DESIGN AND IMPLEMENTATION OF A VIDEO SESSION MIGRATION SYSTEM

Magisterarbeit

zur Erlangung des akademischen Grades Diplom-Ingenieur

Studium: Informatik

Universität Klagenfurt
Fakultät für Wirtschaftswissenschaften und Informatik

Begutachter: Univ.-Prof. Dipl.-Ing. Dr. Hermann Hellwagner
Institut: Institut für Informationstechnologie

February/2005
Ehrenwörtliche Erklärung


Unterschrift:

Klagenfurt, 13. Februar 2005

Word of Honour

I honestly declare that the thesis at hand and all its directly accompanying work have been done by myself. Permission has been obtained for the use of any copyrighted material appearing in this thesis and all such use is clearly acknowledged. The thesis has not been presented to any other examination board.

Unterschrift:

Klagenfurt, 13. Februar 2005
Ich widme diese Arbeit meiner Familie sowie meiner Freundin und all meinen Freunden. Im ganz Besonderen aber möchte ich diese Arbeit meiner Mutter widmen, die mir meine Ausbildung ermöglicht und mich dabei immer unterstützt und gefördert hat.
I Background of Video Session Migration

1 Introduction
  1.1 Some Use Cases for Video Session Migration
     1.1.1 Environment Change of a Mobile User
     1.1.2 VoD Learning Environment
     1.1.3 University Course
     1.1.4 Digital Surveillance System
     1.1.5 Changing Network Characteristics
  1.2 Rationale for Video Session Migration
  1.3 Design Considerations
     1.3.1 Originator of a Video Session Migration
     1.3.2 Where to Put the Business Logic?
     1.3.3 Conclusion
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.7</td>
<td>The SeMiProxy</td>
<td>93</td>
</tr>
<tr>
<td>7.7.1</td>
<td>Profiles</td>
<td>93</td>
</tr>
<tr>
<td>7.7.2</td>
<td>Destination of RTP Data</td>
<td>94</td>
</tr>
<tr>
<td>7.7.3</td>
<td>Support of Non RTSP Proxy Capable Media Players</td>
<td>95</td>
</tr>
<tr>
<td>7.7.4</td>
<td>Media Time Management</td>
<td>95</td>
</tr>
<tr>
<td>7.7.5</td>
<td>Interface to the SeMiServer</td>
<td>99</td>
</tr>
<tr>
<td>7.8</td>
<td>Classes, Files and Programs</td>
<td>100</td>
</tr>
<tr>
<td>7.8.1</td>
<td>Classes used by the SeMiServer</td>
<td>100</td>
</tr>
<tr>
<td>7.8.2</td>
<td>Classes used by the SeMiProxy</td>
<td>102</td>
</tr>
<tr>
<td>7.8.3</td>
<td>Shared Classes</td>
<td>104</td>
</tr>
<tr>
<td>7.8.4</td>
<td>Programs and Files</td>
<td>107</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion and Future Work</td>
<td>108</td>
</tr>
<tr>
<td>A</td>
<td>Terminology</td>
<td>110</td>
</tr>
<tr>
<td>B</td>
<td>Some Details about RTSP Header Fields</td>
<td>115</td>
</tr>
<tr>
<td>C</td>
<td>The Session Description Protocol (SDP)</td>
<td>118</td>
</tr>
<tr>
<td>C.1</td>
<td>Introduction</td>
<td>118</td>
</tr>
<tr>
<td>C.2</td>
<td>Specification</td>
<td>118</td>
</tr>
<tr>
<td>C.3</td>
<td>SDP Example</td>
<td>124</td>
</tr>
<tr>
<td>D</td>
<td>Video Content Management</td>
<td>125</td>
</tr>
<tr>
<td>D.1</td>
<td>The Video List Conforming to MPEG-21 DID</td>
<td>125</td>
</tr>
<tr>
<td>D.2</td>
<td>The XML Style Sheet for the Video List</td>
<td>127</td>
</tr>
<tr>
<td>E</td>
<td>How to Install Linux on a PDA?</td>
<td>130</td>
</tr>
<tr>
<td>F</td>
<td>Media Time Estimation - Source Code</td>
<td>134</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>137</td>
</tr>
</tbody>
</table>
# List of Tables

3.1 RTSP message types (according to [1]) .................................................. 26

6.1 Possible parameters of an RTSP-URI, added by the SeMiServer ... 76

7.1 Example of RTSP-URIs used for active sessions ......................... 92

B.1 Syntax of an SMPTE time range ......................................................... 116
B.2 Syntax of an NPT time range .......................................................... 116
B.3 Syntax of an ISO 8601 time stamp .................................................. 116
B.4 Parameters of the Transport header field of RTSP ....................... 117

C.1 SDP type specifiers [2] ................................................................. 119
List of Figures

1.1 Initial situation of a video session migration ................................. 7
1.2 Objective of a video session migration ........................................... 8

2.1 Screenshots of the MPEG-21 Session Mobility Tool [3] ....................... 12

3.1 Unicast transmission (taken from [4]) ............................................ 19
3.2 Multicast transmission (taken from [4]) ......................................... 19
3.3 Message format of RTSP request messages ...................................... 27
3.4 Message format of RTSP response messages .................................... 27
3.5 RTSP-URI syntax ........................................................................... 28
3.6 State machine of an RTSP server (according to [1]) .......................... 29
3.7 Example of an RTSP OPTIONS message ......................................... 30
3.8 Example of an RTSP DESCRIBE message ........................................ 31
3.9 Example of an RTSP ANNOUNCE message ...................................... 32
3.10 Example of an RTSP SETUP message ............................................ 33
3.11 Example of an RTSP PLAY message ............................................... 33
3.12 Example of pausing and resuming an RTSP session ........................... 34
3.13 Example of an RTSP TEARDOWN message ..................................... 35
3.14 Example of an RTSP GET_PARAMETER message ............................. 35
3.15 Example of an RTSP SET_PARAMETER message .............................. 36
3.16 Example of an RTSP REDIRECT message ...................................... 37
3.17 Example of a DIDL document ....................................................... 42
3.18 Concept of Digital Item Adaptation (DIA) (taken from [5]) ............... 44
7.7 Shared classes (Part-1) ................................. 105
7.8 Shared classes (Part-2) ................................. 106
C.1 Syntax of the origin specifier of SDP (SP=space) .............. 120
C.2 Syntax of the connection data specifier of SDP (SP=space) .... 121
C.3 Syntax of the bandwidth specifier of SDP .......................... 121
C.4 Syntax of the session start/stop time specifier of SDP (SP=space) 122
C.5 Syntax of the repeat times specifier of SDP (SP=space) ........ 122
C.6 Syntax of the encryption keys specifier of SDP .................... 122
C.7 Syntax of the attributes specifier of SDP .......................... 123
C.8 Syntax of the media announcements specifier of SDP ............ 123
C.9 Example of a SDP file ...................................... 124
Danksagung


Acknowledgements

I would like to thank Univ.-Prof. Dipl.-Ing. Dr. Hermann Hellwagner for his supervision and his support with this thesis. Special thanks go to Dipl.-Ing. Dr. Michael Kropfberger for his support, his numerous ideas, and for all the intensive discussions. I am also grateful for the support of Dipl.-Ing. Christian Timmerer in the area of MPEG-21. Finally, I would like to thank all proof-readers of this thesis.
Kurzfassung

Abstract

Today streaming media has become a well known technology which can be used to transmit continuous media data over a computer network. A very popular example of streaming media is radio-streaming, which allows listening to a certain radio station from almost all over the world via the Internet. The emerging growth of new network technologies (e.g. DSL, UMTS, WLAN etc.) allow even wireless transmission of compressed video data. Due to the development of powerful mobile devices, another application of streaming media that is called Video on Demand (VoD) becomes more and more important. VoD enables users to decide when they want to watch a certain video. Moreover, due to the availability of high-speed mobile networks, end-users are no longer constrained to use fixed and wired computers for VoD. This thesis covers the issue of "Video Session Migration" which is, more precisely, the migration of an already established video stream (i.e., a video session) from its original device to another device. That should enable a mobile user of VoD to change the receiving device of a session if the current environment offers, e.g., a more advanced (e.g. with a better display), compatible device. The proposed system uses common technologies and standards (e.g. RTSP and RTP/RTCP) for that purpose and, thus, can be used with every media player that supports streaming media with RTSP and satisfies some minor additional requirements, as discussed in this thesis.
Part I

Background of Video Session
Migration
The aim of this thesis is to design and to implement a video session migration system. Such a system can be used to migrate or copy a video which is streamed over a network, directly consumed at the client side, from the currently receiving device (device-A) to any other capable device (device-B). After the migration, the video session is continued on device-B with changed settings compatible with the hardware capabilities of the new device. The video session may be continued from the current media time position (i.e., its latest media time on device-A) or from a previously marked position. Afterwards, the video stream may be switched back to the original device or may be further migrated/copied to any other capable device. In contrast to a copied session, a migrated session is closed on the original device after the migration process.

Such a system enables some new possibilities in the area of streaming media:

- A user can continue to watch a video on any other streaming-capable device. The system will consider the changed hardware properties (e.g. screen size).

- Used with a Video on Demand (VoD) system, a user can mark positions within the video in order to restart it from any of these marked positions later. The video can be restarted on the same device or on any other device with different hardware capabilities.
Based on these possibilities, Section 1.1 describes some use cases for a practical application of a video session migration system.

The Session Migration System (SeMiSystem) should accomplish the management of available videos and available devices which can be used for migration. Furthermore, it is responsible for the adaptation of a migrating video stream, to meet the hardware configuration of that new device. It might, for instance, be necessary to change the resolution of a video in order to fit to the capabilities of a PDA (e.g. the screen size). Of course, this task could also be accomplished by the client application which is responsible for the playback of the video. However, this way would waste valuable network bandwidth and computing power of the client. In addition, the session migration system would have to estimate the media time of a migrating session (or request it from the client, if possible).

An important goal of this thesis is to use the resulting SeMiSystem in combination with the already existing ViTooKi system [7, 8] from the ADMITS project [9]. This system allows to transmit MPEG-4 videos over a network by using the Real-Time Streaming Protocol (RTSP) and the Real-time Transport Protocol (RTP/RTCP). Furthermore, it is able to perform a quality adaptation of a video stream. This could be a static adaptation (e.g. transcoding) or a dynamic adaptation which is based on the RTCP feedback from the client (e.g. frame dropping). The SeMiSystem should enhance that system to enable migration for an adaptive video stream. The main focus lies on the migration of a video stream from a PC (or Laptop) to a mobile device (e.g. a PDA) attached to a wireless network (WLAN).

**Organization of this Thesis**

This thesis is organized as follows: Section 1.1 starts with some use cases for a video session migration, followed by a description of the rationale of such a
migration. Afterwards, Chapter 2 presents the analysis of related work. In addition to a detailed introduction into streaming media and a brief overview of the Real-time Transport Protocol (RTP), Chapter 3 provides a detailed explanation of the Real-Time Streaming Protocol (RTSP), which is very important for the further understanding of the SeMiSystem architecture. Other important parts used by the SeMiSystem are MPEG-21 Digital Item Declaration and Digital Item Adaptation, which will be described in Chapter 3. Chapter 4 continues with a comparison of features of some popular streaming servers and introduces the requirements for a streaming server in order to be used with the proposed SeMiSystem. Chapter 5 starts with a number of requirements for a player in order to be used with our SeMiSystem. As the migration to a PDA was an especially important goal of this work, Chapter 5 presents the results of an extensive evaluation of media players for PDAs. The architecture and implementation of the SeMiSystem is explained in Chapter 6 (general architecture) and Chapter 7. Finally, Chapter 8 closes with a summary and a conclusion of this thesis and presents some ideas for future work.

1.1 Some Use Cases for Video Session Migration

This section contains some practical use cases, enabled by the SeMiSystem. While the first three use cases consider the operation in a Video on Demand (VoD) system, the fourth use case treats the employment within a live streaming system. Finally, the fifth use case deals with changing network characteristics.

1.1.1 Environment Change of a Mobile User

Consider the scenario in which a mobile user, who is watching a streamed video (e.g. a news video from a VoD system) on a mobile device, enters an environment which allows a more sophisticated playback (a larger display, for instance) of the streamed video. Using the SeMiSystem the user can easily continue watching the video on that sophisticated device.
1.1.2 VoD Learning Environment

Imagine a VoD learning environment, in which students can study by watching educational content from a streaming server. Let us assume that a student, working with a PC attached to the campus network, starts watching a streamed video. Using the SeMiSystem, the student can:

1. Pause the streaming of the video in order to resume learning on another PC which has access to the campus network as well (maybe on a PC at home or in the student dorms).

2. Switch the streaming from the PC to a mobile device (e.g. a PDA) and continue to watch the video on the way home (assumed the mobile device has access to the campus network).

3. Mark positions within the video in order to repeat some parts of it at a later time. The student will be able to start exactly from a selected position and will not have to restart the entire streaming from the beginning.

1.1.3 University Course

Imagine a practical course at a university which requires the students to use a PC in order to practice some topics of the course. An example would be a programming course in which the course leader has prepared a streaming video for the lecture, from which some scenes should be presented to the students. Using the SeMiSystem, it would be possible to allow the students to copy a video session to their own PCs and to watch it from marked positions.

1.1.4 Digital Surveillance System

This use case treats a live broadcasting system. Please note that in a live broadcasting system the recorded content is normally not permanently stored on the server but rather directly forwarded to the clients. Thus, the streaming of a live broadcast
can normally not be restarted from an earlier position. Hence, several features of the SeMiSystem cannot be used with a true live broadcasting system. However, the SeMiSystem can be used at least for changing the client device of the streaming with regard to the changed hardware capabilities.

Imagine a digital surveillance system, where a surveillance camera streams its content to the PC of a watchman (supervising this content). Let us assume he has to go onto an inspection walkway. Using the SeMiSystem, the watchman may easily switch the live streamed video from his high resolution colour monitor to a small grey-scale PDA in order to continue to supervise the content of the camera on his way.

1.1.5 Changing Network Characteristics

Another application of the SeMiSystem would be a situation where the network performance degrades significantly. Assuming a user has started a video streaming session for a high resolution video on his mobile device which is connected to a wireless LAN. If the network load increases and the playback of the streamed video cannot be continued with the current resolution due to less available bandwidth, the user could easily switch to a lower resolution (or to a gray-scaled representation) by using the SeMiSystem.

1.2 Rationale for Video Session Migration

The basic principle of migrating a video session is to create a copy of the media stream for the destination device and afterwards close the media stream on the originating device. Figures 1.1 and 1.2 illustrate the objective of a video session migration. There are three important issues which have to be considered when migrating a video session.

1. The second media stream has to be continued at the correct media time. This has to be either the position on which the streaming to the original device has
been ended or a slightly earlier position. When migrating a video session, a user should not miss any content.

2. The hardware capabilities of the destination device should be considered. This means that a media stream might be transcoded to meet the hardware limitations of the device. Such limitations may be the display resolution, the colour capability, the framerate, the audio configuration (mono/stereo), the audio/video bitrate, or the network bandwidth. In addition to the hardware limitations, users should be able to choose the settings for these properties by themselves (User Preferences).

3. The rights of a user to migrate or copy a media stream to a particular device should also be considered. This is necessary because otherwise a user could be regularly disturbed by migration attempts of other users.

1.3 Design Considerations

1.3.1 Originator of a Video Session Migration

It is very important from which initial point the migration of a video session is started.
Figure 1.2: Objective of a video session migration

Basically, there are two possibilities:

1. The user starts the migration from the device that is currently receiving the media stream. In order to migrate the media stream to another device, the user must select a certain destination device.

2. The user starts the migration from the destination device by choosing an active media stream as the source for a migration. In other words, the user takes over (absorbs) an active media stream.

In both options, it is important that the user has the corresponding rights to perform a video session migration.

If a session migration system uses option-1, it has to provide a list to the user, which contains those devices that are ready to accept a media stream for that particular user. Only those devices should be contained in the list, for which the user has the permission to migrate or copy the media stream to. On the chosen device, there must be some piece of software that is responsible for initiating a media player on that device and cause it to request the migrated/copied media stream.
If a session migration system uses option-2, it has to provide a list of active video sessions to the user, from which the user can choose one to take over or to copy. Only those sessions should be contained in the list, for which the user has the right to take over or copy it.

1.3.2 Where to Put the Business Logic?

No matter which option from Subsection 1.3.1 is used by a system, it has to manage the related data. This data could be managed by a client software running on the concerned devices, and/or a server software. The software is responsible for:

- monitoring involved devices or active video sessions
- managing
  - the hardware capabilities and user preferences
  - the user rights for a migration/copy process
- carrying out the migration/copy process.

Although it would be possible to implement a session migration system which only uses the client software, it is more appropriate to use a client/server architecture. This will assure a consistent data management, enable a scalable system and facilitate an interoperable system. The more logic is put into the server program, the easier it is to support different client platforms.

1.3.3 Conclusion

Based on the design considerations mentioned above, we have decided to use the take over approach (option-2) for the SeMiSystem, and put as much business logic as possible into the server program. This will result in a very flexible system, applicable to a lot of different operating systems with different media players.
This chapter includes the analysis of already existing work in the area of video session migration.

2.1 MPEG-21 Session Mobility Tool

The MPEG-21 Session Mobility Tool [3] was developed by the Multimedia Lab of the Ghent University in 2003. It is a software tool that enables the transfer of a multimedia streaming session to another device. It was submitted to become part of the MPEG-21 Digital Item Adaptation (DIA, see Section 3.3.3) Reference Software.

The Session Mobility Tool is able to load a Digital Item Declaration (DID, see Section 3.3.2) file from a network resource, specified by a URL. Then, the software parses the Choices in this file and provides a Selection to the user. After a valid selection, the Session Mobility Tool can be used to request the related resource from the network and start the playback process. The user can now choose another device running the Session Mobility Tool to transfer the current session onto that device. It uses a web service for the discovery of other devices useable as destination of a session transfer. For the transfer of a session, the resource identifier plus the current state of the session, i.e., the playback state (playing, paused, stopped) and the current media time are written into a Digital Item Declaration file conforming to the MPEG-21 Session Mobility syntax which is afterwards transmitted to the destination device.
When this file has been transmitted successfully, the session is closed on the original device. The Session Mobility Tool does not check the successful establishment of the session on the destination device. When the destination device receives the session description, it immediately tries to restore the session. For this, it uses the specified resource identifier and media time in order to request the media stream. Figure 2.1 contains some screenshots of the Session Mobility Tool.

The Session Mobility Tool was developed using the Microsoft .NET and .NET Compact Framework. It is available for the Microsoft Windows and the Microsoft Pocket PC operating system for Handheld PDAs. Thus, the tool can also be used to transfer sessions between PCs and PDAs. However, when a session is migrated to another device, the media stream is not adapted to the hardware capabilities of the new device. Besides, the Session Mobility Tool has no user management and does not take care of migration permissions. For the playback of the multimedia content, the Session Mobility Tool uses the Windows Media Player Control, which enables the embedded usage of the Windows Media Player within applications. This limits the tool to the features of the Windows Media Player. Currently, only Windows Media files can be directly used, because the Windows Media Player does not natively support streaming of other formats like MPEG-4. However, those formats may be converted to the Windows Media Format on the media server (see Subsection 4.2.2).

More information about the MPEG-21 Session Mobility Tool can be found in [3].
Figure 2.1: Screenshots of the MPEG-21 Session Mobility Tool [3]
2.2 Video Session Migration using Thin Clients

Another possibility of performing a video session migration is the use of a thin client. A thin client is a device (or sometimes a software tool) which allows to work on a remote machine (a host) connected by a network. Each mouse- or keyboard interaction is forwarded to the host. Computations are executed on the host and their output is rerouted to the thin client’s display. Thin clients use specific protocols for that purpose. Examples of such Remote Display Protocols are the Remote Framebuffer protocol (RFB\textsuperscript{1}) [10], the Remote Desktop Protocol (RDP\textsuperscript{2}) [11], or the X11 Protocol [12].

Beside of others, thin client have two important advantages:

1. Due to the execution of computations on the host, the hardware requirements of a thin client are very low.

2. Users can logon to the host from wherever they want. When a user logs off, the active programs (i.e., the session of the user) remain running on the host.

Thus, thin clients can also be used to continue a video streaming session on another machine by simply changing the displaying machine. However, this is no true migration of a video streaming session. Moreover, thin clients were designed to transmit static screen images rather than dynamic content of a video. The compression methods used by thin clients are not perfectly suitable for videos. Thus, users of thin clients will not experience a video quality which is similar to a truly streamed video. More information about thin clients can be found in [13].

\textsuperscript{1}RFB is used by the VNC application.
\textsuperscript{2}RDP is used by the Microsoft Terminal Server and the Microsoft Terminal Client.
This chapter provides a detailed description of those topics and technologies which have been used to design and develop the video session migration system. More precisely, this includes an introduction to streaming media, a brief overview of the Real-time Transport Protocol (RTP), an explanation of the Real-Time Streaming Protocol (RTSP) and a description of some parts of the MPEG-21 standard.

A reader who is already familiar with these topics can easily skip this chapter and continue on page 50 with Chapter 4.

### 3.1 Streaming Media Technology

This section introduces the concept of streaming media. It gives a short introduction into the aim of streaming media and explains the need for such a technology. In addition, it describes the general aspect of a streaming media server and a streaming media player. However, this section does not supply a comparison of several different streaming media servers or players. Please see Section 4 and 5 for such a comparison. Details can be found in [14, 15, 16].

#### 3.1.1 What is Streaming Media?

Streaming media is the transmission of media data over a computer network in a continuous flow, which can be immediately consumed by a client device without
storing the complete media data on a hard disk before playback. The transferred media data can be text, image, video, audio, or a combination thereof. In most cases streaming media is used to transfer video or audio files over a public network like the Internet. Streaming text can be used as subtitles; streaming images can be used to broadcast slide shows. For the remainder of this document, the term *streaming* refers to the streaming of a video over the Internet.

### 3.1.2 Application Areas of Streaming Media

Nowadays, there exist a number of application areas for streaming media. A very popular example is the huge amount of Internet-Radio stations, which use streaming for broadcasting their music. Furthermore, there is an increasing availability of Video on Demand (VoD) offerings, frequently used by publishing companies (e.g. magazine publishers), for instance, in order to enhance their service offerings to their clients. Streaming offers the great new possibility to provide a live transmission of digitally recorded media. With the common download-and-play approach this would not be possible. Thus, streaming can also be used for live broadcasting systems like a digital surveillance system, for instance, one that can stream the recorded images to one or several destination devices. In combination with wireless networks and the Internet, this enables new possibilities and applications in the surveillance area. Imagine being on vacation and having the possibility to watch what is going on in your house at home, from the other side of the world! Another great application of live streaming would be the transmission of an executive speech or company meeting to their employees or cooperative partners, located in other parts of the world. There are also some other areas, where streaming can be used, like in the medical area for instance, where video cameras are used for examinations and surgeries. Another example would be a distance learning environment, where streaming media can be used to provide the teaching content in the form of a video to any student on almost any device. However, I think the most interesting application is VoD where a selection of high quality movies is offered to a certain group of users. They can request the
delivery of a movie at any time on any place. Maybe one day such systems will supplement or supersede the good old cinemas?

### 3.1.3 Benefits of Streaming Media

In comparison with the download-and-play approach, streaming media provides a lot of advantages, which are summarized in this subsection.

- The data stream of the media is immediately consumed and not stored on a permanent storage media. Thus, it is possible to receive streaming media on devices that are limited in storage space (like a PDA or a mobile phone) as well.

- The aforementioned immediate consumption of the streaming media (without permanent storage) reduces the possibility of copyright violations. The number of times a movie is consumed can be measured and controlled by the server. Under certain conditions (e.g. on embedded systems where the use of a stream-recording application can be avoided) this could enable the implementation of a Digital Rights Management (DRM) system.

- The streaming media can be adapted to the capacities of a network and the capabilities of a client. Concerning movies, this may include the audio/video bitrate, the resolution, the framerate, the colour depth, and some audio-based properties like stereo/mono, or the sampling rate. This adaptation can be achieved by holding a number of different versions of the media file on the server. Another way that does not require storing several versions of a file, would be a real-time transcoding process supported by the media server. The ViTooKi MuViServer [7, 8] supports such a real-time transcoding.

- Moreover, the user does not have to wait for the playback until the download of the complete movie has been finished.

- Another advantage of streaming media is the reachability and the ease of use. A content provider can reach a large number of potential clients at a relatively
little expense (e.g. world-wide streaming radio vs. regional radio station). On the other hand, a user only needs the access to the network and a player which can play the streamed media in order to join the group of receivers.

### 3.1.4 Streaming vs. Downloading

Of course, the transmission of a digital video over a network (not a live broadcast) can also be accomplished by the common approach of download-and-play. But in this way, the playback of the video can only be started when the transmission of the entire video has been finished. It does not provide any feedback or interaction while the data is being downloaded. This is a big disadvantage when thinking of the situation where a user has chosen the wrong movie. In such a situation a lot of time, money and network bandwidth is being wasted. In addition, download-and-play cannot be used for streaming live-content. Besides, it is not applicable for devices, which have a very small amount of permanent storage space.

However, streaming is no replacement for the download-and-play approach. Often the network ports of an IP network, used for streaming, are blocked by firewalls. In addition, streaming may be affected by bad network performance. If some packets arrive out of time, this will either cause wrong display of some frames or, even worse, interrupt the playback. Using the common download approach, the packets will be retransmitted, as long as they arrive on the client. Thus, download-and-play is still used to offer high quality movies. By means of this approach, the playback quality is not constrained by the Internet connection speed of a client (that mostly constitutes the bottleneck). Besides, the movie can be easily copied and played, as often as the user wants (supposed there is no Digital Rights Management), although this could also be accomplished by a streaming system that uses a caching/recording functionality.
3.1.5  Progressive Download

There exists also a hybrid solution for the transmission of media files, which is called *Progressive Download*. It is similar to the download-and-play approach with the difference that the playback of the media file can be started before the entire file has been downloaded. With this approach, a user can play a media file while it is being downloaded. However, when the playback reaches the end of the currently received amount of data, the playback process must be halted until more data is available. Some media players have a so-called “HTTP streaming” functionality, which is, in fact, nothing else than progressive download.

The main difference between progressive download and true streaming is that streaming does normally not leave a copy of the received content in the memory of the receiving device. In addition, most implementations of progressive download can start the playback only from the beginning of the media file while true streaming solutions can start from any position within the media file.

3.1.6  Unicast vs. Multicast

When transmitting the same data to several receivers over a network like the Internet, two different techniques can be used for the transmission of the packets. The first one is called unicast, which is the approach of conveying every packet to each receiver separately. Figure 3.1 illustrates a unicast transmission for five clients.

As one might already imagine, this could be very inefficient when watching exactly the same content, especially when most of the receivers are located close to each other at the end of the path of transmission. In order to resolve this problem, another technique can be used, which is called multicast. In this way, each packet is only sent once from the sending host. Depending on the structure of the network and the position of the receivers, the packets may be copied (i.e., replicated) by routers on the way to the receivers through the network.

Figure 3.2 illustrates a multicast transmission for five clients. It should be obvious that multicast transmission could save a lot of bandwidth, especially for streaming
media. For multicasting, a predefined range of IP addresses (a so-called Class D network) is used. This is the range from 224.0.0.0 up to 239.255.255.255. In order to use multicast transmission, all routers on the network path must be multicast capable.

3.1.7 Network Aspects of Streaming Media

There are several aspects which have to be considered when using streaming over a public network like the Internet. The Internet was originally not designed to transmit real-time media. It is a best-effort network and does not provide any guarantees about the transmission. On the Internet, data is transmitted using data packets. When a

\[ \text{Please note: the range from 224.0.0.0 up to 224.0.0.255 is reserved and cannot be used!} \]
huge file needs to be transmitted, it must firstly be fragmented into a number of packets, which are sent separately through the network. At the receiving side, the packets have to be defragmented to reconstruct the original file. When a sub-network on the route of a sent packet gets congested, the packet may be discarded. Depending on the chosen transport protocol, the packet may be retransmitted or not. In many cases a retransmission of a packet will be essential, but when using streaming media, it might not be appropriate. Consider a situation in which the playback of a streamed movie has to be stopped, because the packet which is used for a small portion of the current frame must be retransmitted. A user would rather prefer to discard the single frame affected by the packet and continue the playback instead of waiting for that packet. This is the reason why streaming mostly uses the User Datagram Protocol (UDP) as a transport protocol instead of the Transport Control Protocol (TCP). In contrast to TCP, UDP does not perform a retransmission of lost packets.

In addition, no exact information about the transmission time of a single packet can be predicted. If there is a high load, the packet may be delayed in the network. Thus, consecutive packets of a stream may arrive in different intervals. The variation of these intervals is commonly known as Jitter. In order to reduce the impact of high jitter, streaming clients do not immediately start the playback of a received media stream. They rather use a Prefetch Buffer and start the playing process a few seconds later. The buffer acts for the application like a stock of data which compensates the jitter. However, it should be obvious that a very high jitter can cause the buffer to run out of data and, thus, to get the playback of the client application halted.

Another problem is the out-of-order delivery of packets. Because each packet can take its own route over the network, the sending order of the packets does not assure the same receiving order on the receiver.
3.1.8 The Real-time Transport Protocol (RTP)

To transmit streaming media over a network, an application layer transport protocol with real-time characteristics has to be used. Such a protocol is the Real-time Transport Protocol (RTP). It is an IETF standard which was initially published in January 1996 [17]. Over the years the standard was revised and was republished in July 2003 [18].

Basically, RTP is independent of the underlying transport and network layers. However, currently RTP is mostly used in combination with UDP over IP. When an unreliable protocol like UDP is used, applications themselves should care about out-of-order delivered, or lost packets. To deal with these problems, RTP packets contain timing information and sequence numbers. Furthermore, RTP uses an associated protocol named RTP Control Protocol (RTCP), which is used to provide additional information about an RTP session to both the sender and the receiver. For a sender, RTCP provides information about the current participants as well as feedback about the receiving quality. This information is carried in the Receiver Reports. RTCP also facilitates Sender Reports which are almost equal to receiver reports and can be used by receiving clients to get information about the streaming properties of the session. Sender and receiver reports contain information about the number of received packets, the number of lost packets and the achieved jitter.

With RTP it is possible to transmit various media formats which are defined in a set of profiles. A list of available profiles for RTP can be found in RFC 3551 [19] (or RFC 1890 [20]). RTP can either be used for a unicast transmission which is appropriate for an on-demand session, or it can be used for multicast transmission. If there are many receivers for a session which want to receive the same data, a multicast transmission should be used in order to save network bandwidth. Multicast transmission is a very common technology for live streaming sessions.
Using RTP, each media track is transmitted as a separate RTP session. Thus, a video containing both, video and audio, will be streamed in two RTP sessions. While even port numbers are used for the RTP data, the consecutive odd port numbers are used to transmit RTCP data for each session respectively. For more information about RTP and RTCP, please refer to RFC 3550 [18].

### 3.1.9 Components of a Streaming Media System

The two main components of a streaming media system are:

- a *Streaming Server*, and
- a *Streaming Player*.

The streaming server is responsible for the transmission of the media data. In a VoD system, the server holds a number of available movies and starts the transmission as soon as a client requests a movie. A streaming media server waits for incoming connections and starts the streaming process, after a valid request has been received. A very common way to request a movie is the use of the *Real-Time Streaming Protocol*, which is described in section 3.2. For the transmission of the media data, the streaming server uses a specially suited transport protocol - e.g. RTP. When performing a live broadcast, the streaming server also needs a capture device like a video camera. The recorded images of the capture device must be encoded to a digital video format first, which is used by the server as the source of the streaming process. Please note that the encoding time and the processing time of the server, as well as the transmission time and the decoding time on the client, can significantly affect the delay between the recording of an image on the server and its presentation on the client. Normally, there exist several connections between the server and the client, for one streaming session. At least one connection for the transmission of media data (e.g. RTP), one connection for the transmission of control information (e.g. RTSP) and one connection for the transmission of feedback information about the session.
(e.g. RTCP). These connections will be usually maintained over the entire duration of a streaming session.

A streaming server may store different versions of a movie to be able to provide the suitable format for various clients. A more advanced feature is on-demand transcoding on the server, which allows the server to provide almost any format of the movie, without the need of storing several versions. Besides, a server may provide other features like advertisement insertion, sex/violent scene removal, or Digital Rights Management.

At the client side, a media player which is capable of receiving streaming media has to be used. This might be either a stand-alone application, or a player which is integrated (“embedded”) into a web-browser using a certain plug-in. The player has to buffer the received packets, decode them, and display the resulting frames. For the decoding process, an appropriate Codec is required. In order to receive streaming media, the media player must be able to communicate with the streaming server. However, exactly this is the problem of various streaming servers and streaming players, because some of them do not use official standards but rather proprietary solutions or some extensions to a standard which are not compatible to other streaming systems.
3.2 The Real-Time Streaming Protocol (RTSP)

This section describes the Real-Time Streaming Protocol (RTSP), which is used by the SeMiProxy in order to perform a migration of a video session (see Chapter 6). The section starts with an introduction and afterwards explains the types of RTSP messages and their syntax. However, details about the header fields will be omitted in this section. Some important header fields are described in Appendix B. More information about the header fields and RTSP can be found in [1].

3.2.1 Foreword

The information of this section is mainly based on RFC 2326 [1], published in April 1998 which was submitted by the Multiparty Multimedia Session Control (MMUSIC) Working Group (WG) of the IETF in October 1996. Meanwhile, a lot of revision work in the form of Internet Drafts has been done on the original version. Anyway, these revisions have not yet reached IETF standards status. Therefore, they cannot be used as a reference here (see [21], p.8). For more information about the ongoing work of RTSP, please visit [22].

3.2.2 Introduction

RTSP is the abbreviation of the Real-Time Streaming Protocol which was published as an IETF standard in April 1998. Contrary to the name of this protocol, it is not intended to be used for streaming data itself but to control streaming sessions. This may include: initiation, pausing, resuming, halting, and recording sessions. The protocol can be considered as a network-based remote control for multimedia streaming sessions. It is used to start and stop a streaming session but it has nothing to do with the actual transmission of the media data.
Rather it provides the means:

- for creating and controlling an individual streaming session,
- for participating in a live-streaming session together with other clients, and
- for initiating the recording of a streaming session on a media recording server.

A streaming session controlled by RTSP might be either an on-demand streaming session or a live streaming session like a sport event.

With RTSP it is possible to control one or several media streams, for example, audio or video streams, which might also be synchronized in time. In order to know which streams are available for a particular movie, RTSP requires a presentation description of a movie. This description can either be placed in a file or on a website but can also be directly requested from a media server by an inherent mechanism of RTSP. In addition to a general description, the presentation description contains a list of media streams, information about the connection, which must be used to retrieve the streams, and the encoding format of a certain stream. In many cases, the Session Description Protocol (SDP) is used for such a presentation description. Appendix C will give an overview of SDP.

RTSP is completely independent of the protocol which is used to transfer media data and does not prescribe a specific protocol to use. RTSP is also independent of the lower-level transport protocol. Thus, it can either use UDP or TCP. Anyhow, the most common configuration of multimedia streaming uses RTSP over TCP/IP to control the streaming and RTP over UDP/IP to carry the media data. It is also possible to transmit both, the RTSP control data and the media data, via one transmission channel using interleaved binary data. In this way, the packets with the binary media data (e.g. the RTP packets) are encapsulated into the RTSP messages by using an additional header. Although this interleaving produces additional overhead and, thus, should be avoided if possible, it could be very useful for a client which cannot receive RTP data due to a firewall configuration. More information about interleaving binary data can be found in [1].
### 3.2.3 Protocol Details of RTSP

The Real-Time Streaming Protocol is an application-layer protocol which is purely text-based and which is similar to the *Hypertext Transfer Protocol* (HTTP) v1.1 [23]. It uses simple text messages, which are encoded using the ISO 10646 (UTF-8) [24] character set. Of course, using text messages is not very efficient, still it has some advantages: it is human-readable and can be easily extended in a self-describing manner. Besides, facing the small number of conveyed messages, efficiency is not a big problem. Similar to HTTP, RTSP messages may also be processed by proxies.

RTSP is using the client/server paradigm. It defines a number of message types, which are summarized in Table 3.1. As apparent in this table, only a few messages must be supported by a minimal RTSP server, others are optional. The default server port used for accepting incoming RTSP messages is 554. A client sends the RTSP messages to the server, in a valid order, and waits for response after each message.

<table>
<thead>
<tr>
<th>Method</th>
<th>C → S</th>
<th>S → C</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANNOUNCE</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>DESCRIBE</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GET_PARAMETER</td>
<td>x</td>
<td>x</td>
<td>recommended</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>OPTIONS</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>PAUSE</td>
<td>x</td>
<td></td>
<td>recommended</td>
</tr>
<tr>
<td>PLAY</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>RECORD</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REDIRECT</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>SETUP</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET_PARAMETER</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>TEARDOWN</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: RTSP message types (according to [1])

RTSP messages have a well-defined general syntax. Figure 3.3 shows the format of a request message. As apparent in the figure, it starts with a method name followed by a URI and the version number of RTSP. These fields are separated by a space
(SP). The header of the message follows after a CRLF\(^2\) and contains some Header Fields depending on the type of the message. Appendix B will explain a number of commonly used header fields. Header fields are separated by CRLF. If a message does also contain a body, it will be attached to the header after an additional CRLF.

<table>
<thead>
<tr>
<th>Method</th>
<th>SP</th>
<th>RTSP-URI</th>
<th>SP</th>
<th>RTSP-Version</th>
<th>CRLF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request-Message-Header</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRLF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Request-Message-Body (optional)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.3: Message format of RTSP request messages

Figure 3.4 shows the format of a response message. In contrast to the request message, the response message starts with the version number of RTSP followed by a status code and a reason phrase related to the last request message. Similar to the request message, the response message may contain a message body as well.

<table>
<thead>
<tr>
<th>RTSP-Version</th>
<th>SP</th>
<th>Status-Code</th>
<th>SP</th>
<th>Reason-Phrase</th>
<th>CRLF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response-Message-Header</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRLF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response-Message-Body (optional)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.4: Message format of RTSP response messages

As already mentioned, every request message must contain an RTSP-URI which identifies the movie on the server. There is one exception for the OPTIONS message, which should only contain an asterisk (*), instead of the RTSP-URI. This is due to the fact that the OPTIONS message does not refer to a particular stream or movie, but rather to the server itself.

Figure 3.5 shows the syntax of an RTSP-URI and illustrates some examples. The host-address can be specified as either an IP-Address or as a domain-name. The port is not required. If no port is specified, the default port of RTSP is used, which is 554. The movie has to be requested with the absolute file-path.

\(^2\)CRLF is the abbreviation of Carriage Return, Line Feed which means a line-break consisting of an ASCII-13 character followed by an ASCII-10 character.
RTSP does not require sustaining a connection over the entire duration of the streaming session. An RTSP client could open and close a connection with the server for every single message. Thus, RTSP messages contain session-based sequence numbers (see Appendix B). Nevertheless, in practice most implementations of RTSP clients use a permanent connection over the entire duration of a streaming session.

In contrast to HTTP, the communication messages of RTSP are session-aware. Thus, a particular message may cause a state change on the server. Therefore, an RTSP server has to manage a number of states for each session. Figure 3.6 shows the state machine for an RTSP server. The information about to which session a message belongs to, is contained in the message header, which can contain a *Session Id* (see Appendix B).

As apparent in Figure 3.6, the server starts with the state *Init*. When a SETUP message is received, the server allocates the necessary resources for the streaming process, creates a new session and sends back a response message to the client. After a valid PLAY message, the server switches to state *Playing* and immediately starts to stream the media data. While in playing state, the client may send a PAUSE message to suspend the streaming and continue with a PLAY after a while. Normally, the server does not free resources for the related streams while being suspended. PLAY messages may contain a range value which tells the server to stream only a portion of the media file. Furthermore, a client may send a number of subsequent PLAY messages with different ranges, which will be queued on the server. After playing the...
desired range(s) (of all received PLAY messages), the session will be paused on the server. However, if the client wants to stop the streaming session earlier, it has to send a TEARDOWN message. As a consequence, the server removes the session, frees resources associated with the session and switches back to state Init. If a client wants a server to record another streaming session, it can also send a RECORD message after the SETUP message, which will tell the server to store received packets of the specified session in a denoted URI. This is especially applicable for a live streaming session.

As further described in Subsection 3.2.4, a client may also send a SETUP message to a streaming or recording server for changing the transport parameters of a stream.
3.2.4 RTSP Messages

OPTIONS

The OPTIONS message is used to query the capabilities of an RTSP server and can be sent at any time. It does not affect the session state on the server. Figure 3.7 exemplifies an RTSP OPTIONS message. In the Public header of the response message, the server will return the allowed/implemented messages.

```
C → S:
  OPTIONS * RTSP/1.0
  CSeq: 1
S → C:
  RTSP/1.0 200 OK
  CSeq: 1
  Public: OPTIONS, DESCRIBE, SETUP, PLAY, PAUSE, TEARDOWN,
  GET_PARAMETER, SET_PARAMETER, REDIRECT
```

Figure 3.7: Example of an RTSP OPTIONS message

DESCRIBE

The DESCRIBE message is used to retrieve a presentation description about a certain media object. The client may announce the understandable formats in the Accept header field and the server will return a presentation description in any of the formats it can provide. The Session Description Protocol (SDP, see Appendix C) is a very commonly used protocol for that purpose. Please note that the DESCRIBE message is only required to provide the necessary information about the requested media object to the client and is actually not required to start a streaming session. The client could also get this information from any other source e.g. from a website, an email-attachment, or from a local SDP file. Figure 3.8 exemplifies a DESCRIBE message. In this example, the client does only accept a description according to the Session Description Protocol.
ANNOUNCE

With the ANNOUNCE message, it is possible to post a presentation description. This message can either be sent by a client or a server. Figure 3.9 shows an example of an ANNOUNCE message. Sent by a client, this can be used to provide information about a session to the server, which it should record afterwards (ANNOUNCE followed by RECORD). Sent by a server, this is used to update the client with the information about a session (e.g. when a new stream has been added to a live streaming session).
SETUP

The SETUP message is used to set the transport parameters for a streaming session. Figure 3.10 illustrates an example of a SETUP message. The transport parameters must be specified in the Transport header field (see Appendix B). This message must be sent before the PLAY message and will cause the server to allocate the required resources, which are necessary for the upcoming transmission. However, a server may also allow receiving SETUP messages while in state *Playing* or *Recording*. This would be useful for a client which wants to change the transport parameters during an active session.

After the first successful SETUP request of a session, the server has to return a new session identifier (Session Id) in the Session header field (see Appendix B) of the response message.
CHAPTER 3. ENABLING TECHNOLOGIES

Figure 3.10: Example of an RTSP SETUP message

PLAY

The PLAY message is causing the server to start the delivery of the media data for a particular session. Figure 3.11 shows an example of a PLAY message. The message presumes a successful previous SETUP message and may contain a time range in the Range header field (see Appendix B). If no time range is stated, the server will start streaming from the beginning until the end of the media object is reached or a PAUSE message is received.

It is also possible for a client to send a number of PLAY messages (with different time ranges). The server does not abort streaming for an unfinished time range if a subsequent PLAY arrives. It rather manages a queue for PLAY messages and executes them in receiving order.
PAUSE

The PAUSE message is used for temporarily stopping the streaming on the server without freeing resources. Figure 3.12 exemplifies a PAUSE message. A paused session can be resumed with a PLAY message. If there is no resume within the timeout period of the server, the server may close the session and free the corresponding resources. The PAUSE message can also contain a Range header field that will tell the server on which media time the session should be paused. If this field is missing, the session will be immediately halted when the message arrives. Queued PLAY messages on the server will be discarded when a PAUSE message arrives.

```
C → S:
PAUSE rtsp://localhost:554/mpeg4video.mp4 RTSP/1.0
CSeq: 5
Session: 988009202

S → C:
RTSP/1.0 200 OK
CSeq: 5
```

```
C → S:
PLAY rtsp://localhost:554/mpeg4video.mp4 RTSP/1.0
CSeq: 6
Range: npt=37.560000-519.640000
Session: 988009202

S → C:
RTSP/1.0 200 OK
CSeq: 5
```

Figure 3.12: Example of pausing and resuming an RTSP session

TEARDOWN

The TEARDOWN message is used to close a session on the server. Figure 3.13 shows an example of a TEARDOWN message. All corresponding resources on the server will be freed and the session will be removed.
GET_PARAMETER

The GET_PARAMETER message provides a means to request parameters from the server. Usually, this message is used together with a message body which contains a list of requested parameters. In the response message, the server will return a list of related values for each recognized parameter. The content included in the message body (and the content-type field as well) depends on the implementation. However, the GET_PARAMETER request may also be used without a body for “liveness” testing of a connection. Figure 3.14 illustrates an example of the GET_PARAMETER message. Please note, that the exemplified content fields of this figure are not specified in the RTSP standard.
**SET_PARAMETER**

The SET_PARAMETER message can be used to change parameter values on the server. Figure 3.15 contains an example of the SET_PARAMETER message. Similar to the GET_PARAMETER message, the content of the body and the content-type field depends on a particular implementation. It is recommended that a client sends only one parameter to be changed in one message in order to understand the reason if the request should fail.

```
C → S:
 SET_PARAMETER rtsp://localhost:554/mpeg4/video.mp4 RTSP/1.0
   CSeq: 123
   Content-Type: text/parameters
   Session: 988009202
   Content-Length: 20
   barparam: barstuff

S → C:
 RTSP/1.0 451 Parameter Not Understood
 CSeq: 123
 Content-Length: 10
 Content-Type: text/parameters
   barparam
```

*Note: If transport parameters of a session should be changed, SET_PARAMETER is not the right message. For that purpose, the SETUP message shall be used!*

Figure 3.15: Example of an RTSP SET_PARAMETER message

**REDIRECT**

The REDIRECT message is used to tell the client of a session that it has to switch to another server. Figure 3.16 shows an example of a REDIRECT message. The address of the new server is contained in the Location header field of the message. A missing Location header field is also possible, which means that the server cannot serve the client anymore and does not know an alternative server.
3.3 MPEG-21

This section provides a brief overview of MPEG-21 with a more detailed description of Digital Item Declaration and Digital Item Adaptation, because these parts are relevant for this thesis.

3.3.1 Introduction

MPEG-21 is an open multimedia framework, developed by the Moving Picture Experts Group (MPEG) of the International Standardization Organization (ISO). MPEG-21 was started in June 2000 with the purpose of creating a normative technology for multimedia applications of the 21st century. The framework, which is still under development, was designed to enable a transparent and augmented use of multimedia data among different devices and networks. It should provide a feasible tool for managing (or rather describing) the relations between various elements and areas concerning multimedia by introducing new standards in order to fit together the existing standards. “The intent is that the framework will cover the entire multimedia content delivery chain encompassing content creation, production, delivery, personalization, consumption, presentation and trade” [25].

The MPEG-21 multimedia framework currently comprises 17 parts, of which some have already become International Standard (ISO/IEC 21000):

![Example of an RTSP REDIRECT message](ISO.png)
• Part 1: Vision, Technologies and Strategy
• Part 2: Digital Item Declaration (DID) - *International Standard*
• Part 3: Digital Item Identification (DII) - *International Standard*
• Part 4: Intellectual Property Management and Protection (IPMP)
• Part 5: Rights Expression Language (REL) - *International Standard*
• Part 6: Rights Data Dictionary (RDD) - *International Standard*
• Part 7: Digital Item Adaptation (DIA) - *International Standard*
• Part 8: Reference Software
• Part 9: File Format
• Part 10: Digital Item Processing
• Part 11: Evaluation Methods for Persistent Association Technologies
• Part 12: Test Bed for MPEG-21 Resource Delivery
• Part 13: Scalable Video Coding (SVC)
• Part 14: Conformance Testing
• Part 15: Event Reporting
• Part 16: Binary Format
• Part 17: Fragment Identification for MPEG Media Types

A discussion of all parts of MPEG-21 is beyond the scope of this thesis. Only the parts relevant for this thesis will be described in the remainder of this section. This comprises the description of *Digital Items*, which are used to represent movies in the SeMiSystem and the *Terminal Capabilities*, which are used to specify the device properties for a client. For a more generalized view, see [25, 26].
3.3.2 Digital Item Declaration

A very often used term within the MPEG-21 framework is the Digital Item (DI). A Digital Item is an abstraction of a structured digital object, which constitutes a fundamental unit of distribution and transaction within the MPEG-21 framework. As an example of a Digital Item one can imagine a music track, an image, a movie, a music album, a video collection etc. Nevertheless, there is no technical meaning defined for a Digital Item. The purpose of the Digital Item Declaration (DID) specification is to define a set of terms and concepts which can be used to describe (or rather declare) Digital Items. The set of these terms and concepts is called Digital Item Declaration Language (DIDL). The aim of this language is to provide a flexible way of declaring Digital Items while allowing a higher-level functionality as well.

Digital Items are declared by using XML documents. This subsection continues with a description of all the elements defined for a DIDL document. Figure 3.17 shows a complete example of a DIDL document used to define a list of available movies in a VoD system.

DIDL  This element is the root element of a DIDL document. It contains either a single Item element or a Container element. Moreover, a DIDL element may contain a Declarations element. The DIDL element must include a namespace declaration expressed by the xmlns attribute, which defines the XML namespace for the entire document.

Declarations  This element can be used to declare other elements for later use in the same document. The elements will not be instantiated and can be referenced by using the Reference element. Possible elements are: Item, Descriptor, Component, Annotation, and Anchor.

Container  The Container element can be used to group other Containers or Items. It may contain a Descriptor element including information according to the purpose of the grouping. Containers can be used in a flexible way. They can represent a photo
album, a set of photo albums, a DVD collection, a music album, a play list, or any other logical grouping, for instance.

**Item**  This element can be used to represent an Item, which constitutes an object of lowest granularity within MPEG-21 that can be processed by a user. An Item can contain several sub-Items or Components which are bound to relevant descriptors. Thus, it may also contain Conditions, Descriptors, Choices, References, and Annotation elements. Please do not confuse this element with a Digital Item, which is on a higher level of abstraction. “The relationship between Items and Digital Items (as defined in ISO/IEC 21000-1:2001, MPEG-21 Vision, Technologies and Strategy) could be stated as follows: items are declarative representations of Digital Items” [27].

**Component**  A Component can be used to group a Resource with a set of descriptive elements. The following descriptive elements can be used: Condition, Descriptor, and Reference. In addition, Anchor elements can be used to specify points or regions of interest in the Resource.

**Resource**  This element can be used to represent a resource object (e.g. an image, a video, an audio clip, or a textual object). The actual resource is referenced from the Resource element either by specifying the location in the Ref attribute (e.g. by using a URI) or by specifying a relative local path in the localPath attribute. Furthermore, a MIME type for the data type of the resource should be specified in the mimeType attribute.

**Descriptor**  The Descriptor element can be used to connect descriptive data to the parent element. It can contain either a Statement element (e.g. for textual information) or a Component element (e.g. a thumbnail version of an image).
Statement This element contains information related to its parent element. This might be descriptive, control, revision tracking, or identifying information. The Statement element has a mimeType attribute, which should be used to specify the type of the containing data. It should be obvious that this provides a very flexible way of describing a Digital Item. The data contained in a Statement could be either text, HTML or XML, for instance. Moreover, it is possible to include data by reference, using the ref attribute of the Statement or using the localPath attribute.

Anchor The Anchor element can be used to bind a set of Descriptors to a particular location or part of a Resource within the same parent Component element. This might be useful to annotate a certain timestamp of a movie with a textual description, for example. The corresponding location in the Resource is specified by the fragment attribute, which can contain a string value. If several Anchor elements for a Resource are specified, the precedence attribute can be used to define the priority of those Anchors (e.g. to define the default Anchor).

Choice The Choice element can be used to provide a choice of several Selection elements which will affect the configuration of an Item. The attributes minSelections and maxSelections are used to define the number of selections that must be made for a valid choice. If both attributes are set to the value one, exactly one selection must be chosen. Both attributes can also be omitted, which means that any combination of chosen selections is valid (including all or zero). It is also possible to specify a default Selection for a Choice by using the attribute default.

Selection The Selection element can be used to provide a particular decision at configuration time, related to a Choice. A Selection has an associated predicate, defined by the select_id attribute which may relate to a Condition somewhere within an Item. The value of a Selection may be true, false, or undecided.
Figure 3.17: Example of a DIDL document
**Condition** This element can be used to define a condition for the parent element, affected by the predicates of related Selections. An application which parses the DIDL document will ignore the entire parent element if the Condition is not satisfied. A Condition can have one or more predicates, specified in the `require` attribute, which have to be true in order to satisfy the Condition. In addition, it can also contain predicates, specified in the `except` attribute, which must be false. All predicates specified in those two attributes are combined by a conjunction (AND). However, it is also possible to use a disjunction (OR) of predicates by specifying a number of Condition elements for a parent element.

**Reference** By using the Reference element it is possible to create internal and external references to Containers, Items, Components, Descriptors, Anchors, or Annotation elements. The content of a referenced element will be linked directly to the content of that element which contains the Reference element. In addition to the reuse aspect, external references could also be used to split a single DIDL document into multiple DIDL documents. The referenced element of a Reference is expressed by specifying a URI in the `target` attribute. For an internal link, the value of the `id` attribute from the referenced element can be used as such a URI.

**Annotation** The Annotation element can be used to add Descriptors, Anchors, or Assertion elements to other elements without affecting their content. The target of the Annotation element may be located somewhere else in the same document or in another (external) document. It is expressed by specifying a URI in the `target` attribute of the Annotation. For an internal link, the value of the `id` attribute of the referenced element can be used as such a URI.

**Assertion** An Assertion can be used in combination with a Choice element to assert the predicate values for the associated Selections. This could be used to save the current state/configuration of a Digital Item, for instance. An Assertion is stated by the use of an Annotation. The affected Choice element must be specified in the `target` attribute of the Assertion.
CHAPTER 3. ENABLING TECHNOLOGIES

For a more detailed description of the Digital Item Declaration Language, containing a lot of examples, please refer to [27].

3.3.3 Digital Item Adaptation

The idea behind Digital Item Adaptation (DIA) is to provide a set of tools (i.e., syntax and semantics) that may assist the adaptation process of Digital Items. These tools could, for instance, be used to describe the capabilities of a client and use them for adaptation. However, DIA is not responsible for the adaptation itself - this task has to be performed by an external application. Figure 3.18 illustrates this concept.

![Figure 3.18: Concept of Digital Item Adaptation (DIA) (taken from [5])](image)

The Digital Item Adaptation tools consist of eight major categories. However, only two parts of it will be described in this document, because the others are not relevant for this thesis. More information about DIA can be found in [5].

- **Usage Environment Description Tools**. The tools of this category can be used to describe a variety of properties related to users and their usage environment, which can be used for Digital Item Adaptation. The description comprises *User Characteristics, Terminal Capabilities, Network Characteristics,* and *Natural Environment Characteristics.*
For this thesis, Terminal Capabilities are especially important. They are used to describe the constraints of a terminal (a device respectively), regarding the processing and consumption of multimedia data. This may include:

- Codec capabilities (acceptable formats and parameters),
- Device capabilities (device-class, power-characteristics, storage and data I/O characteristics, benchmarks, IPMP Tools characteristics),
- Input-Output capabilities (display and audio output capabilities).

Figure 3.19 exemplifies a Terminal Capabilities document describing a terminal, which has two possible display devices, several audio input devices, and an equipment for audio output.
Figure 3.19: Terminal Capabilities example (taken from [5])

- **Session Mobility.** The tools of this category provide means to save the current state of a user’s interaction with a Digital Item (DI). The current state of interaction of a streaming media session might be the current media-time and the client configuration of that stream, for instance. Thus, the Session Mobility tools can be used to implement a pause/resume feature for a continuous media; i.e. save the current state of interaction into a file and resume the
consumption/interaction at a later time. Furthermore, this does also allow to transfer the state of a streaming session to another device in order to continue the session on that device.

MPEG-21 Session Mobility distinguishes between a *Content DI*, which is the original Digital Item of an interaction, and the *Context DI*, which contains the saved configuration state of the Content DI. More precisely, the state of the Selections (of the Content DI) and/or an application specific state is saved therein. Figure 3.20 and Figure 3.21 show an example of a ContentDI and a ContextDI, respectively. When a session should be reconstructed, a client must first load the ContentDI referenced by the *SessionMobilityTarget* element of the ContextDI. As a second step the specified Choice has to be configured for each Assertion of the ContextDI using the value of the Assertion.

The *MPEG-21 Session Mobility Tool* [3], described in Section 2.1, provides an example of how the Session Mobility tools can be used. This tool saves the current state of a video stream into a ContextDI, transmits the ContextDI together with the ContentDI to another instance of this tool (on another device) which reconstructs the original session on that device.

### 3.4 Conclusion

This chapter has covered those technologies which are important for the design and development of the proposed video session migration system. A detailed description of streaming media with its possible application areas, benefits, network aspects, and alternative solutions has been given.

Moreover, this chapter explained two relevant protocols used for streaming media, RTP and RTSP. While RTP is used to transmit real-time media data over a computer network, RTSP is used to control a streaming session. For this thesis it is very important to have a detailed understanding of RTSP because this protocol is used
as a basis for the proposed session migration system, discussed in the second part of this document.

Finally, this chapter provided a brief overview of some parts of MPEG-21, which is an extensive multimedia framework, designed to handle multimedia data and the related elements and issues. The Digital Item Declaration Language is a powerful tool for describing multimedia objects and relations among them. With the covered parts of the Digital Item Adaptation (DIA) tools it is possible to describe the properties of a device used for the consumption of multimedia data, and, thus, assist the adaptation process for multimedia objects. Furthermore, DIA provides means for storing and transferring the state of interaction with a multimedia object. By these features it is also possible to implement a session migration system, as Section 2.1 describes.
Figure 3.21: Session Mobility: example of a ContextDI (taken from [5])
This chapter is intended to compare some streaming servers regarding their features and supported streaming formats. Besides, this chapter defines the requirements for a media server in order to be used with the Session Migration System (SeMiSystem), developed in this thesis.

4.1 Required Features of a Media Server to Support Video Session Migration

Based on the architecture of the SeMiSystem (see Chapter 6), there are some requirements which a streaming server has to satisfy in order to be useable with the SeMiSystem:

1. The server should allow streaming of its data to an address specified in the destination field of the RTSP SETUP message, which could be different from the address of the RTSP client (see Section 7.7.2). This requirement relates to the SeMiProxy which would not be located on the same machine as the media player as usual.

2. The server must be able to understand the Terminal Capabilities, specified in a format conforming to MPEG-21 DIA (see Subsection 3.3.3), which are attached to the body of the RTSP DESCRIBE message. Furthermore, it should be able
to perform real-time transcoding of the media stream according to the specified Terminal Capabilities. This is not a compelling requirement - nevertheless, if the media server does not satisfy this requirement, it is not possible for the SeMiSystem to adapt the media stream to the hardware limitations of a client device.

4.2 Comparison of Media Streaming Servers

This section provides an overview of some popular streaming servers, including the ViTooKi MuViServer [7], which has been used for the practical part of this thesis. For a feature-comparison of all mentioned servers, please consider Figure 4.2 on page 57. Please note that the information of this section relates to the denoted references only. A practical evaluation of the servers has not been carried out (with the exception of the ViTooKi MuViServer indeed).

4.2.1 Darwin Streaming Server / QuickTime Streaming Server

The Darwin Streaming Server (DSS) is an open-source version of Apple’s QuickTime Streaming Server (QTSS) which is included in Mac OS X Server (v10.3). Although they are based on the same source code, they are slightly different. While the QuickTime Streaming Server is developed for Mac OS and provides some enhanced features, the Darwin Streaming Server is freely available for a set of operating systems, but without enhanced features. QTSS/DSS uses RTSP and RTP for media delivery and supports on-demand streaming as well as live broadcasting. In addition to Apple’s QuickTime movie format, it has native support for streaming of ISO-compliant MPEG-4- and 3GPP-files.

Note: In this document the term ISO-compliant MPEG-4 refers to the MPEG-4 standard file format (.mp4) which has been defined by the ISO. Furthermore, there exists a 3GPP file format (.3gp), defined by the Third-Generation Partnership Project.
(3GPP). This file format is primarily intended to be used with mobile phones. It can contain MPEG-4 or H.263 video, AAC and AMR audio, and 3GPP timed text. More information can be found in [28].

QTSS/DSS is also able to stream a lot of other formats (like MPEG-1/2). However, for streaming, these formats will be converted to the QuickTime Movie (.mov) format. Since version 4.0, it is possible to stream MP3 files by using the Icecast-protocol\(^1\). An interesting feature of QTSS/DSS is *skip-protection*. The idea behind skip-protection is to send data in less time than the frame rate for the stream requires. This results in an “over-buffering” at the client. Assuming there is enough bandwidth, this will cause the content of the client-buffer to grow faster and, hence, prevent an interruption of the playback on network fluctuations. Since version 4.1, an enhancement of this feature called *instant-on* is supported. It significantly reduces the start-up delay (or rather the pre-buffering time) for a requested stream, by - similar to the aforementioned feature - starting the streaming with sending much more data as required for the stream. If there is enough bandwidth, this enables an almost instantaneous response to the request. Hence, it does also allow to “quickly traverse through a movie” by using the time slider of the player. However, this feature does only work if the QuickTime Player (v6.0 or higher) is used as a client. Moreover, it does severely increase the network load and is not TCP-friendly.

QTSS/DSS uses an enhancement of the UDP called **Reliable UDP**, which has similar features to TCP like client-acknowledgement of packets, congestion-control, and packet retransmission. The use of Reliable UDP is specified in the RTSP SETUP request (see Section 3.2). If a client wants to use Reliable UDP, it has to add the *x-Retransmit* header with the name of the protocol (“our-retransmit”). For more information on Reliable UDP see [29]. Another (more standardized) approach would be the use of RTP extensions for immediate feedback and retransmission (see [8]). More information about QTSS and DSS can be found in [30, 31, 32].

\(^1\)More information about Icecast can be found here: http://www.icecast.org/
4.2.2 Windows Media Services

Microsoft’s solution for streaming media is called *Windows Media Services* (WMS). The latest version (WMS 9) is a complete redesign of previous versions (WMS 4.0, WMS 4.1), and is currently available as a package for the Windows Server 2003. WMS 9 does support on-demand streaming, as well as the transmission of live streams, which might be delivered either using unicast or multicast. With Windows Media Services, several methods can be used to request a stream from the server. Supported protocols are the Microsoft Media Server (MMS) protocol, HTTP, and RTSP. MMS is a proprietary protocol, which is similar to RTSP. In contrast to RTSP, it supports fast-forward and reverse, but is not able to record a stream. By default, the MMS protocol works on port 1755. The syntax of an MMS stream is: `mms://server_name/publishing_point_name/file_name`. MMS can be used with UDP or TCP. Depending on the used lower-level protocol, the URI starts with either `mmsu://` or `mmst://`.

**Note:** WMS 9 continues the above mentioned idea of specifying the lower-level transport protocol also for RTSP, which is not entirely conformant to the original RTSP standard! (The syntax `rtspt://` is not defined by RTSP)

However, the actually used transport protocol might not always be the protocol which was requested in the prefix of the URI, because WMS use a *protocol rollover strategy*. This means that the client and the server will negotiate a transport protocol for the request. When a client, like the Windows Media Player 9, connects to the server, it sends information about the protocols it can support. Afterwards, the server will connect to the client, using that protocol which it can support as well. WMS 9 uses the MMS protocol for the first connection attempt. Usually, the first attempt is successful and no further action is taken. However, if the first attempt has not been successful, the server starts trying other protocols. If MMS has been used for
the request, the server tries to connect to the client with RTSP/TCP\(^2\). If this is not successful, the server tries to connect with RTSP/UDP. If it is not successful again, the server uses HTTP.

WMS 9 is able to stream Windows Media Audio (WMA), Windows Media Video (WMV), Advanced Systems Format (ASF), MP3 with constant bit-rate (CBR), and JPEG images. Other file formats can also be used for streaming, if an appropriate media-parser plug-in which can convert those formats to one of the supported streaming formats is available on the server.

Similar to the *instant-on* feature of QTSS/DSS, WMS 9 provides a feature called *Fast Start*. Here the server delivers the beginning of the stream as fast as the network allows, but slows down the transmission when the prefetch buffer at the client side has been filled (and hence, the playback process is running). The server may also speed up the transmission during a session if necessary - this mechanism is called *Fast Cache*. WMS 9 also supports packet retransmission of the last ten seconds of a stream, called *UDP resend request*. However, the above mentioned features are only available, if the Windows Media Player 9 (or later) is used at the client side.

For more information about Windows Media Services 9, see [33] or visit http://www.microsoft.com/windows/windowsmedia.

### 4.2.3 Helix Universal Server

The *Helix Universal Server* is a commercial product of RealNetworks, based on the Helix DNA platform. The Helix DNA platform is an open-source project, which comprises a number of applications, including the Helix DNA Client, Helix DNA Producer and the Helix DNA Server. Helix applications are based on source code from RealNetworks that in turn can use the results of the Helix project (i.e., the applications) for its own products. Figure 4.1 shows this constitution.

In contrast to the Helix DNA Server, which is primarily intended for software developers, the current version of the Helix Universal Server (9.0) has some extra

\(^2\)RTSP was introduced with WMS 9 and is not supported by older players.
Figure 4.1: The Helix DNA platform (taken from [6])

features, including a broader support of streamable formats (e.g. MPEG-4, Windows Media, QuickTime etc).

The Helix Universal Server runs on a wide variety of operating systems. In addition to some RealNetwork formats, it can also be used to stream Macromedia Flash, Windows Media, QuickTime and MPEG files as well. Furthermore, some image and audio formats are supported too (e.g. MP3, RealAudio, JPG, etc.). For a detailed list of supported formats, consider Table 4.2. Helix Universal Server is also capable of understanding several streaming protocols. If a Windows Media Player is used at the client side, the server uses the MMS protocol for communication, while for RealPlayer or QuickTime clients, RTSP is used. For backward compatibility to older versions of the RealPlayer, the proprietary Progressive Networks Audio (PNA) format is supported as well. Helix Universal Server does also support the HTTP protocol for sending media data to clients behind a firewall. The Helix Universal Server is able to stream on-demand media as well as live broadcasts. Data can be delivered using unicast or multicast technology over TCP or UDP. If RealPlayer is used at the client side, the streaming data is delivered using the proprietary RealNetworks Data Transport (RDT) protocol as a default. For other clients, RTP is used.

With Helix Universal Server, it is possible to install a redundant server environment, which allows a client to continue receiving data, for a certain stream, from an
alternative server. The original server transmits a list of alternative servers to the client during the setup phase, which can be later used to reconnect if the connection to the original server has been broken.

More information about the Helix Universal Server can be found in [4, 34]. For more information about the Helix DNA Platform, see [6].

4.2.4 ViTooKi MuViServer

The *Video Toolkit* (ViTooKi) project [7] is an open-source project with the aim of developing a multimedia framework that can be used for a variety of multimedia applications. In addition to various other applications, this project does include a streaming server called *MuViServer*. Building on the ffmpeg library [35] and XviD [36], MuViServer currently supports streaming of ISO-compliant MPEG-4 using RTSP with RTP over UDP plus RTP extensions for immediate feedback and retransmission (see [8, 37, 38]).

A very interesting feature of MuViServer is the ability to perform real-time transcoding of the requested media. For that purpose, a client has to specify its Terminal Capabilities in the RTSP DESCRIBE message, using MPEG-21 DID as the descriptive format. Parallel to streaming, the server will directly transcode the data of the requested media file to a format compatible to the specified Terminal Capabilities in real-time.

Furthermore, the server can stream data using an adaptive approach. More precisely, the server can discard some frames if the RTCP feedback from the client indicates a bad network performance. Moreover, the server can retransmit lost UDP packets, due to the above mentioned standardized RTP extensions. Based on those two features, the server provides streaming of a single media file in an adaptive manner to a variety of clients with different network connections and different capabilities. Only the required quality of the movie is transmitted; no network capacity is wasted.

More information about ViTooKi can be found in [7, 8, 39].
<table>
<thead>
<tr>
<th>Supported Operating Systems</th>
<th>DSS 5.0</th>
<th>QSS 5.0</th>
<th>WMS 9</th>
<th>Helix Universal Server 9.0</th>
<th>ViTooKi MuViServer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mac OS X (v10.2.8 or later), Linux (Red Hat 8 or higher), Solaris 9, Windows 2000/2003 Server</td>
<td>Mac OS X v10.3 (included)</td>
<td>Windows Server 2003</td>
<td>Sun Solaris 2.7 and 2.8, Windows NT 4.0, Windows 2000 Server, Linux 2.4.18, HP UX 11.0/11.1, IBM AIX 4.3 and 5, Compaq Tru64 5.15.1A, FreeBSD 4.0 and 4.2</td>
<td>Linux, Windows 2000, Windows XP</td>
</tr>
<tr>
<td>Protocols</td>
<td>RTSP, HTTP, RTP, SDP</td>
<td>RTSP, HTTP, MMS</td>
<td>RTSP, HTTP, MMS, RTP, SDP</td>
<td>RTSP, HTTP, MMS, RTP, SDP</td>
<td>RTSP, RTP, SDP</td>
</tr>
<tr>
<td>Stream-Formarts</td>
<td>Native: Hintsed QuickTime (.mov), hintsed MPEG-4, hintsed 3GPP, MP3 (using looast compatible protocols)</td>
<td>Native: ASF, WMV, WMV, MP3, JPEG</td>
<td>RealAudio (.rm), RealVideo (.rm, .rmvb), RealPix (.rg), RealText (.rt), Macromedia Flash, Windows MediaCast (.wm, .wmv), QuickTime Movie Format (.mov), MPEG-1, MPEG1-Layer 3 (MP3), MPEG-4, GIF (.gif), JPEG (.jpeg, .jpg), PNG (.png), AU (.au), AIFF (.aif, .aiff), WAV (.wav)</td>
<td>MPEG-4, MPEG1 Layer 3 (.MP3)</td>
<td></td>
</tr>
<tr>
<td>Live Streaming</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IP Multicast</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Interesting Features</td>
<td>kip protection, instant-on</td>
<td>fast streaming (fast start, fast cache, fast recovery, fast reconnect)</td>
<td></td>
<td>real-time transcoding, dynamic adaptation, retransmission of UDP packets, RTP immediate feedback</td>
<td></td>
</tr>
</tbody>
</table>
This chapter starts with a definition of the requirements for a media player in order to be used with the SeMiSystem. Furthermore, it continues with an evaluation of media players for PDAs that support streaming media.

5.1 Required Features of a Media Player to Support Video Session Migration

Based on the architecture of the SeMiSystem, there are some minor requirements which a client system has to meet. The first requirement is the installation of a web browser, which can be used to access the SeMiServer. The remaining requirements for a media player are:

1. It has to support the Real-Time Streaming Protocol (RTSP).

2. It has to be able to receive the streaming data (e.g. RTP packets) from a host, different to the host of the RTSP connection.

3. It should be able to process
   - a direct RTSP URI (clicked by the user in the web browser)
   - or an SDP file (opened by a user).
If a player does not support a direct RTSP URI or an SDP file, the SeMiSystem can be easily enhanced to provide other types/formats.

Basically, every player fulfilling these requirements can be used with the SeMiSystem.

5.2 Evaluation of Media Players for PDAs

Based on the aforementioned requirements, an evaluation of available media players for PDAs which support streaming media was accomplished at the beginning of this thesis in order to determine the further course of actions for the project. This section presents the results of this evaluation.

The evaluation was performed by using the following configuration:

- The Helix Universal Server was used as a streaming server. This server was chosen because it provides the most extensive support of media formats. Media players that support streaming of ISO-compliant MPEG-4 files were tested with the ViTooKi MuViServer as well. As a network, an 11 Mbit/sec Wireless LAN was used.

- The media players were tested with the following media formats: MPEG-1, ISO-compliant MPEG-4 (.mp4), DivX 5.03 (.avi), XviD 1.0.3 (.avi) with simple profile on level 3, MP3, RealMedia 8, Apple QuickTime, and Windows Media 8. In order to produce the ISO-compliant MPEG-4 test files, the mp4creator tool, which is part of the MPEG4IP\(^1\) project, was used to process the XviD coded avi files. MPEG-2 was not tested, because it is only an advanced version of MPEG-1, primarily intended to support higher resolutions and bandwidths and, thus, not important for PDAs.

\(^1\)See http://www.mpeg4ip.net
As mobile devices, two different PDAs were used:

- An HP iPAQ h2210, running Microsoft Pocket PC Version 4.20.0 (Build 14053). This device is equipped with a 400 MHz Intel XScale CPU, 64 MB RAM and a display resolution of 240 x 320 pixels.

- A Compaq iPAQ H3630, running two different operating systems. The first one was Microsoft Pocket PC Version 3.0.11171 (Build 11178) and the second one was Familiar Linux 0.7.2 in combination with Opie 1.0.2 (Kernel version 2.4.19). This device is equipped with a 206 MHz Intel StrongARM CPU, 32 MB RAM and a display resolution of 240 x 320 pixels.

5.2.1 Players tested with Microsoft Pocket PC

This subsection contains the results for those players which will run on the Microsoft Pocket PC operating system.

pvPlayer 3.3 (build 007)

The PacketVideo Player (pvPlayer) can play 3GPP- and ISO-compliant MPEG-4 files and can be used with the Microsoft Pocket PC operating system. It has been developed in 2003 by PacketVideo Corporation[40]. However, they have stopped the development for Pocket PC because their general focus is based on embedded systems (e.g. mobile phones).

The pvPlayer does support streaming of MPEG-4 files via RTSP/RTP as well. It is the only player in this comparison which is able to stably receive and play videos from the ViTooKi MuViServer, described earlier. However, the player is not able to play video files which have a bigger resolution than 192 x 144 pixels, even not locally. Moreover, the player does only support I-Frames and P-Frames; B-Frames are not

\[2\text{Also known as Windows CE 3.0}\]
\[3\text{StrongARM is a processor architecture developed by Intel. More information can be found here: http://www.intel.com/design/strong/datashts/278241.htm}\]
supported. In addition, it does not have an integrated support of an RTSP proxy. Figure 5.1 summarizes the evaluation of the pvPlayer 3.3.

<table>
<thead>
<tr>
<th>pvPlayer</th>
<th>MPEG-1</th>
<th>MPEG-4</th>
<th>MP3</th>
<th>QT</th>
<th>RealM</th>
<th>WM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media support</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Streaming support</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Streaming protocol(s)</td>
<td>RTSP</td>
<td>RTP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Developed by</td>
<td>PacketVideo Corporation, 2003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software version</td>
<td>3.3 build 007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Licence</td>
<td>Unknown</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supported file formats</td>
<td>.mp4, .3gp, .sdp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.1: Summary of the pvPlayer

**Philips Platform4 Player for PocketPC 3.0**

The *Philips Platform4 Player* has been developed by Koninklijke Philips Electronics N.V. in 2003. It was a product of the *Digital Networks* business group of Philips Electronics (formerly reachable at http://www.digitalnetworks.philips.com). Nowadays, it is no longer available.

The player does support playback of MP3 files and ISO-compliant MPEG-4 files. It has an HTTP based download-and-play feature and is able to receive MPEG-4 data via RTSP/RTP streaming as well. However, with the configuration used for this evaluation, the player is not able to start the playback for MPEG-4 files that have been requested by RTSP (HTTP download-and-play works).

The player does perform a very slow playback of MPEG-4 files. This problem might be related to the bad performance of the iPAQ 3630 device, which was used for testing this player. The use of this device was necessary, because there is no further development and support for the Platform4 Player. It has been developed for

---

4This website has changed to http://www.software.philips.com/mp4net/
Pocket PC 3.0 and does not work on current operating systems like Pocket PC v4.20. Figure 5.2 summarizes the evaluation of the Philips Platform4 Player.

<table>
<thead>
<tr>
<th>Philips Platform4 Player for PocketPC</th>
<th>MPEG-1</th>
<th>MPEG-4</th>
<th>MP3</th>
<th>QT</th>
<th>RealM</th>
<th>WM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media support</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Streaming support</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Streaming protocol(s)</td>
<td>RTSP, RTSP, HTTP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software version</td>
<td>3.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Licence</td>
<td>Player is not available anymore</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supported file formats</td>
<td>.mp4, .mini, .smi, .smil, .3gp, .amr, .adp, .mp3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Supports RTSP proxy, allows selection of UDP/TCP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.2: Summary of the Philips Platform4 Player

**PictPocket Cinema 4.0**

*PictPocket Cinema* 4.0 [41], a commercial product developed by DigiSoft, is a media player for PDAs running Microsoft Pocket PC 3.0 or higher.

It supports playback of MPEG-1, ISO-compliant MPEG-4, MP3, QuickTime, and Windows Media. Moreover, it has a HTTP streaming feature (progressive download) for AVI, QuickTime and ASF files. According to the help file it should also support RTSP streaming of MPEG-4 files. However, such a support could not be recognized in the test environment.

The player seems to have some performance problems with MPEG movies. On the test device (the iPAQ h2210), it has not been able to smoothly play an MPEG-4 movie from local storage. In addition, the player shows some “choppy behaviour” when playing an MPEG-1 movie. Figure 5.3 summarizes the evaluation of PictPocket Cinema 4.0.
### Pocket DivX Player 0.8

The *Pocket DivX Player* [42] is an open source media player for Microsoft Pocket PC that has been developed by Marc E. Dukette and Pedro Mateu. The latest version (v0.8) was released in January 2002.

The player allows local playback of MPEG-1, MP3, DivX, and OpenDivX (not tested) and has support for HTTP streaming (progressive download). However, the player seems to have some problems with the iPAQ h2210 device (or its operating system). On that device the player is not able to properly play any of the above mentioned formats while they work fine with the iPAQ H3630 device. Figure 5.4 summarizes the evaluation of the Pocket DivX Player 0.8.

### Pocket MVP 0.8.011804 and 0.8072703

The *Pocket Music and Video Player* (MVP Player [43]) is an open source media player developed by Marc E. Dukette and Pedro Mateu, who are the project leaders of the Pocket DivX Player as well. Thus, the Pocket MVP Player is similar to the Pocket DivX Player but has support of more advanced features. It supports playback of MPEG-1, DivX, XviD, OpenDivX (not tested), OGG (not tested), VP3 (not tested)

---

**Table: PocketDivXPlayerSummary**

<table>
<thead>
<tr>
<th></th>
<th>MPEG-1</th>
<th>MPEG-4</th>
<th>MP3</th>
<th>QT</th>
<th>RealM</th>
<th>WM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media support</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Streaming support</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Streaming protocol(s)</td>
<td>HTTPp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5.3: Summary of the PictPocket Cinema Player**
and MP3 (by using a third-party library called *MAD Decoder*) and HTTP streaming (progressive download) of these files as well.

For this evaluation the player has been tested on both the iPAQ H3630 and the iPAQ h2210 device. Although there exists a Windows Mobile 2003 (Pocket PC 4.20) version (0.8072703), the player does not officially support the iPAQ h2210 device (according to the help file of the player). Nevertheless, version 0.8072703 is able to play MPEG-1, DivX, and MP3 on the iPAQ h2210 device (HTTP streaming is not possible). Figure 5.5 summarizes the evaluation of Pocket MVP. More information about Pocket MVP can be found at http://www.pocketmvp.com.

**PocketTV 0.15.5**

*PocketTV* [44] is a media player for MPEG-1 files which is available for a number of mobile devices running the Microsoft Pocket PC operating system. There are two versions of PocketTV, a free version for personal use, and an enterprise version for commercial use. The main technical difference between those two versions is the improved performance and the advanced support of the enterprise version.

As already mentioned, the only file format supported by PocketTV is MPEG-1.
CHAPTER 5. MEDIA PLAYERS

Figure 5.5: Summary of the PocketMVP Player

The player is also able to receive MPEG-1 over a network using HTTP download-and-play or HTTP progressive download. Figure 5.6 summarizes the evaluation of PocketTV 0.15.5.

Figure 5.6: Summary of the PocketTV Player
RealOne Player 2.0.0.28

The RealOne Player for PocketPC\textsuperscript{5} [45] is a special version of the RealPlayer, which is available for Symbian OS, Palm OS 5, Microsoft Pocket PC, and some smartphones like the Nokia 9200 Series Communicators or Nokia Series 60 phones (3650, 7650).

The player supports playback of RealVideo, RealAudio, and 3GPP content. Furthermore, it does also support streaming via RTSP and allows using an RTSP Proxy. Although the website\textsuperscript{[45]} says that "The functionality of RealPlayer varies across devices", in the test environment the only supported format by the player was RealMedia. Figure 5.7 summarizes the evaluation of the RealOne Player. More information about the RealOne Player for PocketPCs can be found at [45].

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
\textbf{RealOne Player} & \textbf{MPEG-1} & \textbf{MPEG-4} & \textbf{MP3} & \textbf{QT} & \textbf{RealM} & \textbf{WM} \\
\hline
\textbf{Media support} & & & & & & \checkmark \\
\textbf{Streaming support} & & & & & & \checkmark \\
\textbf{Streaming protocol(s)} & & & & & & RTSP \textbf{RTP} \\
\hline
\textbf{Developed by} & \texttt{RealNetworks Inc., 2002} & & & & & \\
\textbf{Software version} & \texttt{2.0.0.28} & & & & & \\
\textbf{Licence} & Free & & & & & \\
\textbf{Supported file formats} & \texttt{rm, ra} & & & & & \\
\textbf{Comments} & Supports RTSP proxy & & & & & \\
\hline
\end{tabular}
\caption{Summary of the RealOne Player}
\end{table}

Windows Media Player 9 (build 14053)

The Windows Media Player 9 [46] for Pocket PC is a media player, developed by Microsoft, that is shipped with the Microsoft Pocket PC operating system. It supports local playback of Windows Media and MP3 files. Furthermore, the player is also capable of receiving streaming of Windows Media files by using MMS or HTTP

\textsuperscript{5}Also called RealPlayer for Mobile Devices
download-and-play. Figure 5.8 summarizes the evaluation of the Windows Media Player 9.

![Windows Media Player table]

<table>
<thead>
<tr>
<th>Windows Media Player</th>
<th>MPEG-1</th>
<th>MPEG-4</th>
<th>MP3</th>
<th>QT</th>
<th>RealM</th>
<th>WM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media support</td>
<td>ISO</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Streaming support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Streaming protocol(s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MMS, HTTP</td>
<td></td>
</tr>
<tr>
<td>Developed by</td>
<td>Microsoft, 2003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software version</td>
<td>9.0 build 14063</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Licence</td>
<td>Free; shipped with operating system</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supported file formats</td>
<td>.mp3, .wmv, .wm, .wma, .asf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comments</td>
<td>Allows selection of UDP/TCP/HTTP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.8: Summary of the Windows Media Player

5.2.2 Players tested with Familiar Linux

This subsection contains the results of those players, which have been tested with the Familiar Linux system, installed on the Compaq iPAQ H3630 device. In order to enable the reader to try one of these players, Appendix E shows how to install the Familiar Linux on such a device.

**MPlayer 0.90rc1-2.95.1**

Based on the original MPlayer\(^6\), which is a movie player for Linux with a broad support of file formats, there exist some ports which also run on an ARM based PDA using Familiar Linux. The player uses the FFmpeg/libavcodec libraries for decoding. In the test environment, version 0.90rc1-2.95.1 has been used, which can be downloaded from: http://www.killefiz.de/zaurus/showdetail.php?app=803

The player supports MPEG-1, DivX, XviD, MP3, QuickTime, RealMedia, and Windows Media. Furthermore, it is able to receive these files from a network resource

---

\(^6\)See http://www.mplayerhq.hu/
by using HTTP progressive download. MPlayer uses the LIVE.COM streaming media library\(^7\) in order to support RTSP streaming. A port of MPlayer including this library, or a separate ARM based installation package of it could not be found for evaluation. Thus, RTSP streaming could not be evaluated.

In the test environment the player was not able to open RealMedia or Windows Media files due to missing codecs. Moreover, it was not possible to display any video content of the above mentioned files on the screen of the iPAQ. Neither using different configurations (e.g. Familiar Linux with Opie (Qt based) or with GPE (X11 based)) nor exporting the display to another Linux machine was successful. The player terminated with the message **FATAL: Couldn’t initialize video filters (-vop) or video output (-vo)!** Nevertheless, using the option `-vo null` in order to prevent the output of any video content, the player was able to start the playback for those files; only the audio track was audible. Figure 5.9 summarizes the evaluation of MPlayer 0.90rc1.

\(^7\)More information can be found at [http://www.live.com/liveMedia/](http://www.live.com/liveMedia/)
Opie Player 2

The *Open Palmtop Integrated Environment* (Opie) is an open source based graphical user environment for PDAs running Linux. It was originally split off from the Qtopia environment, developed by Trolltech [47], and can be used on top of an underlying Linux package like the Familiar Linux.

Within Opie there exist a lot of applications; one of them is the *Opie Player 2*. It is based on the *xine*\(^8\) multimedia player and supports playback of MPEG-1, MP3, OGG (not tested), QuickTime, and DivX. Moreover, it is able to receive these files from a network resource by using HTTP progressive download. With the used test configuration the player has some problems with QuickTime movies. It terminates the playback with the error message “no demux found for this media”. Figure 5.10 summarizes the evaluation of the Opie Media Player 2. More information about this player can be found at [48].

<table>
<thead>
<tr>
<th>Opie Media Player 2</th>
<th>MPEG-1</th>
<th>MPEG-4</th>
<th>MP3</th>
<th>QT</th>
<th>RealM</th>
<th>WM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media support</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Streaming support</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Streaming protocol(s)</td>
<td>HTTP(^8)</td>
<td>HTTP(^8)</td>
<td>HTTP(^8)</td>
<td>HTTP(^8)</td>
<td>HTTP(^8)</td>
<td></td>
</tr>
<tr>
<td>Developed by</td>
<td>Opie (Open Palmtop Integrated Environment) open source group, based on xine</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software version</td>
<td>1.0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Licence</td>
<td>GNU GPL Licence (Open Source)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supported file formats</td>
<td>.avi, .ogg, .mp3, .mov, .mpg, .mpeg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^8\)See [http://xinehq.de/](http://xinehq.de/)

Figure 5.10: Summary of the Opie Player 2
5.2.3 Conclusion

As shown in this evaluation, there are many players for PDAs that support playback of MPEG-1 or MP3 files and can receive media data by using HTTP.

However, the number of media players for PDAs that support true streaming of ISO-compliant MPEG-4 files via RTSP/RTP is very small. The PacketVideo Player (pvPlayer) is the only player in this evaluation that has stable support of this feature. Thus, the pvPlayer has been used for this thesis in order to test the session migration system on PDA devices.
Part II

Implementation of a Video Session Migration System
This chapter describes the architecture of the Session Migration System (SeMiSystem), developed for this thesis. The SeMiSystem realizes a video session migration system that is independent of the chosen media player on the client.

6.1 Introduction

A relatively simple way of developing a video session migration system would be to implement a client application that is closely coupled to a media player. In this way, the application could easily retrieve information about the session from the player (e.g. by an API call) and transmit it to another application that is located on the destination device. The application on the destination device needs to be able to cause the player to request the concerned video using the proper media time. After a successful re-establishment of the session, the application on the original device can close the session.

However, such an application has a number of drawbacks. First, this application could only be used in combination with specially developed players. If a player changes its API in later versions, the client application will not work with this player anymore. Besides, new players can only be used when adapting the client application. Second, if a programming language is used that is not platform-independent the application has to be ported to every operating system which should be supported. Another
problem of such an application is the limitation to those operating systems supported by the media player.

6.2 Architecture Overview

The SeMiSystem uses a completely different approach which is independent of the operating system and the media player as well. Thus, a media player on a Macintosh Laptop could be used as a client as well as a player on a Linux PC or a player on a Windows PDA. Any media player which meets some minor requirements can be used with the SeMiSystem (see Section 5.1). As already mentioned in Section 1.3.2, the business logic of a session migration system can be located in a client application or in a client/server architecture.

Instead of a client application, the SeMiSystem uses a centralized approach in order to be accessible from any client using standardized protocols. In addition to the session migration functionality, the SeMiSystem incorporates a user management including user profiles and user rights. Thus, the SeMiSystem can also be used as a video stream management system. It consists of two collaborating parts:

- **The SeMiServer** which is used to manage
  - Users and their profiles
  - Available videos
  - User rights

- **The SeMiProxy** which is used to
  - Forward RTSP messages from the client to the server and vice versa. This will allow the proxy to monitor active sessions and their current media time.
Add Terminal Capabilities to the request, which are further used by the media server to transcode the video content according to the requirements of the user/client.

Execute the process of a video session migration.

The SeMiServer represents the interface between the SeMiSystem and the end user. As apparent in Figure 6.1, it consists of three programs, running on top of a web server, which are accessible from clients (web browsers) by the Common Gateway Interface (CGI). This should ensure that a huge set of different clients will be able to use the SeMiServer. The main purpose of the SeMiServer is to provide a simple way of starting or migrating a video session for the user. The programs of the SeMiServer are responsible for:

- Allowing the user to authorize him- or herself to the SeMiSystem
- Providing an authorized user with a selection of available profiles
- Providing an authorized user with a selection of available videos

When a user wants to interact with the SeMiSystem, a web browser has to be used. The user must first logon to the SeMiServer and select an appropriate profile. After a successful logon, the SeMiServer will provide the following possibilities to the user:

- Start a new session
- Migrate an existing session
- Copy an existing session
  - From the current position
  - From a previously marked position
For every video, the SeMiServer generates a link which can be used to start a streaming session of that video. The link contains the RTSP-URI of the selected video and some special parameters (summarized in Table 6.1), added by the SeMiServer. For example, an original RTSP-URI like `rtsp://127.0.0.1/xmen2.mp4` will be changed to: `rtsp://127.0.0.1/xmen2.mp4?uid=3&proid=1&id=stream1`.

This link must be handed over to a media player of the client which is able to use the RTSP protocol. On a well configured system, this hand-over will be performed automatically when clicking the `rtsp://` link in the web browser. However, for those systems or media players which are not able to directly process an RTSP-URI, the
SeMiServer provides alternative ways. An example is the use of SDP files instead of RTSP-URIs.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id=</td>
<td>Id of the concerned video</td>
</tr>
<tr>
<td>uid=</td>
<td>Id of the user currently logged on</td>
</tr>
<tr>
<td>proid=</td>
<td>Id of the selected profile</td>
</tr>
<tr>
<td>sid=</td>
<td>Session-Id of the concerned active video session (to migrate/copy)</td>
</tr>
<tr>
<td>action=</td>
<td>Number of desired action, chosen by the user; Possible values: 1=migrate, 2=copy, 3=copy section</td>
</tr>
<tr>
<td>section=</td>
<td>Number of a starting section of a session which should be copied (used for continuing on marked positions)</td>
</tr>
<tr>
<td>sub=</td>
<td>Number of seconds that should be subtracted from the current media time of an active session</td>
</tr>
<tr>
<td>host=</td>
<td>Host address of real media server (described in Section 7.7.3)</td>
</tr>
<tr>
<td>port=</td>
<td>Port number of real media server (described in Section 7.7.3)</td>
</tr>
</tbody>
</table>

Table 6.1: Possible parameters of an RTSP-URI, added by the SeMiServer

Due to the additional parameters added by the SeMiServer, the links cannot be used by a media player for directly requesting the video stream from the media server. Instead, the media player on the client must be configured to use an RTSP proxy, which has to be the SeMiProxy. In this way, the RTSP messages will be redirected to the proxy which will then forward every message to the proper destination. When the SeMiProxy forwards a message to the media server, it removes the parameters which have been added by the SeMiServer. Furthermore, corresponding to the selected profile, the SeMiProxy adds Terminal Capabilities (see Subsection 3.3.3) to the RTSP DESCRIBE message. That will cause the media server to transcode the requested video to the specified requirements, if possible.
Due to the parameters of Table 6.1, the SeMiProxy is able to monitor and control the RTSP sessions. More precisely, the proxy recognizes:

- Which user has initiated the request?
- Which video/session has been requested by the user?
- Which profile was selected by the user?
- Which action has been requested for that video?

In addition to this information, the proxy also notices the start and stop events of a video session and, thus, knows which sessions are currently active. The SeMiServer can request this information from the SeMiProxy and adds it to the list of available videos. This allows a user to select an active session which should be migrated to the current device of the user. In this manner, a user can start the migration of an active session, using the same procedure as used for starting a new video session. The user has to go to the destination device, log on to the SeMiServer and choose the required session to be migrated. When a user requests a session to be migrated, the SeMiProxy performs a client-based media time estimation for that session and uses the result for the RTSP PLAY message of the new request. Thus, the session is continued on that position which has been reached on the original device. In contrast to a copied session, the SeMiProxy closes the original session after a successful migration.

In order to work on the same data, the SeMiServer and the SeMiProxy use a relational database. The design of this database is illustrated in Figure 6.2. It contains information about: users, profiles of users, and available videos. In addition, this database does also contain the rights of a user to start a video, or to migrate an active session of a video. Thus, the SeMiSystem is not only a video session migration system, but also a video management system with respect to user permissions. While users, profiles and rights must be entered by an administrator, the video content has
to be periodically synchronized with the media server. Therefore, the media server used by the SeMiSystem should provide an interface which can be used to retrieve a list of available media files.

As apparent in Figure 6.2, the information about active sessions is not stored in the database. The SeMiProxy rather manages this information in its local memory and allows the SeMiServer to request this information using a TCP connection.
6.3 Migration Example Procedure

This section will give a step-by-step guidance of how to perform a video session migration with the SeMiSystem.

First of all, the user has to connect to the SeMiServer by entering the following address into a web browser: \texttt{http://<serveraddress>:8081/index.html}. After a successful connection has been established, the SeMiServer returns a list of registered usernames from which the user has to choose one. Moreover, a user has to select a certain profile. Figure 6.3 will illustrate this procedure. As one can see from this figure, the user may also change the page size. This should allow using the SeMiServer-GUI from a device with a large display as well as from a device with a small display.

![Figure 6.3: SeMiServer - logon](image-url)
<table>
<thead>
<tr>
<th>SMALL</th>
<th>LARGE</th>
</tr>
</thead>
</table>

**Description:**
- **Title:** SeMiServer - start a video session
- **Image:** A screenshot of the SeMiServer interface showing options for video session start and size selection.

**Figure 6.4:** SeMiServer - start a video session
After the user has selected a profile and entered the correct password, a list of available videos is presented (Figure 6.4). The user can start a video session by clicking the Start link beside the corresponding entry. Moreover, the user can also change his/her previously selected profile or the link type by using the appropriate selection-box on the top of the page. Figure 6.5 shows an active video session and the corresponding list, generated by the SeMiServer. In this screenshot, Embedded QT was chosen as a link type. Thus, the video is opened in a popup window containing an embedded media player. Corresponding to the previously chosen profile (see Figure 6.3), the video is displayed using grey-scaled colour space and a resolution of 352 x 288 pixels.

Figure 6.5: SeMiSystem - video session started
In the right part of Figure 6.5 the output of the SeMiServer is illustrated, which has been generated after the session has been started. The additional information about active sessions contains:

- The link to the corresponding video of the session.
- The address of the client receiving this video session.
- The state of the session (active/paused).
- The advanced time, the creation time and the duration of the session.
- If this session has been already migrated/copied: the elapsed time of the original session on the time of migration.

A user can now migrate or copy a session by clicking the Migrate or Copy link of the session entry. Furthermore, the user may select the number of seconds to subtract from or add to the current media time of the session (e.g. a user wants to copy the session two seconds before the current media time). Figure 6.6 shows the result of a session copy process, using the QCIF profile and 2 seconds to subtract. Due to the QCIF profile, the video is displayed using a 16 bits-per-pixel colour space and a resolution of 176 x 144 pixels. The new video session is requested from the media server using the current media time of the original session.
<table>
<thead>
<tr>
<th>User</th>
<th>Session Status</th>
<th>Migrate</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>User1</td>
<td>Active</td>
<td><img src="image1.png" alt="Migrate" /></td>
<td><img src="image2.png" alt="Copy" /></td>
</tr>
<tr>
<td>User2</td>
<td>Inactive</td>
<td><img src="image3.png" alt="Migrate" /></td>
<td><img src="image4.png" alt="Copy" /></td>
</tr>
</tbody>
</table>

Figure 6.6: SeMiSystem - video session copied
This chapter presents some details about the implementation of the SeMiSystem developed in the practical part of this thesis.

### 7.1 Configuration

In addition to the SeMiServer and the SeMiProxy, a media server is required in order to use the SeMiSystem. In this work the ViTooKi MuViServer has been used. Tests of a session migration have been performed with the ViTooKi MuViPlayer on Linux, the QuickTime player v.6.5.2 on Microsoft Windows and the pvPlayer v3.3 on Microsoft Pocket PC.

### 7.2 Simplifications in the Current Implementation

The current implementation of the SeMiSystem uses the following simplifications:

- The system does not consider user-rights (for starting or migrating videos).
- Instead of a database, simple text files are used for storing the necessary information.
- The static content of the SeMiServer (i.e., users and profiles) cannot be altered by a web interface. Thus, an administrator has to create this content by editing the text files.
• The list of videos on the SeMiServer is not periodically synchronized with the media server. Rather, the current implementation requests the list of videos on every logon of a user to the SeMiSystem. More information can be found in Section 7.6.

7.3 Data Management

The current implementation of the SeMiSystem does not use a relational database as mentioned in Section 6.2. Instead, text files are used for storing data. This should allow creating a demonstration system which can be easily installed (i.e., without a database system). Users and profiles are stored in two simple text files, named .users and .profiles. Figure 7.1 exemplifies the content of these text files.

Figure 7.1: SeMiSystem - text files for users and profiles

The description of a video is stored using an XML file according to the syntax of MPEG-21 DID (see Section 3.3.2). The SeMiServer directly requests this file from the media server. The next subsections contain a detailed explanation of how this works.
7.4 Modifications to the MuViServer

7.4.1 Movielist Creation

As already mentioned before, the list of videos should be synchronized between the SeMiServer and the media server. Thus, the media server must support an interface for providing the video content. Therefore, the MuViServer has been enhanced to create a movie list using the MPEG-21 DIDL (see Section 3.3.2) format. Due to this enhancement, the MuViServer creates a recent version of this list on every start and hands it over to a built-in simple HTTP server.

The creation process adds an entry of every known video file\(^1\), which is found in the directory-tree starting at the root directory of the MuViServer, to the list. The list contains the name and the RTSP-URI of every media file on the media server. In addition, an available image and/or a description of a video is considered too. The creation process uses the metadata subdirectory to search for such additional information, which has to be stored using the same filename as the media file, however using a different extension. Descriptive text must be stored as plain text (.txt); images must be stored in the JPEG format (.jpg).

The resulting list can be requested by the SeMiServer via HTTP in order to get the list of videos of the media server. Listing D.1 of Appendix D shows the syntax of the generated video list. As apparent in this listing, the file contains a reference to an XML style sheet, which can be used by an XSL Transformation (XSLT) processor [49]. Listing D.2 illustrates the syntax of this style sheet. Some web browsers have an inherent support for XSLT transformations. Thus, using this style sheet, the video list sent by the media server can be transformed into a well-formatted HTML file. Figure 7.2 illustrates the result of an XSLT transformation using the files from Listing D.1 and D.2. Moreover, the MuViServer does also generate an HTML based movielist which can be used by clients that do not contain an XSLT processor. Due to this functionality, a user can request the list of available videos from the media server.

\(^1\)Currently, the following extensions are considered: .avi,.mpeg,.mpg,.mp3,.mp4
by simply using a web browser. This functionality of the media server can be used completely independent from the SeMiSystem.

muviserver - list of available movies

If you want to start a video streaming, please select one of these available streams:

- xmen2.mp4
  No description available
  Watch in: xmen2

- letm.mp4
  Directed by: Peter Cawson
  Writing credits: J.R.R. Tolkien
  Actors: Karl Appleby, Alexandra Acton, Beat Acton,
  John Be I, Simon Brown, Tara von Braun,
  They see the 8.5/10 (67,251 votes)
  Watch in: LETM

- two_bsbeth.mp4
  No description available
  Watch in: TWO_BSBETH

Created by Klaus Schöffmann, University of Klagenfurt - ITEC, 2004

Figure 7.2: Result of an XSLT transformation using the video list from the MuViServer in combination with an XML style sheet

7.4.2 Media Time Request

The MuViServer has been extended by another functionality which is used by the SeMiProxy for the media time management of an active session. This functionality allows the proxy to request a server based estimation of the client's media time
of an active RTSP session. The proxy can request this value by using the GET_PARAMETER message of RTSP with the newly introduced field position. The MuViServer returns the estimated media time of the session in milliseconds. Listing 7.1 exemplifies such a request.

Note: As described in Section 3.2, the body of the GET_PARAMETER message is not defined by the RTSP standard. This definition is rather left to the implementation. The position field was introduced in this thesis - it is not defined in the RTSP standard!

Listing 7.1: Message used to request the media time of a client

<table>
<thead>
<tr>
<th>P→S</th>
<th>GET_PARAMETER rtp://127.0.0.1/imen2.mp4 RTSP/1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSeq:</td>
<td>5</td>
</tr>
<tr>
<td>Content-Type:</td>
<td>text/parameters</td>
</tr>
<tr>
<td>Session:</td>
<td>123456789</td>
</tr>
<tr>
<td>Content-Length:</td>
<td>12</td>
</tr>
<tr>
<td>position</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S→P</th>
<th>RTSP/1.0 200 OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSeq:</td>
<td>5</td>
</tr>
<tr>
<td>Content-Length:</td>
<td>21</td>
</tr>
<tr>
<td>Content-Type:</td>
<td>text/parameters</td>
</tr>
</tbody>
</table>

7.5 Modifications to the MuViPlayer

7.5.1 Media Time Request

The MuViPlayer has also been extended by the media time request functionality. Thus, the proxy can also request the real media time of the MuViPlayer by using the GET_PARAMETER message of RTSP as described above.
7.6 The SeMiServer

The SeMiServer was implemented on Linux using C++. It currently consists of three main programs which are designed to be accessed by using the Common Gateway Interface (CGI) protocol. The programs return a web page (i.e., HTML code) as a result of a program call. An important goal for the design of the resulting web page was the compatibility to a huge set of client devices. The web page should be usable on a PC as well as on a PDA.

The remainder of this section describes which CGI programs are used on the SeMiServer and how they work.

7.6.1 Provide a Selection of Users

This program (called readusers.cgi) is initially called from index.html when a user connects to the SeMiServer (see Section 6.3). It is used to read all users from the .users file and to return them as a selection list on a web page. After the user has chosen an entry, readprofiles.cgi is called for further processing.

Furthermore, this program is also responsible for requesting the list of videos from the media server. The program opens a TCP connection to the media server and uses the HTTP protocol to request the list of videos. If the list contains references to images (on the media server) these images will be requested as well.

Please note that the request to the media server is not executed periodically. It is rather performed on every login of a user to the SeMiServer. The program does not check currently active sessions when requesting the list of videos. This simplified behaviour may lead to the following situation: A user has started a new session of a video with a certain video ID. The logon of another user causes the SeMiServer to request the list of videos again. At this time, the content on the media server and the IDs may have been changed. Thus, the video-ID of the active session may not belong to the same video on this new list. As a result, the SeMiServer will display the wrong
video reference for an active session, to the new user, on the generated session list because it does only use the ID for identifying the reference to active sessions. An easy solution for that problem would be the use of unique IDs in the list of movies on the MuViServer. Thus, newly added videos would not change the IDs of the already existing ones.

7.6.2 Provide a Selection of Profiles

This program (called readprofiles.cgi) is responsible for reading available profiles of a specified user from the .profiles file and to return them as a selection list on a web page. The resulting web page does also contain a password field and information about the properties of each profile. The user has to select a profile and enter a password in order to get the list of videos, which is generated by logon.cgi.

7.6.3 Provide a Selection of Videos and Sessions

This program (called logon.cgi) is the most important part of the SeMiServer. First, it checks if the previously entered password of a specified user is valid. Then, it loads the list of videos and converts it into an HTML representation, which is similar to the result of the XSLT transformation described earlier (see page 86). The main difference is the aforementioned extension of the RTSP-URIs by the parameters defined in Table 6.1.

Different Link Types    The SeMiServer generates links which can be used from a client to start a session or to migrate/copy a session. The SeMiServer offers different link types for those links. It should be easy to integrate new link types (e.g. links that will start a player in a Java applet).
Currently, the following link types are implemented:

1. **RTSP links.** This is the default setting of the SeMiServer. It produces common RTSP links which can be used by most of the media players.

2. **SDP files.** Here the program creates an SDP file for every entry of the list and uses this file as the destination for the link. This might be useful for a client which cannot directly process RTSP links but is able to process SDP files. The generated file does only contain three lines. For the example used above, it would look like this:

```plaintext
v=0
s=movie_stream1_u3_p1
a=control:rtsp://127.0.0.1/xmen2.mp4?uid=3&proid=1&id=stream1
```

For a description of the fields used in the SDP file, please see Appendix C.

The name of the SDP file corresponds to the id of the video, the user, and the profile. In the previous example the SDP file would be named `movie_stream1_u3_p1.sdp`. This naming policy is necessary because other users who can also request this video should use another link and, thus, another file (e.g. with their own user id and another profile id).

3. **Embedded QuickTime.** Here the program generates some JavaScript code for opening a new browser popup window that contains HTML code which embeds the QuickTime player into a web page. The CGI program `embedded.cgi` is used for that purpose. The code snippet for embedding the player is shown in Listing 7.2.

**Listing 7.2: HTML code to embed the QuickTime player into a web page**

```xml
<object classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
codebase="https://www.apple.com/qttactivex/qtplugin.cab" width="402" height="338">
<param name="qtsrc" value="rtsp://127.0.0.1/xmen2.mp4?uid=3&proid=1&id=stream1" />
<param name="autoplay" value="true" />
</object>
```
Active Sessions  In addition to the list of videos on the media server, the logon.cgi program does also generate a list of currently active sessions. It uses a TCP connection to the SeMiProxy to request the list of active sessions. For every entry of the session list, the program will generate at least two links: a link for migrating this session, and a link for copying this session. In addition, if a session was paused and resumed, the program will create a link for every resume position which can be used to copy the session starting at this position. Table 7.1 exemplifies how these links look like for an active session that was resumed two times.

<table>
<thead>
<tr>
<th>Action</th>
<th>RTSP-URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>migrate:</td>
<td>rtsp://127.0.0.1/xmen2.mp4?uid=3&amp;proid=1&amp;id=stream1&amp;action=1&amp;sub=5&amp;sid=192886333</td>
</tr>
<tr>
<td>copy:</td>
<td>rtsp://127.0.0.1/xmen2.mp4?uid=3&amp;proid=1&amp;id=stream1&amp;action=2&amp;sub=5&amp;sid=192886333</td>
</tr>
<tr>
<td>copy section:</td>
<td>rtsp://127.0.0.1/xmen2.mp4?uid=3&amp;proid=1&amp;id=stream1&amp;action=3&amp;sub=5&amp;sid=192886333&amp;section=2</td>
</tr>
</tbody>
</table>

Table 7.1: Example of RTSP-URIs used for active sessions
7.7 The SeMiProxy

The SeMiProxy is the core component of the SeMiSystem. It must be started on a certain port, where it will wait for incoming connections from media players. Basically, the general task of the proxy could be described as follows:

1. Forwarding received RTSP messages to the specified media server and returning the response to the client. When a message arrives, the proxy inspects the address of the media server, opens a connection to this address and forwards the message to it.

2. Removing parameters from the RTSP-URI of the RTSP message that were added by the SeMiServer (and used by the SeMiProxy to identify the user, the profile, and the session, for instance).

Thus, the proxy acts like an interface between the media server and the streaming client. This allows the proxy to monitor and control all messages between the server and the client. By inspecting the messages it recognizes when a streaming session has been created, started, paused, resumed, or closed.

7.7.1 Profiles

The parameters added by the SeMiServer identify the end-user of a particular RTSP session and tell the proxy which profile the client wants to use for this session. Table 6.1 on page 76 shows all implemented parameters.

Due to the requested profile the proxy will add Terminal Capabilities according to the MPEG-21 DIA syntax to the body of the DESCRIBE message. This will cause the media server to transcode the corresponding media data according to the specified properties. Listing 7.3 exemplifies the resulting DESCRIBE message.
Listing 7.3: DESCRIBE message altered by the SeMiProxy

```
DESCRIBE rtsp://127.0.0.1/xmen2.mp4 RTSP/1.0
CSeq: 1
Accept: application/sdp
User-Agent: ItecMP4Player
Content-Type: application/mpeg21-dia
Content-Length: 624
+xml version="1.0" encoding="UTF-8" ?
<DIDL xmlns="urn:mpeg:mpeg21-dia: schema:2003">
<TerminalCapabilities>
  <InputOutput>
    <Display bitsPerPixel="16" colorCapable="TRUE" refreshRate="100">
      <Resolution horizontal="176" vertical="144" />
    </Display>
  </InputOutput>
  <AudioOut samplingFrequency="44100" bitsPerSample="16" numChannels="2" />
</TerminalCapabilities>
</DIDL>
```

### 7.7.2 Destination of RTP Data

Normally, a media server will convey the streaming data (e.g. RTP data) to that client which has sent the RTSP messages of a session. However, the SeMiProxy should only act as an RTSP proxy to the client and should not be concerned about RTP data. Thus, it tells the server which IP address should be used for the streaming data by adding the IP address of the client to the SETUP message. The parameter `destination` is used for that purpose. Listing 7.4 illustrates the resulting SETUP message.
Listing 7.4: SETUP message altered by the SeMiProxy

| SETUP rtsp://192.168.0.10/xmen2.mp4/track1D=1 RTSP/1.0 |
| CSeq: 2 |
| Transport: RTP/UDP; unicast; destination = 192.168.0.15; client_port=27428-27429; client_rtx_port=30794 |

In this way the streaming data will be transmitted to the address specified in the destination parameter without traversing the RTSP proxy. The proxy is only used for the transmission of RTSP messages.

Note: According to the RTSP standard a media server should not allow an untrusted/unauthenticated RTSP client to direct media streams to another address. Such a behaviour could be used for Denial-of-Service attacks (see page 59 in [1]).

7.7.3 Support of Non RTSP Proxy Capable Media Players

The SeMiProxy has an integrated support of media players that do not natively allow to use an RTSP proxy.

For such players, the SeMiServer can be configured to change the host address and the port number in the RTSP URI in order to force the media player to connect directly to the SeMiProxy. The address and the port number of the real media server are added to the RTSP URI as additional parameters.

Therefore, the proxy will look for the host and port parameter in the RTSP URI in order to use the values of those parameters for changing the RTSP message (or rather the RTSP URI) and forwarding the message to the real media server.

7.7.4 Media Time Management

When an existing video session should be migrated, the SeMiProxy has to use the correct media time for that migration. The new session should start on that media time on which it was stopped on the original client. Due to its independent concept
the proxy cannot directly access the media time of the client. It can only perform an estimation of that value. However, the proxy can try to ask the client by using the GET_PARAMETER message of RTSP. If this request fails, the proxy can furthermore try to request the media time from the media server in the same way. Finally, the proxy will only use its own estimation when the media time can be neither requested from the client nor from the server. Figure 7.3 presents the sequence diagram of a session migration. Appendix F contains the source code which is used by the SeMiProxy to estimate the media time of an active session.

**Media Time Requested from the Client** The best result relating to the media time of a migration will be reached when a media player answers the media time request from the SeMiProxy. This way, the migrated session will be exactly continued from the position that was reached on the original client - neglecting the processing time.

**Media Time Requested from the Server** If the media time could not be requested from the media player, the SeMiProxy will try to request it from the media server using the same method as described above. Of course, this value will be not as accurate as the value requested from the client. However, it will be much better than the estimation of the proxy because the media server receives RTCP feedback from the client and, thus, can derive the media time on the client.
Figure 7.3: Sequence diagram of a session migration
**Media Time Estimation** If the media time could not be requested from the media server too, the SeMiProxy will use an estimation of the media position on the client. For this estimation, the proxy uses two timestamps. The first timestamp is stored when the PLAY response message of a session is received from the server. The second timestamp is stored when this particular session should be migrated. It should be obvious that the first timestamp is not really identical to the start of the session on the client. It does rather indicate the time on which the server is going to start the transmission of the media data. Moreover, the transmission of the media data may have been already started by the server. The progress of the transmission of the RTP data will primarily depend on the network structure. The media time estimation of the proxy is highly dependent on the time difference between this timestamp and the display of the first frame on the client. The proxy does not really know when the transmission of the media data on the server has been/will be started. Furthermore, the proxy does not know the network delay between the server and the client and it will normally not know the pre-buffering time used on the client. Figure 7.4 illustrates this problem.

![Figure 7.4: SeMiProxy - media time management](image)

The pre-buffering time on the client primarily depends on the chosen media player. Fortunately, some media players announce their maximum buffer size (in seconds) in the **x-prebuffer** header field of the PLAY message. Therefore, the proxy can use this value and add it to the recorded timestamp of the RTSP PLAY response message.
from the media server in order to reduce the error in the time estimation. If a player does not announce its buffer size, the SeMiProxy will use the `bufferingDelay` value from the profile chosen by the user. However, in this way a profile can only be used in combination with one particular player - otherwise the time estimation might be completely wrong.

**Correction of the Estimation** If a session is paused by a user and resumed afterwards, the SeMiProxy can correct its media time estimation. This is possible, because the consecutive PLAY message contains the requested media time in the `Range` header field. As described below, the SeMiProxy manages a list of sections for every session which can be used to copy a session from a previously marked position\(^2\). Due to the correction, the estimated media time values of such marked positions will be very close to the actual media time on the client. However, a resumed session will be buffered on the client again. Thus, the estimation used for a further migration or copy process will diverge from the media time on the client again.

**Sections of a Session** The SeMiProxy manages every active video session as a list of time sections. A *Section* is defined as the time range between starting the session and pausing or closing it. Thus, a session may consist of only one section or can contain several sections. A client may copy the video from each start of any section by denoting the section number.

### 7.7.5 Interface to the SeMiServer

An important task of the SeMiProxy is to answer requests from the SeMiServer. Therefore, it listens on a pre-configured port for incoming requests from the SeMiServer and returns the complete list of active sessions as a result to the request. This list will be used by the SeMiServer to provide the possibility to migrate or copy a session of a user. The list contains for each active session

\(^2\)A user may mark positions by pausing and resuming a session on a particular point.
• The Session-Id of the RTSP session
• The internal SeMi-Id of the video
• The entire URI of the session
• The address of the client that started the session
• The state of the session
• The duration of the session
• The elapsed time of the session (in milliseconds)
• The number of sections for this session
• For a session which has already been migrated:
  – The Session-Id of the original session
  – The elapsed time of the original session

7.8 Classes, Files and Programs

This section is used to shortly describe the classes, files and programs of the current implementation. A more detailed description about the classes and their methods can be found at http://vitooki.sourceforge.net/docs/.

7.8.1 Classes used by the SeMiServer

This subsection describes the classes which are used by the SeMiServer. Figure 7.5 contains an overview of those classes.

CGIParser This class is used by the SeMiServer to parse parameters, submitted by an HTML form.
Movie  This class is used to store one particular video.

MovieList  This class manages the available videos. It is also responsible for loading all videos into memory.

MovieListRequest  This class is used by the SeMiServer for requesting the movie list from the media server (via HTTP) and write it to a file. In addition, the descriptive images specified in this list will be requested via HTTP and stored in a file, respectively.
SessionList  This class is used by the SeMiServer to manage a list of currently active sessions requested from the SeMiProxy.

7.8.2 Classes used by the SeMiProxy

This subsection describes the classes and files which are used by the SeMiProxy. Figure 7.6 contains an overview of those classes.

SemiProxySession  This class incorporates the core functionality of the SeMiProxy. It is implemented as a thread which is started from SemiProxy when a particular client connects.

ServerCommunication  This class is used by the SeMiProxy for the communication with the SeMiServer. It is responsible for sending a list of currently active streaming sessions to the SeMiServer.

SessionMgmt  This class is used to manage the currently active streaming sessions. Furthermore, it listens for incoming requests (from the SeMiServer) to which it will reply a list of currently active streaming sessions (by using ServerCommunication).

URIParameters  This class contains all parameters of a streaming session that were added to the RTSP-URI by the SeMiServer.
Figure 7.6: Class model of the SeMiProxy
7.8.3 Shared Classes

This subsection describes those classes which are used by both the SeMiProxy and the SeMiServer. Figures 7.7 and 7.8 give an overview of those classes.

Profile  This class is used to store a profile of a user.

ProfileList  This class manages the available profiles. It is also responsible for loading all profiles into the memory.

SemiGlobals.cpp  This file contains the implementation of two classes:

- The *Globals* class which is used to manage the configuration of the SeMiSystem.
- The *SemiDebug* class which is used for debugging purposes.

Moreover, this file contains a number of predefined constants and two namespaces:

- The *semifunc* namespace that incorporates some frequently used functions.
- The *html* namespace that is used by the SeMiServer to produce HTML formatted output.

SSession  This class contains information about an active streaming session. It is used by *SessionMgmt* (SeMiProxy) and by *SessionList* (SeMiServer).

TimeMeasurement  This class is used to measure the elapsed time of a streaming session and to manage the time-sections for a session. Furthermore, this class contains the media time estimation described on page 95.

TimeSection  This class is used to measure the elapsed time of one particular section of a streaming session. A section is defined as the time range between a PLAY and a PAUSE event.
User   This class is used to store data about a certain user.

UserList  This class manages the available users. It is also responsible for loading all user data into the memory.
Figure 7.8: Shared classes (Part-2)
7.8.4 Programs and Files

This subsection describes the programs and files which are used by the SeMiSystem.

**embedded.cpp**  This CGI program is used by the SeMiServer to generate an HTML page containing the QuickTime player as an embedded component.

**index.html**  This HTML file is used by the SeMiServer as the start page. The code contained in this file redirects to the `readusers.cpp` CGI program.

**logon.cpp**  This CGI program is used by the SeMiServer to check the logon-data of a user (username/password). Furthermore, this program will produce a list of available movies and a list of currently active streaming sessions.

**readprofiles.cpp**  This CGI program is used by the SeMiServer to provide a selection of corresponding profiles (and a password-field) for a specified user.

**readusers.cpp**  This CGI program is used by the SeMiServer to provide a selection of all available users. Moreover, this program requests the movie list from the media server using `MovieListRequest`.

**SemiProxy.cpp**  This is the main program of the SeMiProxy. It loads all user data and profiles into memory, creates a `SessionManagement` object (which listens for requests from the SeMiServer) and creates another socket, on which incoming client connections will be handled by a `SemiProxySession` thread.
The result of this thesis is the design and the implementation of a video session migration system that can be used to migrate a streamed video from one device to another. Although this system has some minor requirements on a client, it is independent of the media player and independent of the operating system at the client side. Furthermore, it is independent of the chosen video format as well. The only strong requirement for a client is the support of the Real-Time Streaming Protocol. Thus, the system may be used to migrate any RTSP-requestable media stream to any RTSP-compatible media player on any operating system. Another interesting feature is to copy an active video stream either from the beginning or from some marked positions. However, the session migration system does not synchronize a copied session with the original session.

A challenging part of the session migration system is the estimation of the media time on the client. Due to the fact that no client application is used which could request the current media time on the client from the media player (e.g. by an API call), the system cannot exactly determine the media time of an active session. It can only perform an estimation or ask the media server for its estimation which should be more accurate due to the information from the RTCP packets. It should be obvious that the local estimation of this value on the SeMiProxy may diverge from the real elapsed media time on the client. However, if a media player can provide its media time by the RTSP protocol (like the ViTooKi MuViPlayer), the SeMiSystem can
request this information and use it for a migration. Thus, the migration of a session established with such a player will exactly start at the media time that has been reached on the original client of the session.

The session migration system is currently available for the Linux operating system. It has been successfully tested with the ViTooKi MuViPlayer [7] on Linux, the QuickTime Player v6.5.2 on Microsoft Windows, and the pvPlayer v3.3 on Microsoft Pocket PC 4.20.

The current implementation of the SeMiSystem could be improved in the following ways:

- The rights for a user to start, to migrate, or to copy a videostream should be considered in the implementation.

- Instead of text files, a relational database (as described in Section 6.2) should be used to store users, profiles and user-rights.

- A web interface which allows to add, edit, or delete entries of the database could be implemented.

- Some parts of the user-profile could be determined automatically (e.g. the width, the height, and the color-depth by using JavaScript). However, this information should only be used as a suggestion to the user.

- Other link types could be added. An example would be the link to a Java applet which embeds an MPEG-4 player into a web page.

- The synchronization of the list of videos with the media server should be performed in a periodical manner.

- The system (i.e. the media server) could be enhanced to support the migration of a live-streamed video too.
Bitrate

The bitrate defines the speed at which bits are processed or transmitted and is usually expressed in bits per second (bps).

Client

“A program that establishes connections for the purpose of sending requests” [23].

Connection

“A transport layer virtual circuit established between two programs for the purpose of communication” [23].

Frame

One single image of a video is called a frame.

Framerate

The framerate defines the speed at which consecutive images of a video will be displayed. The framerate is expressed in frames per second (fps) or equivalently in hertz (Hz).
Host

A computer which has access to a network (e.g. the Internet) and which is able to communicate with other computers in this network. A computer which provides some network based service (e.g. Mail, Web) is sometimes also called a host.

IETF

The Internet Engineering Task Force (IETF) is the driving force for the development of the Internet. It is an international community, organized in several working groups, consisting of researchers, vendors, operators, and network designers, which work together in order to take care of the evolution of the Internet (e.g. define new protocols)

Media server

“The server providing playback or recording services for one or more media streams. Different media streams within a presentation may originate from different media servers. A media server may reside on the same or a different host as the web server the presentation is invoked from” [1].

Media Stream

A media stream is a consecutive set of multimedia data (e.g. an audio stream or a video stream), which is received from a media server over a network and immediately processed at the client side.

Media Player

Some piece of software at the client side, which is able to process and display video files.
MIME

The *Multipurpose Internet Mail Extensions* (MIME) were defined to extend the Internet Mail Protocol for the purpose of transmitting non-ASCII content too. This, for instance, enables the transmission of images, audio files, or video files attached to an email. MIME was specified in RFC 1522. Nowadays, there do exist a lot of MIME types for almost any kind of file format which are not only used for mail purpose. Also a web browser uses MIME types for interpreting data, received from a web server.

Port

“The abstraction that transport protocols use to distinguish among multiple destinations within a given host computer. TCP/IP protocols identify ports using small positive integers” [50].

Proxy

“An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers” [23].

Router

A router is a network device, connected to at least two packet-switched networks. It forwards incoming packets towards their destination in a process called routing.

Server

“An application program that accepts connections in order to service requests by sending back responses” [23].
Session Announcement

“A session announcement is a mechanism by which a session description is conveyed to users in a proactive fashion, i.e., the session description was not explicitly requested by the user” [2].

Session Description

“A well defined format for conveying sufficient information to discover and participate in a multimedia session” [2].

Session / Streaming Session

“A multimedia session is a set of multimedia senders and receivers and the data streams flowing from senders to receivers. A multimedia conference is an example of a multimedia session” [2].

Time To Live (TTL)

A field specified in the Internet Protocol (IP) is called *Time To Live* (TTL). The value of this field defines how long routers will forward IP packets within the network. When a router forwards a packet, it decrements the TTL value of the packet. When a router receives a packet already having a TTL value of zero, it destroys the packet (i.e., it will not be forwarded).

Transcoding

*Transcoding* means the conversion from one encoding format to another.

URI

URI is the abbreviation for *Uniform Resource Identifier*. A URI is an identifier with a well-defined syntax that is used to address resources. URIs are a superset of the
Uniform Resource Locators (URLs) which are used to address websites.

**Video on Demand (VoD)**

*Video on Demand* denotes an interactive television system, which allows users to start watching a chosen video, whenever they want. When the user has chosen a video, the server immediately starts the streaming for that video to the client. A VoD system normally uses unicast transmission. In addition to VoD, there exist *Near Video on Demand* (NVOD) systems, which allow a user to start watching a video with a small delay. This means, a NVOD server transmits popular videos at short intervals using multicast.

**XML**

The *eXtensible Markup Language* (XML) is a standardized language which provides a very flexible way to create a structured description about something in a human readable format. It is similar to the HyperText Markup Language (HTML) and is a subset of the Standard Generalized Markup Language (SGML). Nowadays, XML is very often used by business application for the purpose of data-exchange.
Some Details about RTSP Header Fields

This chapter describes a small portion of RTSP header fields. For a complete reference, the interested reader is referred to RFC 2326 [1].

**Content-Length** This field is used to specify the length of the message body in bytes. If no content length has been specified, a default value of zero is assumed.

**CSeq** The CSeq header field contains a sequence number for the RTSP message which correlates with the current connection and must be contained in every message. The sequence number starts with the value one and is increased with every new message. The receiver of the message responds to a request with the same sequence number as received. If a message has to be retransmitted, the original sequence number must be used.

**Range** When included in a request message, the Range header field defines the media time section which should be played or recorded. If a response message contains a Range header field, it indicates the actual played or recorded time section. The time interval can be either open (e.g. “0-”) or closed (e.g. “5-15”). There are several possible time formats: SMPTE, NPT or Absolute Time (clocks). An SMPTE relative timestamp contains the time relative to the start of the clip. Table B.1 shows the syntax for such a timestamp. Sub-frames are measured in a one-hundredth of a frame. An NPT (Normal Play Time) timestamp contains the absolute time, relative to the
APPENDIX B. SOME DETAILS ABOUT RTSP HEADER FIELDS

Syntax:  hours:minutes:seconds:frames.subframes
Examples:  smpte=10:12:33:20-
          smpte=10:07:33-10:07:22:05.01

Table B.1: Syntax of an SMPTE time range

beginning of the presentation. Its format is denoted in Table B.2. As illustrated in the example of the table, the special constant now can be used for identifying the current time and may be useful for live streaming sessions.

Syntax:  hours:minutes:seconds.secFraction or seconds.secFraction
Examples:  npt=02:30:04.2-
          npt=87.32-97
          npt=now-

Table B.2: Syntax of an NPT time range

The third possibility is the use of an ISO 8601 timestamp which expresses the absolute time using UTC (GMT). This format consists of a date value followed by a time value which may also contain fractions of a second after a dot. Table B.3 shows the syntax of such a timestamp.

Syntax:  utcDate “T” utcTime “Z”
Example:  20040930T153721.75Z
          (September 30, 2004 at 15h37m21s and a three-quarter second)

Table B.3: Syntax of an ISO 8601 time stamp

Note:  A simple playback server without recording functionality must only support the NPT time format. (see [1], p34)

Session  As already mentioned earlier, RTSP messages normally belong to a particular session which has been previously created on the server as the result of a SETUP message. In response to the SETUP message, the server appends the Session Id of the new session. The client uses this Session Id for further messages and
adds it to the message header. In this fashion, the server can assign the message to the correct session.

**Transport** The Transport header field is used in the SETUP request to set the transport parameters which should be used by the server. The general syntax for this header field is `transport/profile/lower-transport`, where:

- `transport` is the transport protocol to use (e.g. RTP)
- `profile` is the chosen profile (e.g. AVP)
- `lower-transport` is the lower-level transport protocol (e.g. UDP)

The Transport header field may contain some other parameters, listed in Table B.4.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unicast</td>
<td>identifies a unicast transmission</td>
</tr>
<tr>
<td>multicast</td>
<td>identifies a multicast transmission (default)</td>
</tr>
<tr>
<td>destination</td>
<td>specifies the address to which the media data should be sent</td>
</tr>
<tr>
<td>mode</td>
<td>specifies the requested mode (PLAY (default) or RECORD)</td>
</tr>
<tr>
<td>ttl</td>
<td>defines the time-to-live value for a multicast session</td>
</tr>
<tr>
<td>port</td>
<td>specifies the RTP/RTCP port pair for a multicast session</td>
</tr>
<tr>
<td>client_port</td>
<td>provides the RTP/RTCP port pair, which a client has chosen for a unicast session</td>
</tr>
<tr>
<td>server_port</td>
<td>specifies the RTP/RTCP port pair for a unicast session on which the server listens for media data and control information</td>
</tr>
</tbody>
</table>

Table B.4: Parameters of the *Transport* header field of RTSP
This chapter gives a short overview of the Session Description Protocol, which is used to describe multimedia sessions. The SDP is used by RTSP in the DESCRIBE and ANNOUNCE messages.

C.1 Introduction

SDP is a text-based protocol, which was designed to describe multimedia sessions. The purpose of SDP is to convey information about media streams of a multimedia session which can be used by the recipients to participate in the session. However, SDP is not intended to be used for negotiation of multimedia encodings. SDP is purely text-based, using the ISO 10646 character set with UTF-8 encoding [24].

Note: If SDP needs to be used in cooperation with Internet-based applications like Email or WWW, the MIME type “application/sdp” should be used for that purpose.

C.2 Specification

As it is apparent from Table C.1, a common SDP description consists of a session description, a time description, and several media descriptions (optional). More
precisely, each part comprises a particular number of lines, separated by CRLF. Each of them contains an assignment of the following format: \texttt{Type=Value}

\textbf{Type} is exactly one character which is case-sensitive. \textbf{Value} is a text string with a format that depends on the value of \textit{Type}. There is no whitespace\textsuperscript{1} admitted on either side of the equality sign. Table C.1 gives an overview of all allowed type specifiers.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>v=</td>
<td>SDP protocol version</td>
<td>x</td>
</tr>
<tr>
<td>o=</td>
<td>owner/creator and session identifier</td>
<td>x</td>
</tr>
<tr>
<td>s=</td>
<td>session name</td>
<td>x</td>
</tr>
<tr>
<td>i=</td>
<td>session information</td>
<td></td>
</tr>
<tr>
<td>u=</td>
<td>URI of description</td>
<td></td>
</tr>
<tr>
<td>e=</td>
<td>email address</td>
<td></td>
</tr>
<tr>
<td>p=</td>
<td>phone number</td>
<td></td>
</tr>
<tr>
<td>c=</td>
<td>connection information</td>
<td></td>
</tr>
<tr>
<td>b=</td>
<td>bandwidth information</td>
<td></td>
</tr>
<tr>
<td>z=</td>
<td>time zone adjustments</td>
<td></td>
</tr>
<tr>
<td>k=</td>
<td>encryption key</td>
<td></td>
</tr>
<tr>
<td>a=</td>
<td>session attributes</td>
<td></td>
</tr>
<tr>
<td>t=</td>
<td>time the session is active</td>
<td>x</td>
</tr>
<tr>
<td>r=</td>
<td>repeat times</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>m=</td>
<td>media name and transport address</td>
<td>x</td>
</tr>
<tr>
<td>i=</td>
<td>media title</td>
<td></td>
</tr>
<tr>
<td>c=</td>
<td>connection information (only optional if included in session description above)</td>
<td></td>
</tr>
<tr>
<td>b=</td>
<td>bandwidth information</td>
<td></td>
</tr>
<tr>
<td>k=</td>
<td>encryption key</td>
<td></td>
</tr>
<tr>
<td>a=</td>
<td>attribute lines</td>
<td></td>
</tr>
</tbody>
</table>

Table C.1: SDP type specifiers [2]

\textbf{Note:} In an SDP session description, the sequence of the type specifiers shall be exactly the same as defined in Table C.1. Furthermore, all specified types in the part of the session description are also used for types in the part of the media description unless overridden.

\textsuperscript{1}A character representing a voidage is called a \textit{whitespace}. Such a character is the Space, the Tab, and sometimes even a line break.
This section continues with a very short overview of all type specifiers, defined by the Session Description Protocol. More information about SDP and these specifiers can be found in [2].

**SDP Protocol Version (v=)** This specifier defines the SDP-version used in the session description. The version number starts with the value “0”.

**Origin (o=)** This specifier defines the originator of the session. The meaning of the fields for this specifier, apparent in Figure C.1, is defined as follows:

- **username** is the login-name of the user, which has started the session. If no username is available, this field can also contain a “-”

- **sessionId** is a unique identification number for this session. It is suggested to use a Network Time Protocol (NTP) timestamp (see [51]).

- **version** is the version number for this SDP announcement.

- The next three fields are used to specify the address from which the session was originally created.
  - For **networkType** the value “IN” is currently defined, which means “Internet”.
  - For **addressType** the values “IP4” and “IP6” are possible.
  - Finally, **address** must contain the globally unique address of the host, which can be either an IP-address or a fully specified domain-name.

<table>
<thead>
<tr>
<th>o=</th>
<th>Username</th>
<th>SP</th>
<th>SessionId</th>
<th>SP</th>
<th>Version</th>
<th>SP</th>
<th>NetworkType</th>
<th>SP</th>
<th>AddressType</th>
<th>SP</th>
<th>Address</th>
</tr>
</thead>
</table>

Figure C.1: Syntax of the origin specifier of SDP (SP=space)
Session Name (s=)  This specifier defines the name of the session.

Session and Media Information (i=)  This specifier is used to provide a description of the session.

URI (u=)  This specifier provides a URI for additional information about the session.

EmailAddress (e=)  This specifier contains the email address of the person who is responsible for the session.

Phone Number (p=)  This specifier contains a phone number of the person, who is responsible for the session.

Connection Data (c=)  This specifier defines the connection for joining the session. Figure C.2 shows the syntax of the Connection Data specifier. The syntax and the meaning of the fields is the same as already mentioned in the Origin specifier. When a multicast session is used, the address field has to contain a Time-To-Live (TTL) field as well, which is appended to the address after a slash (/).

<table>
<thead>
<tr>
<th>c=</th>
<th>Network Type</th>
<th>SP</th>
<th>Address Type</th>
<th>SP</th>
<th>Connection Address</th>
</tr>
</thead>
</table>

Figure C.2: Syntax of the connection data specifier of SDP (SP=space)

Bandwidth (b=)  This specifier provides the bandwidth in kbps, which is required for the session. Figure C.3 contains the syntax of the Bandwidth specifier. The field Modifier contains the meaning of the bandwidth value.

<table>
<thead>
<tr>
<th>b=</th>
<th>Modifier</th>
<th>$\text{rate}$</th>
<th>Bandwidth Value</th>
</tr>
</thead>
</table>

Figure C.3: Syntax of the bandwidth specifier of SDP
Session Start/Stop Time (t=) This specifier denotes the time, when the session is active and defines the duration of a session. Figure C.4 illustrates the syntax of this specifier. StartTime contains the NTP start time in seconds. If the start time is zero, the session is assumed to be permanently active. StopTime contains the NTP stop time in seconds and can also be zero. It is also possible to have many “t=” specifiers in a SDP session description. This option should only be used for irregular session times because there exists also a “r=” specifier, for regular session times.

<table>
<thead>
<tr>
<th>t=</th>
<th>StartTime</th>
<th>SP</th>
<th>StopTime</th>
</tr>
</thead>
</table>

Figure C.4: Syntax of the session start/stop time specifier of SDP (SP=space)

Repeat Times (r=) This specifier defines the repeat times for a session. Figure C.5 shows the syntax of the Repeat Times specifier. All fields are specified in seconds using the NTP time format. RepeatInterval defines the interval (or distance) between two sessions while ActiveDuration defines the duration of a subsequent session to be active. The list of offsets from start is used to specify several repeat times whereby each offset-value is added to the start time.

<table>
<thead>
<tr>
<th>r=</th>
<th>RepeatInterval</th>
<th>SP</th>
<th>ActiveDuration</th>
<th>SP</th>
<th>List of offsets from start</th>
</tr>
</thead>
</table>

Figure C.5: Syntax of the repeat times specifier of SDP (SP=space)

Encryption Keys (k=) If encryption of data is necessary for a session, this specifier can be used to convey the required information - e.g. encryption method and encryption key. Figure C.6 contains the syntax of the Encryption Keys specifier. For details see [2] and [20].

<table>
<thead>
<tr>
<th>k=</th>
<th>Method</th>
<th>&quot;#&quot;</th>
<th>Encryption Key</th>
<th>(optional)</th>
</tr>
</thead>
</table>

Figure C.6: Syntax of the encryption keys specifier of SDP
Attributes (a=)  This specifier can be used to extend SDP. Figure C.7 shows the syntax of the Attributes specifier. The value field is optional; if no value is specified, the attribute is used as a flag (which is set when the “a=” specifier with this flag is contained in the SDP session description). A list of suggested attributes can be found at p23-27 in [2].

```
c=  Attribute  n=  Value
   (optional)
```

Figure C.7: Syntax of the attributes specifier of SDP

Media Announcements (m=)  This specifier is used to convey information about the media tracks of the session. Figure C.8 illustrates the syntax of the Media Announcements specifier.

```
m=  Media  SP  Port  SP  Transport  SP  Format  List
```

Figure C.8: Syntax of the media announcements specifier of SDP

The meaning of the fields for this specifier is defined as follows. Media can be “audio”, “video”, “application”, “data”, and “control”. Port defines the port to which the media stream will be sent. If several ports are necessary, the number of ports is appended to the value for the port after a slash (/). The value of the Transport field depends on the address type of the “cs=” specifier and does usually contain one of these values:

- RTP/AVP - Real-time Transport Protocol used with Audio/Video profile [20].
- udp - User Datagram Protocol

The second value is used if an application uses a proprietary media format and/or transport protocol.
Note: If RTP is used as a transport protocol and if there are several ports required for the media track, only even ports are used for RTP data, because uneven ports are used for RTCP. For more details on RTP/RTCP see [17]

The FormatList field is used to specify used media formats which should be defined in the used profile (like AVP).

### C.3 SDP Example

Figure C.9 illustrates an example of a Session Description Protocol file.

```
v=0
c=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 31
m=application 32416 udp wb
a=orient:portrait
```

Figure C.9: Example of a SDP file
D.1 The Video List Conforming to MPEG-21 DID

This section contains an example file of the video list which is created from the muviserver on every startup.

Listing D.1: List of videos corresponding to the MPEG-21 DID language

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file was created by muviserver on Mon Nov 8 21:38:01 2004 -->
<?xml-stylesheet type="text/xsl" href="MovieList.xsl" ?>
<DIDL xmlns="urn:mpeg:mpeg21:2002:01-DIDL-NS"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Container>
    <Item>
      <Choice minSelections="0" maxSelections="1">
        <Descriptor>
          <Statement mimeType="text/plain">
            If you want to start a videostreaming, please select one of these available streams:
          </Statement>
        </Descriptor>
        <Selection select_id="stream1">
          <Descriptor>
            <Statement mimeType="text/plain">xmen2.mp4</Statement>
          </Descriptor>
          <Descriptor>
            <Statement mimeType="text/plain">No description available</Statement>
          </Descriptor>
          <Component>
            <Resource mimeType="image/jpeg" ref="xmen2.jpg"/>
          </Component>
        </Selection>
      </Choice>
    </Item>
  </Container>
</DIDL>
```
<Selection select_id="stream2">
  <Descriptor>
    <Statement mimeType="text/plain">Directed by: Peter Jackson; Writing credits: J.R.R. Tolkien; Actors: Noel Appleby, Alexandra Astin, Sean Astin, John Bach, Seon Bean, Orlando Bloom; User Rating: 8.9/10 (57,631 votes)</Statement>
  </Descriptor>
</Selection>

<Selection select_id="stream3">
  <Descriptor>
    <Statement mimeType="text/plain">No description available</Statement>
  </Descriptor>
</Selection>

</Choice>

</Item>
</Container>
</DIDL>
D.2 The XML Style Sheet for the Video List

This section contains a listing of the XML style sheet, developed with this thesis in order to be used for transforming the video list (shown above) to an HTML file.

Listing D.2: An XML style sheet used for the transformation of the video list

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="urn:mpeg:mpeg21:2002:01-DIDL-NS"
    xmlns:did="urn:mpeg:mpeg21:2002:01-DIDL-NS">
    <xsl:output method="html"/>
    <xsl:variable name="lowercase" select="'abcdefghijklmnopqrstuvwxyz'"/>
    <xsl:variable name="uppercase" select="'ABCDEFGHIJKLMNOPQRSTUVWXYZ'"/>
    <xsl:template match="/"
        <html>
        <head>
            <title>muviserver - list of available movies, (c) by KS</title>
        </head>
        <style type="text/css">
            td {
                font-family:Verdana;
                font-size:8pt;
            }
            body {
                font-family:Verdana;
                font-size:8pt;
            }
            font {
                color:darkblue;
                font-weight:bold;
                font-size:10pt;
            }
        </style>
        <body>
            <h3>muviserver - list of available movies</h3>
            <hr size="1"/>
            <xsl:apply-templates/>
            <hr size="1"/>
            <br/>
            <font size="1">
                Created by Klaus Schoeffmann,
                University of Klagenfurt – ITEC, 2004
            </font>
        </body>
        </html>
    </xsl:template>
    <xsl:template match="/did:Item[1]">
```
<br/>
<table>
<tr>
<td colspan="2">
<br/>
</td>
</tr>
<tr>
<td colspan="2">
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
</xsl:template>
<xsl:template name="getResourceLinkByCondition">
  <xsl:param name="nodes"/>
  <xsl:param name="id"/>
  <xsl:for-each select="$nodes">
    <xsl:if test="./did:Condition/@require=$id">
      <xsl:value-of select="./did:Resource/@ref"/>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
This appendix gives a short summary of how to install Linux on an iPAQ device. Linux for iPAQ is available from the website of the Familiar Project (see www.handhelds.org) for the iPAQ H3100, H3600, H3700, H3800, H3900, H5400, and H5500 series. This summary applies to the iPAQ H3630.

In order to install Linux, firstly a boot loader like the Compaq bootldr, which is able to boot the device with Windows CE or Linux, has to be installed. The installation is very straightforward:

1. Connect the iPAQ with your PC, using the serial cradle.

2. Copy bootldr and BootBlaster to the iPAQ (e.g. with ActiveSync)

3. Save your current boot loader and the Windows CE ROM using BootBlaster and copy the saved files to your PC (Menu: ”Flash” → ”Save Bootldr .gz Format”, ”Save Wince .gz Format”). This is very important, because otherwise you will not be able to reinstall Windows CE.

4. Use BootBlaster to install the bootldr (Menu: ”Flash” → ”Program”).

You can download both files from www.handhelds.org. For this thesis, BootBlaster_1.18.exe and bootldr-2.18.01.bin were used.
The device is now ready for the installation of Familiar Linux:

1. Download Familiar Linux from www.handhelds.org. There are a number of tarballs which differ from a blank textual system (bootstrap), to a graphical interface (e.g. Opie).

2. Press the joypad of the iPAQ and push the reset button (on the bottom of the device). When the boot loader appears, release the joypad and press the calendar button on the iPAQ.

3. Use a terminal emulator (e.g. HyperTerminal, shipped with Windows) to connect to the iPAQ. The following configuration shall be used for the serial connection:

   - 115200 bits per second
   - 8 data bits
   - no parity
   - 1 stop bit
   - no flow control (no hardware handshaking)

   If your PC is properly connected with the iPAQ, the boot> prompt is shown. You might have to press ENTER, to get the prompt.

4. Several protocols can be used for transferring data to the iPAQ: XModem, YModem, and ZModem. Specify the use of YModem, by typing “set ymodem 1”.

5. Type “load root”. After this command, the iPAQ will be ready to receive data.

6. Transfer the .jffs2 file, from the tarball downloaded earlier (“Transfer” → ”Send File”). Assure to use YModem as a protocol. The transfer will take a while.
7. When the transfer has been successfully finished, type “boot” to reboot the iPAQ.

8. You can now connect to the Linux on the iPAQ using a terminal emulator program. A username and a password must be entered. The default is: user root, password rootme.

Furthermore, you can create a Point-to-Point (PPP) connection to your Linux on the iPAQ. The following section will describe how this can be accomplished by using Windows XP:

1. Use a terminal emulator to log on to your Linux on the iPAQ.

2. Edit /etc/hosts and add the following two lines (of course, you can use any valid IP address of your network):
   
   192.168.0.3 ipaq
   192.168.0.4 desktop
   
   These addresses will be used for setting up the PPP connection.

3. Edit /etc/ppp/options and add the following two lines:
   
   ipaq:desktop
   connect ’/usr/sbin/chat -v CLIENT CLIENTSERVER’

4. Now create a new direct connection (“advanced connection”) for the serial port of your Windows XP that is connected with your iPAQ (e.g. COM1). Use the PC as a guest. Configure your connection with the following values:

   • 115200 bits per second
   • no flow control, no modem error control, no modem compression
   • show terminal window, no modem speaker
In the *Network* tab, use *PPP* as server type. Deselect the use of a *default gateway on remote network* on the *TCP/IP* settings.

5. When establishing the connection, a terminal window is displayed:

   **ATE1**

   **PASSWORD:**

   Press ENTER and type **ppp** as login name (lower case!), followed by ENTER.

You should now be able to use tools like Secure-Shell (SSH), Secure-Copy, with your Linux on the iPAQ (IP: 192.168.0.3). The following procedure describes how to install the *opie mediaplayer 2* on *Familiar Linux 0.7.2*:

- Visit [http://opie.handhelds.org/](http://opie.handhelds.org/) and download the following files:
  
  - `opie-mediaplayer2_1.0.3_arm.ipk`
  - `opie-mediaplayer2-codecs_0.7-.3_arm.ipk`
  - `opie-mediaplayer2-skin-default_1.0.2_arm.ipk`

- Copy the downloaded files to your iPAQ (e.g. to `/mnt/ramfs`).

- Connect to the *Familiar Linux* system on your iPAQ and install the OpiePlayer 2 by entering the following command:

```
ipkg -force-overwrite -force-reinstall
   opie-mediaplayer2_1.0.3_arm.ipk
   opie-mediaplayer2-codecs_0.7-.3_arm.ipk
   opie-mediaplayer2-skin-default_1.0.2_arm.ipk
   -force-overwrite
```

- Reset your iPAQ.
This appendix shows the source code that is used by the SeMiProxy in order to estimate the media time of an active session (Listing F.1).

Listing F.1: Source code used by the SeMiProxy to estimate the media time

```cpp
ulong TimeMeasurement::getElapsedTime(uint sectionNr,
                                         ulong currserverPlayout, ulong currclientPlayout)
{
    // elapsed time, considering all time-sections
    long tmElapsed = 0;

    // cumulative client-received
    this value contains the duration of all time-sections, which
    were confirmed by the client (by consecutive PLAYs)*/
    ulong cumClientReceived = 0;

    /* if a section-number has been specified, do only iterate
       until the specified section-numbers*/
    uint untilSection = sections.size();
    if (sectionNr != 0) untilSection = sectionNr;

    // iterate through sections
    vector<TimeSection*>::const_iterator iter = sections.begin();
    for (uint i =1; iter != sections.end() && i <= untilSection; iter++, i++)
    {
        TimeSection *ts = *iter;
        long tsElapsed; // elapsed time of current time-section

        if (ts->getMeasuredStarttime() == 0) {
            cerr << "\nWARNING: getmeasuredstarttime of TS " "i " is 0!";
        } else {
            /* calculate elapsed time of current time-section
```
if we do not have an endtime yet, use current time as end-time

if (ts->getMeasuredEndtime() != 0) {
} else {
    tsElapsed = ts->getCurrentMsecs() - ts->getMeasuredStartTime() - ts->getPrebufferTime();
}

// if session has not been started already, correct value to null
if (tsElapsed < 0) tsElapsed = 0;

/* if we have a value confirmed by the player for the latest section
   (which has been requested by GETPARAMETER), use this value*/
if (currclientPlayout != 0 && ((iter+1) == sections.end())) {
    tmElapsed = currclientPlayout;
} else if (ts->getClientReceived() != 0) {
    /* if the range of the current timesection has been confirmed
       by the player on PAUSE (by the extraction
       of the range value from the PLAY message), use this value*/
    tmElapsed += ts->getClientReceived();
    cumClientReceived += ts->getClientReceived();
} else if (currserverPlayout != 0 && ((iter+1) == sections.end())) {
    /* if we have a value confirmed by the server for the latest section
       (which has been requested by GETPARAMETER), use it
       BUT: if we have a value confirmed by the player for the last section
       (by extraction of range value), correct the value of the server!*/
    if (i > 1 && cumClientReceived != 0) {
        tmElapsed = cumClientReceived + (currserverPlayout - *(iter-1))->getServerPlayout());
        // do not use this value for later sections
        cumClientReceived = 0;
    } else {
        tmElapsed = currserverPlayout;
    }
} else {
    if (ts->getClientPlayout() != 0) {
        /* if we have a value confirmed from the client by GETPARAMETER
           for the current time section, use it (note: this value is cumulative!); otherwise use calculation above*/
        tmElapsed += ts->getClientPlayout();
    }
}
else if (ts->getServerPlayout() != 0) {
    /* if we have a value confirmed from the client by GETPARAMETER
       for the current time section, use it (note: this value is cumulative!);
       otherwise use calculation above */
    BUT: if we have a previous section, which
    were confirmed by the client (by extraction of range value),
    add difference to that confirmed values/
    if (i > 1 && cumClientReceived != 0) {
        tmElapsed = cumClientReceived + (ts->getServerPlayout()
            - (*iter-1)->getServerPlayout());
        // do not use this value for later sections
        cumClientReceived = 0;
    } else {
        tmElapsed = ts->getServerPlayout();
    }
} else {
    /* if we do not have a value confirmed by the player nor by the server,
       use the difference of the timestamp (minus the buffer size)*/
    tmElapsed += tsElapsed;
}

// add duration of previous session (if this session is a migrated session)
return tmElapsed + previousSessionDuration;
Bibliography


All specified URLs were checked for validness on 10th of January 2005.