.Mrl)
- uk.co.caprica.vlcj.runtime.RuntimeUtil
- uk.co.caprica.vlcj.mrl.SsmMrl (implements uk.co.caprica.vlcj.mrl.Mrl)
- com.sun.jna.Structure
- uk.co.caprica.vlcj.binding.internal.libvlc_audio_output_t
Appendix

  - uk.co.caprica.vlcj.binding.internal.libvlc_event_t
  - uk.co.caprica.vlcj.binding.internal.libvlc_log_message_t
  - uk.co.caprica.vlcj.binding.internal.libvlc_media_stats_t
  - uk.co.caprica.vlcj.binding.internal.libvlc_media_track_info_audio_t
  - uk.co.caprica.vlcj.binding.internal.libvlc_media_track_info_t
  - uk.co.caprica.vlcj.binding.internal.libvlc_media_track_info_video_t
  - uk.co.caprica.vlcj.binding.internal.libvlc_module_description_t
  - uk.co.caprica.vlcj.binding.internal.libvlc_rectangle_t
  - uk.co.caprica.vlcj.binding.internal.libvlc_track_description_t
  - uk.co.caprica.vlcj.binding.internal.media_duration_changed
  - uk.co.caprica.vlcj.binding.internal.media_freed
  - uk.co.caprica.vlcj.binding.internal.media_list_item_added
  - uk.co.caprica.vlcj.binding.internal.media_list_item_deleted
  - uk.co.caprica.vlcj.binding.internal.media_list_player_next_item_set
  - uk.co.caprica.vlcj.binding.internal.media_list_will_add_item
  - uk.co.caprica.vlcj.binding.internal.media_list_will_delete_item
  - uk.co.caprica.vlcj.binding.internal.media_meta_changed
  - uk.co.caprica.vlcj.binding.internal.media_parsed_changed
  - uk.co.caprica.vlcj.binding.internal.media_player_length_changed
  - uk.co.caprica.vlcj.binding.internal.media_player_media_changed
  - uk.co.caprica.vlcj.binding.internal.media_player_pausable_changed
  - uk.co.caprica.vlcj.binding.internal.media_player_position_changed
  - uk.co.caprica.vlcj.binding.internal.media_player_seekable_changed
  - uk.co.caprica.vlcj.binding.internal.media_player_snapshot_taken
  - uk.co.caprica.vlcj.binding.internal.media_player_time_changed
  - uk.co.caprica.vlcj.binding.internal.media_player_title_changed
  - uk.co.caprica.vlcj.binding.internal.media_preparsed_changed
  - uk.co.caprica.vlcj.binding.internal.media_state_changed
  - uk.co.caprica.vlcj.binding.internal.media_subitem_added
  - uk.co.caprica.vlcj.runtime.windows.internal.MSLLHOOKSTRUCT
    - uk.co.caprica.vlcj.runtime.windows.internal.MSLLHOOKSTRUCT.ByReference (implements com.sun.jna.Structure.ByReference)
  - com.sun.jna.Union
    - uk.co.caprica.vlcj.binding.internal.libvlc_event_u
    - uk.co.caprica.vlcj.binding.internal.libvlc_event_u.ByValue (implements com.sun.jna.Structure.ByValue)
A novel approach to local multimedia sharing

- uk.co.caprica.vlcj.binding.internal.libvlc_media_track_info_u
- uk.co.caprica.vlcj.binding.internal.libvlc_media_track_info_u.ByValue (implements com.sun.jna.Structure.ByValue)
- uk.co.caprica.vlcj.binding.internal.vlm_media_event
- uk.co.caprica.vlcj.filter.swing.SwingFileFilterFactory
- uk.co.caprica.vlcj.player.TrackDescription (implements java.io.Serializable)
- uk.co.caprica.vlcj.player.TrackInfo
- uk.co.caprica.vlcj.player.AudioTrackInfo
- uk.co.caprica.vlcj.player.SpuTrackInfo
- uk.co.caprica.vlcj.player.UnknownTrackInfo
- uk.co.caprica.vlcj.player.VideoTrackInfo
- uk.co.caprica.vlcj.mrl.UdpMrl (implements uk.co.caprica.vlcj.mrl.Mrl)
- uk.co.caprica.vlcj.mrl.VcdMrl (implements uk.co.caprica.vlcj.mrl.Mrl)
- uk.co.caprica.vlcj.version.Version (implements java.lang.Comparable<T>)
- uk.co.caprica.vlcj.player.VideoOutputLatch
- uk.co.caprica.vlcj.player.embedded.videosurface.VideoSurface (implements java.io.Serializable)
- uk.co.caprica.vlcj.player.embedded.videosurface.CanvasVideoSurface
- uk.co.caprica.vlcj.player.embedded.videosurface.ComponentIdVideoSurface
- uk.co.caprica.vlcj.version.VlcjVersion
- uk.co.caprica.vlcj.mrl.WebMrl (implements uk.co.caprica.vlcj.mrl.Mrl)
- uk.co.caprica.vlcj.mrl.FtpMrl
- uk.co.caprica.vlcj.mrl.HttpMrl
- uk.co.caprica.vlcj.mrl.MmsMrl
- uk.co.caprica.vlcj.mrl.RtspMrl
- uk.co.caprica.vlcj.runtime.windows.WindowsMouseHook (implements uk.co.caprica.vlcj.runtime.windows.internal.LowLevelMouseProc)
- uk.co.caprica.vlcj.runtime.windows.WindowsRuntimeUtil
- uk.co.caprica.vlcj.player.embedded.videosurface.windows.WindowsVideoSurfaceAdapter (implements uk.co.caprica.vlcj.player.embedded.videosurface.VideoSurfaceAdapter)
10.2 Coding example of ID3 tags with Jaudiotagger

Jaudiotagger is a Java library which can be downloaded from the internet. The library is well supported and has still an active community supporting it. This last fact is a very important issue to take into account when choosing the required library. When developing an application using an out-of-the-box library and you stumble upon problems, you need to have a community to support your problems. Besides the active community, Jaudiotagger also supports a wide range of music tags, so when it is desired to implement other music tags, the possibility exists. The whole range of ID3 tags is supported to, because there exists at the moment of writing 4 different versions. As last advantage Jaudiotagger provides an easy to understand programming structure, this will be clear after the overview of a coding example for extracting ID3 tags out of an .MP3 file.

To implement Jaudiotagger, you will first need to import the following classes after downloading the required library.

```java
import java.io.File;
import org.jaudiotagger.audio.AudioFile;
import org.jaudiotagger.audio.AudioFileIO;
import org.jaudiotagger.audio.AudioHeader;
import org.jaudiotagger.tag.FieldKey;
import org.jaudiotagger.tag.Tag;
```

The following examples refer to all tag formats supported by Jaudiotagger. The stuff you might want to know, but you cannot change, is stored in the AudioHeader class, the meta information that you can change is stored in the Tag interface.

```java
AudioFile f = AudioFileIO.read(testFile);
Tag tag = f.getTag();
AudioHeader = f.getAudioHeader();
```

The class AudioHeader includes:

```java
f.getAudioHeader().getTrackLength());
f.getAudioHeader().getSampleRateAsNumber());
```

The developers try to provide a common API for all tag formats, the FieldKey enum lists all fields that can be mapped to any of the fully supported formats. A full list of how fields are mapped to the underlying format is available on the website of Jaudiotagger. We can display the first value of a field using getFirst() , here are some examples:

```java
tag.getFirst(FieldKey.ARTIST);
tag.getFirst(FieldKey.ALBUM);
tag.getFirst(FieldKey.TITLE);
tag.getFirst(FieldKey.COMMENT);
tag.getFirst(FieldKey.YEAR);
tag.getFirst(FieldKey.TRACK);
tag.getFirst(FieldKey.DISC_NO);
tag.getFirst(FieldKey.COMPOSER);
tag.getFirst(FieldKey.ARTIST_SORT)
```

FieldKey contains full support for all Musicbrainz identifiers, such as

```java
ntag.getFirst(FieldKey.MUSICIP_ID);
tag.getFirst(FieldKey.MUSICBRAINZ_TRACK_ID);
tag.getFirst(FieldKey.MUSICBRAINZ_ARTISTID);
tag.getFirst(FieldKey.MUSICBRAINZ_RELEASEARTISTID);
tag.getFirst(FieldKey.MUSICBRAINZ_RELEASEID);
```

You do not have to worry about complications with different formats, for example there are two fields TRACK and TRACK_TOTAL for storing the track number on a release, and the
total tracks on a release. MP3 and MP4 formats stored both these values within a single field whereas WMA, Flac and OggVorbis store them as two separate fields, Jaudiotagger takes care of this behind the scenes and you can just treat it as two separate fields, whatever format you are reading or writing to.

If the format supports multiple items with the same id you can retrieve all items with a particular id using:

```java
List list = tag.get(FieldKey.ARTIST);
for(TagField field:list)
{
    //do something with the value of field
}
```

To retrieve all items in a tag use the following code:

```java
Iterator iterator = tag.getFields();
while(iterator.hasNext())
{
    iterator.next(); //do something using this iterator value
}
```

The examples above assume textual data, but what about binary data? You can retrieve the complete tag field instead of just a String representation of its value by using `getFirstField()`

```java
TagField binaryField = tag.getFirstField(FieldKey.COVER_ART));
```