HOGESCHOOL GENT
LID VAN DE ASSOCIATIE UNIVERSITEIT GENT

Academiejaar 2007–2008

Departement Toegepaste Ingenieurswetenschappen
Schoonmeersstraat 52, 9000 Gent

Development of a Streaming Video Platform for Educational Purposes

Eindwerk voorgedragen tot het behalen van het diploma van
MASTER IN DE INDUSTRIËLE WETENSCHAPPEN: INFORMATICA

Tim DE PAUW

Promotoren: Dr. Marleen DENERT
Dr. Frederik QUESTIER

PREFACE

My sincere gratitude is extended to my thesis promoters, Dr. Marleen Denert of Hogeschool Gent and Dr. Frederik Questier of Vrije Universiteit Brussel, for the time they devoted to proofreading this document and providing me with a lot of useful feedback.

I would also like to thank Frederik Questier, Inez Hoeijmakers, Roel Neefs, Thijs Braem, Bruno Dooms, Stijn Van Achter and Thea Derks, all of the Onderwijsvernieuwing & OnderwijsServiceCentrum, for their much appreciated input over the course of our pleasant collaboration.

Finally, my thanks go out to all those who have supported me and continue to do so. You know who you are.

Tim De Pauw
Ghent, June 2008

ABSTRACT

ENGLISH. This document outlines the development cycle of a streaming video platform for Vrije Universiteit Brussel. The purpose of the platform is to allow teachers to share video material with their students via a Web application. Similar functionality is already offered by many modern Web sites. What sets this platform apart from those sites is that it is based on free and open source software and integrated with the e-learning environment Dokeos, deployed by the university. This integration is optional: any PHP-based Web site can embrace the developed library set. A proof-of-concept Web application is also supplied as a starting point for third-party applications. The platform converts user-supplied video files to Ogg Theora; to accommodate bandwidth limitations, several quality settings may be used in parallel. Other than PHP, the platform uses a client-side Java utility for uploading new video files, and a Java applet for playback.

KEYWORDS. streaming video, video sharing, Ogg, Vorbis, Theora, Linux, Debian, PHP, Perl, MySQL, Java, Web application, Web service, SOAP, open source, Dokeos, LCMS, learning object, education, pedagogy

NEDERLANDS. In dit document wordt het ontwikkelingsproces beschreven van een streaming-videoplatform voor de Vrije Universiteit Brussel. Dit platform voorziet onderwijzend personeel in de mogelijkheid om via een webapplicatie videomateriaal te delen met studenten. Verscheidene moderne websites bieden reeds dergelijke functionaliteit aan. Het platform onderscheidt zich hiervan door uitsluitend te steunen op vrije open-sourcesoftware en door de integratie met de Dokeos-leeromgeving, waar o.a. de universiteit gebruik van maakt. Het resultaat kan echter worden opgenomen in een arbitraire PHP-applicatie. Bovendien werd een voorbeeldwebapplicatie ontwikkeld, die dienst kan doen als vertrekpunt voor nieuwe implementaties. Het platform converteert aangebrachte videobestanden naar Ogg Theora-formaat. Om tegemoet te komen aan bandbreedtebeperkingen, kunnen meerdere kwaliteitsinstellingen parallel worden toegepast. Verder wordt een Java-applicatie ingezet om bestanden op te laden en een Java-applet voor het afspelen.

TREFWOORDEN. videostreaming, videosharing, open source, Ogg, Vorbis, Theora, Linux, Debian, PHP, Perl, MySQL, Java, webapplicatie, webservice, SOAP, Dokeos, LCMS, leerobject, onderwijs, pedagogie

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Education has come a long way since the wax tablets of ancient Rome. Modern technology in all its forms is revolutionizing today's learning process.

To a great extent, the Internet gives way to electronic learning. Web-based *e-learning* environments enable instant access to a myriad of relevant *learning objects* to all parties involved. These learning objects may range from simple announcements to full-fledged discussion forums or learning paths.

With broadband Internet access becoming commonplace, one novelty that the Web has seen over the past couple of years is the introduction of one-click *video sharing*. The ever so popular Web site *YouTube* is rife with often nonsensical digital video clips and continues to spur a variety of clones.

Evidently, that same video sharing technology can be harnessed to enrich e-learning environments. Teachers may provide video clips that clarify a single aspect of one or more courses, but also a recording of one of their lectures in its integrity, for instance.

This document describes the project which aims to provide *Vrije Universiteit Brussel* with a *streaming video platform*, managed by its *Onderwijsvernieuwing & OnderwijsServiceCentrum (OSC)*.

Prior to the launch of this project, a rudimentary solution was in place, which required a great deal of manual effort. A teacher would provide the raw material to the OSC personnel. Subsequently, each piece would be converted and published at a URL, which would then be passed to the teacher and subsequently to students. This tedious process is probably the main reason of the infrastructure's limited popularity.

With free-of-charge video sharing Web sites on the rise, one might consider abandoning the existing setup altogether and favoring the use of YouTube, for instance. However, this raises a number of concerns. Most of the Web sites are supported by advertisements. In addition, many of them offer additional features which may add to the entertainment value, but are highly likely to have a negative impact on the learning process. There is also the matter of ownership and access rights: what if a video clip's target audience is restricted to a certain group of students? Finally, material used for education purposes can be made exempt from certain copyright restrictions, but only if maintained internally.

Thus, it is certainly favorable to rely on an in-house video sharing solution. The first goal of this project is therefore to thoroughly streamline the video sharing process. Manual intervention by the OSC personnel will be avoided where possible, allowing for a large-scale deployment. The intention is to achieve all this using solely free and open source software, without compromising usability.

The project's second, equally important goal is to ensure tight integration with *PointCarré*, the university's adaptation of the open source e-learning environment *Dokeos*. A variety of other respected institutions, both national and international, also deploy Dokeos-based solutions. As such, the results of this project will not be limited to the Vrije Universiteit Brussel infrastructure.

Moreover, the development of generic libraries and supporting tools will allow for deployment separate from a Dokeos environment. The structure of this document reflects this. First, we will discuss the development of a generic streaming video platform, as well as a proof-of-concept demonstration portal. Subsequently, the supporting tool set and libraries will be integrated with the Dokeos environment.

2 MEDIA FILES

In this chapter, we will cover the general structure of a media file. While the focus will lie on video files, most of the information is also applicable to audio material.

Portions of the information presented in this chapter are based on *Wikipedia* [1]. While citing the online encyclopedia in academic research is generally frowned upon, one could argue that Wikipedia is an acceptable source for an introductory chapter such as this one. Relevant articles are referenced throughout the chapter; in addition, the motivated reader is invited to examine linked articles not explicitly referenced.

2.1 General Model

Even though we exclusively mention *video* files, moving pictures alone will not be satisfactory, for they are usually accompanied by sound. Moreover, this sound may occur in multiple versions; for instance, consider a dubbed film. When dealing with an extensive piece, one may want to mark individual chapters. Furthermore, subtitles may be a part of it. Finally, it may be useful to include a variety of other specifics under the form of metadata, such as the author of the piece, the publication year, copyright information, etc.

Therefore, we cannot limit ourselves to storing merely visual material. Because of this, *containers* [2] are used. Containers literally *contain* several media *streams* and as such act as a sort of packaging for them. Thus, your average video file is actually a container holding a video and an audio stream.

Obviously, there needs to be some sort of structure for both the container and its streams. This task may seem relatively simple, but a number of aspects in fact make it increasingly difficult.

2.2 Terminology

Audio and video *streams* are formatted using *codecs* [3]. This term is a contraction of the words *cod*er and *dec*oder, or alternatively *co*mpressor and *dec*ompressor. Thus, it describes the software used to both store and interpret stream data.

Codecs may be either *lossy* or *lossless*. *Lossy* codecs are able to shrink down a stream significantly by allowing the quality of the material to decrease. The quality that is eventually attained depends on the algorithms used by the codec. *Lossless* codecs on the other hand maintain the original quality of the material, but attempt to compress it as efficiently as possible nonetheless. Obviously, if a stream is compressed using a lossless codec, it will use considerably more memory than with a lossy one.

The encoding process divides the original stream into fixed time units. Each of these is compressed into a certain amount of memory. The relation between the two is defined as the *bit rate*. It is generally expressed in bits per second; 'kilobits per second' is commonly abbreviated to *kbps*.

A stream's bit rate may be *constant (CBR)* or *variable (VBR)*. CBR streams maintain the same bit rate throughout the entire stream. VBR encoders empirically decide on an amount of bits for each time unit, in order to maintain a certain quality level. Consequently, to give an indication of a VBR stream's overall quality, its average bit rate is usually calculated.

Note that the bit rate is different from the *sampling rate*. When creating a digital representation of a real-world signal, which is analog, the process of *sampling* records the signal's data at fixed time intervals in order to approximate it. The frequency at which this occurs is known as the sampling rate. Evidently, the more samples, the higher the accuracy of the resulting recording, but the larger the file becomes. Audio material is often sampled at 32, 44.1 or 48 kHz. Video material generally has a sampling rate of 50 Hz or 60 Hz; this is commonly referred to as the *frame rate*, expressed in frames per second.

A lot of codecs are in circulation these days. They may be distinguished based on the following aspects:

- **Quality vs. memory usage.** Evidently, the objective of encoding is to obtain the highest possible media quality for the lowest amount of memory. Whereas determining the quality of media can be a subjective process, the bit rate is incontestably set. Most codecs only support a limited set of bit rates.
- **Speed.** A codec that takes an extensive amount of time to encode or, more importantly, decode a stream often will not be a good choice, even if it delivers superior picture or sound quality.
- **Compatibility.** Sadly, not one media player or platform supports (or bundles) every possible codec. If one aims to reach a large audience, this should be taken into consideration.
- **Commercial model.** Not all codecs are free to use. Often, however, a free alternative is available.

2.3 Audio Codecs

We will now look at the most commonly used audio codecs. Rather than getting into the intricacies of each codec, we will briefly discuss their most important characteristics. An overview can be found in [4].

Lossy as well as lossless audio codecs exists. Popular lossy codecs greatly reduce file size without sacrificing too much quality. They achieve this thanks to advancements in the field of *psychoacoustics*, the study of the subjective human perception of sound: the encoder can omit certain parts of the recording, because they are simply inaudible or because the perceiver's mind will automatically assume that they are present. [5]

2.3.1 MPEG-1 Audio Layer 3 (MP3)

Developed in the late 1980s and standardized by ISO in the early 1990s, MP3 is probably the most widespread audio compression standard. From 1995 on, it gave way to the digital audio piracy controversy that is still going on today. In addition, it has become synonymous with today's portable audio player.

Because MP3 is a standard, but does not explicitly define the compression process, several implementations of the codec exist, each with their merit. The quality of a decoded MP3 recording therefore greatly depends on the software used for encoding.

MP3 codecs are *lossy* codecs that can reduce a raw recording to about one tenth of its original size, without substantial quality loss. So-called "near-CD" quality is achieved at the common bit rate of 128 kilobits per second. Other bit rates may be used, generally ranging from 32 kbps to 320 kbps.

2.3.2 MPEG-4 Advanced Audio Coding (AAC)

AAC became an MPEG standard in 1997 and greatly improves on MP3; it was designed to be its successor. Among the improvements are support for up to 48 channels, whereas MP3 only supports 2-channel stereo, and better overall performance.

While AAC may not be so well-known, it is certainly widely used: it is supported by many portable audio players, Apple's *iTunes Store* exclusively sells AAC recordings, Nintendo's *Wii* and Sony's *PlayStation 3* game consoles use the format, etc.

2.3.3 Windows Media Audio (WMA)

Microsoft's response to MP3 was WMA. Standard WMA uses techniques that are similar to MP3's and, according to Microsoft's claims, produces better results; this has often been contested. [6] The WMA suite also contains a lossless codec, and versions targeted at high-resolution multi-channel audio and voice audio.

WMA is backed by Microsoft's *Windows Media Player* and *Zune* franchise, as well as several third-party portable audio players and other audio equipment. This makes it the second most popular digital audio format, behind MP3.

Being a Microsoft product, WMA is not an open standard. Nonetheless, it is fairly well supported on other platforms than Windows, through third-party effort.

2.3.4 Vorbis

In 1998, Fraunhofer IIS, co-designer of the MP3 standard, imposed a licensing fee on any software that used an MP3 encoder and/or decoder. This spurred the development of a free, open source lossy audio codec. In 2002, version 1.0 of *Vorbis* was released.

Vorbis is often referred to as *Ogg Vorbis*, to emphasize that it is generally used in conjunction with the *Ogg* container, which we will discuss later in this chapter. Both are developed by the *Xiph.Org Foundation*.

While sound quality will always remain a subjective matter, a lot of experiments in this field seem to indicate that Vorbis outperforms MP3 and even AAC, especially at low bit rates; at higher bit rates, it is hard to make a distinction. [7] It should however be noted that the algorithms are ever-changing, as are the test results.

The Vorbis codec continues to gain popularity. For one, several mainstream computer games rely on Vorbis. Also, because of its open source nature, most, if not all audio players on the Linux platform support it well; for most other platforms, Xiph.Org maintains a number of plug-ins.

2.3.5 Dolby Digital (DD) and Digital Theater System (DTS)

Both DD, also known as *Adaptive Transform Coder 3 (AC-3)*, and its competitor DTS are lossy audio codecs targeted at high-end recordings for theatrical purposes, often making heavy use of up to 7.1-channel surround sound. DD and DTS tracks are found on *DVDs* as well as modern *Blu-ray Discs* and *HD-DVDs*.

2.3.6 Free Lossless Audio Codec (FLAC)

FLAC is everything its name suggests: free and lossless. Like Vorbis, it is distributed by Xiph.Org, and it is therefore seen as its lossless counterpart. FLAC can reduce your average recording's size by about 50%, without sacrificing any quality.

2.3.7 Miscellaneous Codecs

A number of other codecs exist, such as the lossy *Musepack (MPC)* and the lossless *Monkey's Audio (APE)*. We will not elaborate on these codecs, as they are far less common.

2.4 Video Codecs

Like we did for audio codecs, we will now list common video codecs and briefly discuss their characteristics. Note that to achieve acceptable quality, substantially more raw data is required per time frame than in the case of audio streams. Thus, common video codecs are all lossy. An overview can be found in [8].

2.4.1 MPEG-4

We have already encountered AAC, which is part of MPEG-4. That same standard also includes the means for video compression. It is divided into a number of *parts*, which are used by the actual video codecs. A lot of common video codecs are based on MPEG-4, including:

- **DivX.** This closed-source codec is to video what MP3 is to audio. By allowing pirates to compress films to acceptably sized files, DivX kick-started the digital video controversy. It is backed by *DivX, Inc.*, which generates revenue by charging license fees to its partners. The codec itself, however, is a free download.
- **Xvid.** Originally named *XviD* (which is DivX spelled backwards), Xvid is the open source alternative to DivX. The codec originated within the DivX project, after development of its *OpenDivX* codec ceased.

- **H.264.** Where older codecs use *MPEG-4 Part 2*, H.264 is based on *MPEG-4 Part 10*, dubbed *Advanced Video Coding (AVC)*. Thanks to modern compression techniques, the standard reaches noticeably higher quality at lower bit rates than its predecessors. The codec is widely used: *Blu-ray* and *HD-DVD* players are required to support it, Apple's *iTunes Store* exclusively uses it for its video content, recent versions of Adobe's *Flash Player* natively supports it, etc. The open source community has also adopted it using the *x264* codec. H.264's predecessors *H.261* and *H.263* are still found in legacy applications.

2.4.2 Windows Media Video (WMV)

To complement WMA, the *Windows Media* framework also contains a video codec. Microsoft claims WMV offers better compression than MPEG-4 codecs, a statement which is often debated. [9] Like WMA, it is a closed source codec, but reasonably well supported on non-Microsoft platforms, through community effort.

2.4.3 TrueMotion VP

The VP proprietary codec line by *On2 Technologies* may not sound familiar, but it is actually widely used: among other things, VP6 was adopted into Adobe's *Flash* platform. On2 Technologies claims TrueMotion VP7 is superior to H.264. [10] However, the latter is more common overall.

2.4.4 Theora

An early version of the VP codec, *VP3*, was released into the open source community back in 1992 and lives on today as Theora, the complement to the Xiph.Org Foundation's Vorbis and FLAC audio codecs. Obviously, Theora is a free and open source codec. Although it is a direct competitor to MPEG-4, Theora's VP3 legacy negatively affects its performance. Despite this, Theora is widely supported in the open source community and, thanks to software add-ons, can be played back on most platforms.

2.5 Container Formats

The *container format* indicates which streams are stored in a container.

A first approach to designing a container format would probably be to store the streams sequentially inside the file. However, a more sensible approach is to keep related parts of the streams in close proximity within the container. This is achieved by the process of *interleaving*, i.e. alternating between small segments of the contained streams.

The following factors may or may not set container formats apart:

- **Content.** Some container formats support more content types than others. For instance, not every format provides the functionality to embed subtitles.
- **Supported codecs.** Recent codecs, notably H.264, rely on advanced features of the container to increase output quality. Sadly, not every container format actually supports these features.

- **Compatibility.** Like in the case of codecs, an arbitrary system is unlikely to support any container format.
- **Commercial model.** In this field as well, parallels may be drawn with codecs.

We will now look at some commonly used container formats. Yet again, we will not go into great detail, but simply form an idea of what is in circulation. An overview can be found in [11].

2.5.1 Audio Video Interleave (AVI)

AVI is the de facto multimedia container of the Windows platform. Dating back to 1992, it unfortunately has its flaws. Most importantly, the H.264 codec's advanced compression technology uses a number of features not found in the AVI format. In addition, embedded subtitles are not supported. Nonetheless, thanks to excellent support by modern media players, the AVI format remains popular.

2.5.2 MPEG-4 Part 14 (MP4)

The MPEG-4 standard collection also defines a container format. MPEG-4-contained files generally use the extension `.mp4`, hence the format's common name. The format is obviously aimed at use with other MPEG standards, but supports a wide range of codecs. Subtitles are supported using the *MPEG-4 Part 17* format, dubbed *MPEG-4 Timed Text*. Among others, Apple's *iTunes Store* offers MP4 content.

2.5.3 QuickTime (MOV)

Apple's QuickTime container format, which uses the extension `.mov`, was accepted as the basis of *MPEG-4 Part 14*. MP4 and MOV can use the same codecs and are mostly interchangeable. However, MP4 is far more widely supported, as QuickTime media files are heavily aimed at Apple's QuickTime product line.

2.5.4 Advanced Systems Format (ASF)

Windows Media content is generally encapsulated in an ASF container, though other container formats support the codecs as well. The format is aimed at streaming media—the letter S in ASF originally stood for *streaming*—but is not limited to this application. ASF files may use the extension `.asf`, but also `.wmv` for video and `.wma` for audio.

2.5.5 Flash Video

Adobe's *Flash* multimedia platform uses its own container format. It is proprietary, but understood by a lot of modern media players. Codec support is severely limited: standard audio should be MP3 and video either VP6 or H.264, as mentioned earlier; there is also legacy support for H.263. The original extension for Flash Video files was `.flv`, but restrictions in the format prompted a pending revision, which will be targeted at MP4 and introduce new file extensions.

2.5.6 Ogg

Ogg is the Xiph.Org Foundation's complement to the Theora, Vorbis and FLAC codecs. Ogg files can also contain streams in a wide range of codecs, including MPEG-4. Thanks to the success of the Vorbis audio codec, the Ogg format is widely supported these days.

2.5.7 Matroska

Matroska is another free and open source container format, which is set to replace Ogg. It addresses some of the concerns raised by the Ogg format, including support for even more codecs[1] and overall flexibility of the container. Matroska files use the extension `.mkv` for video and `.mka` for audio.

2.6 Unification

The codecs and container formats we have listed are just the tip of the iceberg. Consequently, the video clips that users will want to share using our platform may occur in vastly different forms. Rather than requiring that the computer on which clips are played back support all these formats, we will decide on a single audio codec, video codec and container format. But can we just convert any supplied clip to this format?

We have already discussed the processes of *encoding* and *decoding*. A type of third process exists, known as *transcoding*, which involves conversion from one codec to another.

Generally speaking, transcoding consists of decoding the source material to a raw uncompressed version, which is then encoded into a different format. These two tasks may or may not be carried out by the same software. There are several approaches; efficient implementations, for instance, minimize disk operations by only using internal memory to store the raw version.

This all sounds wonderful, but there is a catch. Converting between lossy formats has a negative impact on the quality of the media. Each time lossy encoding is carried out, media quality degrades: the compression process leaves behind traces, called *artifacts*.

Nonetheless, transcoding the supplied clips is definitely our best option. It allows us to target our solution at a single configuration and guarantees that users will be able to play back the clip as long as they agree to use the supporting software we provide. Minor loss of quality is a necessary evil.

1 Ogg's limited codec support even prompted the development of a hack of the format, named *OGM*.

3 STREAMING MEDIA

The previous chapter discussed the structure of a video file. As mentioned earlier, the objective is to make the video material available on a network. In this chapter, we will see how this can be accomplished.

3.1 Motivation

To share our video material on an intra- or internetwork, we could take the following simple approach: store the files on a Web server, provide users with a location where they may download the media files and let them handle their local copy as they wish. This method raises a number of concerns:

- The user cannot access the media until the file has been fully downloaded. With large files, this can be a major setback.
- Until now, we have assumed the existence of a media *file*. How do we handle live broadcasts?
- Once the download has been completed, the user possesses an exact copy of the media file. For copyright reasons, this might not be desired.

Streaming addresses these concerns. Streaming media transports subsequent parts of the streams over the network. Thanks to the *interleaving* process we discussed earlier, the user can enjoy the media as upcoming parts trickle in; these are stored in a buffer. Furthermore, the bit rate may be controlled dynamically: should the connection slow down drastically, then the player can fall back to a lower quality stream and vice versa.

Consequently, large files do not pose a problem anymore. A live broadcast can be treated as an infinitely large file. As for our copyright concerns, it is a lot harder to permanently store streamed data; note, however, that it will never be absolutely impossible to do so.

3.2 Protocols

Streaming applications may be based on either of two well-known connectivity models: the *client-server* model and the *peer-to-peer* model. In our case, the former will be applied.

Our streaming infrastructure will essentially consist of a server that offers media streams, and a number of clients that request them. A number of protocols enable the parties to communicate. We will describe the two most commonly deployed ones.

Before diving into those protocols, we will identify the basic operations that may be performed on a stream. We will then compare the protocols by examining how they handle them. The operations are:

- **Play.** The stream is started from the beginning.
- **Pause.** The stream is interrupted, but the user has the intent to to resume playback in the near future. Buffered data is kept.
- **Stop.** The stream is interrupted and the user suggests that he is no longer interested in the content. Buffered data is discarded.
- **Rewind and fast forward.** The user freely navigates through the stream and decides to resume playback from an arbitrary point. Buffering may precede playback. Standard DVD players manage to provide the viewer with a still preview of the content at any point while navigating. Due to network latency, this is generally impossible to achieve with streaming media—except if the content happens to be in the buffer.
- **Navigation to a chapter.** Playback is resumed from a predefined point in the stream. Again, buffering may precede playback.

3.2.1 HTTP

The ubiquitous *Hypertext Transfer Protocol (HTTP)* is suitable for streaming media applications, though it does have limitations. The HTTP model is limited to one request, followed by one monolithic response. Thus, there will usually be a chain of requests.

The standard operations are generally implemented as follows.

- **Play.** A standard `GET` request and response are transferred. While playback is happening, the rest of the stream is downloaded without further control. *Throttling* (artificially slowing down the connection) may be beneficial, as well as forcibly closing down the connection in order to make a new request.
- **Pause.** The HTTP protocol does not have an equivalent of this operation. Again, throttling and/or subsequent new connections may be used.
- **Stop.** The connection is closed and the rest of the response is discarded.
- **Rewind and fast forward.** The connection is closed and a new request is made. For resuming the download of a stream, HTTP 1.1 defines two headers: `Range` and `Content-Range`. One important drawback is the fact that these only support *byte* amounts. Thus, some arithmetic is required to map timestamps in a stream to byte amounts in a file. Even if the stream has a constant bit rate, this is not to be taken lightly.
- **Navigation to a chapter.** Here too, the remark about time vs. memory units stands. However, as chapter marks are known in advance, the translation may be carried out prior to playback as well.

Obviously, efficient implementations strive to limit the number of requests. The larger the buffer size, the smaller the chance of having to perform a new request; however, unless the user does not navigate through the piece, performing multiple requests virtually cannot be avoided.

Even if one manages to circumvent the problem regarding multiple connections, there is still the matter of HTTP being restricted to *TCP*. Packet loss, for instance, is not a tremendous issue: the resulting media can simply be distorted for a moment; however, TCP does not allow this. For this and other reasons, *UDP* is a better choice for streaming applications.

Despite the protocol's limited possibilities, HTTP-based streaming solutions are frequently deployed. After all, it is simple and ubiquitous. In addition, firewalls generally do not pose a problem.

3.2.2 RTP, RTCP and RTSP

A more suitable protocol for streaming applications is the *Real-time Transport Protocol (RTP)*. It was developed specifically for audio and video streaming over the Internet; it is also used by *VoIP*[2] solutions. RTP uses *UDP* and as such does not guarantee message delivery; however, each message receives a sequence number, so the software can order them. [13] If combined with the *Real-time Transport Control Protocol (RTCP)*, there is support for *QoS monitoring*. [14]

Control of the media stream uses a separate protocol, namely the *Real Time Streaming Protocol (RTSP)*. It can either run on top of UDP (including *multicast*) or TCP. In addition, RTSP may be used independently from RTP. RTSP's syntax is a lot like HTTP's, yet the protocol is *stateful*, thanks to the inclusion of a session ID into each message. [15] In a typical RTSP session, the communication will take place as follows: [16]

- The client receives an `rtsp://` URL from an external source.
- The client asks the RTSP server to enumerate the streams associated with the URL it received. This is achieved using a `DESCRIBE` request.
- The server responds by sending information about the streams to the client. Usually, it is formatted using the *Session Description Protocol (SDP)*.
- The client selects one or more streams and sends a `SETUP` request for each of them. Among other things, the request contains the port number to use for the RTP connection.
- The server responds with a newly assigned session ID.

Once the session has been established, the operations we mentioned earlier can be carried out:

- **Play.** The client sends a `PLAY` request and the server initiates the RTP stream.
- **Pause.** The client sends a `PAUSE` request and the server pauses the stream.
- **Stop.** The client sends a `TEARDOWN` request and the stream is discarded.

2 *VoIP* is short for *Voice over Internet Protocol*. It is the main protocol used for Internet telephony. More information about the protocol can be found in [12], among others.

- **Rewind and fast forward.** The client sends a `PAUSE` request, followed by a `PLAY` request with a `Range` header. This is similar to HTTP, but uses seconds instead of bytes to indicate the point from which to resume playback, eliminating the translation problem.
- **Navigation to a chapter.** Again, this operation is largely similar to the previous one.

RTSP has a myriad of other possibilities; for instance, it is possible for the client to record a stream and send it to the server. We will not elaborate on these features. The key points are that RTP uses UDP and offers specific streaming media-related functionality.

Be that as it may, RTSP and RTP have their drawbacks. Apart from the obvious need for software support, port numbering may also cause problems. RTSP uses the fixed port number 554, but the port number for the RTP session is chosen in the RTSP dialog; furthermore, the RTCP port number is always one greater than this. When using firewalls or NAT, this needs to be taken into account.

4 STREAMING SOFTWARE

In this chapter, we will select tools that our streaming video platform will rely on, on both the server and the client side. This chapter focuses solely on processing the media files; in the next chapter, we will discuss the application encompassing this task.

4.1 Server Software

In this section, we will encounter a number of applications. Interestingly, all of them are available for any modern operating system. However, it should not come as a surprise that a free and open source solution will be preferred, such as *Linux* or *BSD*.

4.1.1 Transcoding

Probably the most important thing we need is the means to transcode user-submitted video files. Since we are looking to automate this task, a command line utility is obviously what we are after.

Note that we have yet to decide on codecs and container formats. To make this choice even more complicated, every application capable of transcoding has support for a different subset of the formats that are currently available. In addition, we will see that the streaming server and playback software also vary with the used format. Therefore, we will postpone our choice until after we have looked at all three aspects.

A couple of open source transcoding utilities exist. *MEncoder* (a sibling of *MPlayer* [17]) and *FFmpeg* [18] may sound familiar to Linux users. Both are capable of reading and writing most common codecs and container formats, as well as modifying the streams; specifically, their ability to change the video's resolution is interesting to us. The aptly named *Transcode* suite [19] offers similar functionality.

However, there is one drawback to all three of these: at the time of this writing, none of them have support for creating media files in Xiph.Org's formats. Luckily, there is the standalone project *ffmpeg2theora* [20], which borrows FFmpeg's feature set and adds the ability to create Ogg files, containing a Theora video stream and a Vorbis video stream.

4.1.2 Streaming

As mentioned in the previous chapter, media streaming may either use plain old HTTP or the specialized RTP protocol set. Let us take a look at software for streaming using either protocol.

For HTTP-based streaming, we can obviously turn to generic Web servers, such as *Apache* [21] or maybe the lightweight *lighttpd* [22], which has been gaining popularity recently.

Looking at servers which are specifically aimed at streaming, most are focused on on-the-fly transcoding and live broadcasting. However, one is not obligated to use these features.

A popular option for HTTP-based streaming is Xiph.Org's *Icecast* [23]. This project's server application originated as an audio streaming server, supporting Vorbis, MP3 and AAC. Recently, streaming Theora video was added to its capabilities.

Another alternative is the *Flumotion Streaming Server* [25] by *Fluendo*. This is an open source modular streaming solution, with an easy-to-use graphical administration interface. It is based on *GStreamer*[3] and supports a wide range of formats.

The *VideoLAN* project's all-round *VLC media player* [26] is capable of serving both HTTP and RTP streams. Unfortunately, it requires a separate instance for every stream, which would be highly ineffective in our case.

Our platform is set to replace the configuration currently in place at Vrije Universiteit Brussel, which uses Apple's *Darwin Streaming Server* [27]. DSS was the first open source RTP-based streaming server and forms the base of the more advanced, closed source *QuickTime Streaming Server* [28]. Most noteworthy is that the server is capable of streaming H.264 video.

4.2 Client Software

In the case of the client, we are not at liberty to choose a platform. We need to make sure our configuration is compatible with as many systems as possible. Our goal will be to provide a solution for systems running Windows, Mac OS and common Linux distributions.

As mentioned earlier, the choice of a container format will greatly depend on the effort required to play back the transcoded clip. Two situations arise. Firstly, the user should be able to watch the clip from within his Web browser. Secondly, if the user prefers to do so, he can download the video file and play it back using his favorite media player. Note that, in terms of formats, this downloaded version is not necessarily the version that is used for in-browser playback.

4.2.1 In-Browser Playback

Probably the easiest method to play back a clip inside the browser window is to provide a standard link to the HTTP URL of the clip, assuming that the user has a plug-in installed. For instance, this is the case with the

3 *GStreamer* [24] is a high-performance multimedia framework, used primarily by the *GNOME* desktop environment.

QuickTime plug-in: opening a URL to a supported media file fills the view port with a minimalist *QuickTime Player* interface. Of course, there are more elegant and trustworthy methods.

Using HTML's `<object>` element, pages may embed a variety of applications, all supported by plug-in software. Two types of plug-ins may be used. The first type is an embedded version of a standard media player, again such as *QuickTime*. The second type is more elaborate: it supplies an intermediate platform, such as *Sun Java* or *Adobe Flash*, which in its turn runs a custom-built media player as an *applet*.

Despite the fact that it is used by most video sharing Web sites and that free Flash video players are available[4], we will rule out Flash. After all, there are a couple of downsides: it is not open source, support on Linux is still subpar and it requires the use of a proprietary container format.

So we are left with Java applets. Java's platform independence should make our solution fully cross-platform, which is a nice start. Sun's *Java Media Framework API* [30] is supposed to implement cross-platform multimedia support, but in reality seems largely abandoned.

Luckily, we have an alternative: while Fluendo's *server* software may not be our best option, the company also offers the *Cortado* applet [31], capable of playing Ogg Theora over HTTP. Cortado is used by the *Wikimedia Foundation* [32] on the ever so popular *Wikipedia* [1], among others.

Finally, the next version of HTML, HTML 5, is very likely to feature native support for audio and video clips, via the new `<audio>` and `<video>` elements. [33] At one point, Ogg formats were favored by the preliminary specification, but Apple and Nokia opposed this move. [34] Nonetheless, it is something to take into consideration for future versions of our platform.

4.2.2 Post-Download Playback

If the user chooses to fully download the video clip prior to watching it, a standalone media player capable of playing back the file needs to be available on his computer. To some extent, this is the user's own responsibility. However, if our format cannot be played out-of-the-box, we can aid users somewhat by directing them to media players that are known to support it.

As mentioned, *VLC media player* is somewhat of a Swiss army knife. It is available for many operating systems and is capable of playing back a myriad of formats. As such, no matter what operating system the user is running, we can recommend installing VLC media player.

For a more operating system-native solution, alternatives generally exist:

- On **Windows** computers, the standard media player is obviously *Windows Media Player*. It uses Microsoft's *DirectX* multimedia framework [35], which gains supports for extra formats if one installs a supporting *DirectShow filter*. As an added benefit, any third-party media player built on DirectShow can also use the filter. A common all-round set of filters, which is also open source, is *ffdshow tryouts* [36], a fork of the now unmaintained *ffdshow*.

4 The *JW FLV Media Player* [29] is a popular free Flash video player.

- A similar approach is required on **Mac OS X**. The *QuickTime* framework is Apple's answer to DirectX; where DirectX uses filters, QuickTime relies on *components*. Freely available components exist for most formats that are not supported by default.
- Modern **Linux** distributions are capable of playing back open formats. Additionally, documentation on supporting proprietary formats is generally provided. Most of the software uses the same libraries as our server-side configuration. Similarly, other UNIX derivatives can use them.

4.3 Decision Time

We have now covered all three factors affecting the choice of a container format and codecs: transcoding, streaming and playback. Consequently, it is time to reach a verdict.

Even though the *Xiph.Org* formats may not be the latest and greatest, they offer a number of benefits:

- The Ogg container as well as the Theora and Vorbis codecs are *free and open source*.
- The *ffmpeg2theora* utility can transcode virtually any input to Ogg with Theora and Vorbis.
- The *Cortado* applet can play back our clips on any platform.
- For playing back downloaded clips, the Xiph.Org Foundation's web site has links to *DirectShow filters* and *QuickTime components*. Linux platforms embrace the formats by default.

Of course, there are also some drawbacks …

- Not all users will have the Java Runtime Environment (and plug-in) installed. Its installation is somewhat tedious compared to, for instance, Flash. Nonetheless, Java applet technology is widely used on the Web, making it a reasonable requirement.
- The applet does not support RTP streams, so we will use an Apache HTTP server. As mentioned, we will not be doing live broadcasting or real-time transcoding, so this should not be an issue.
- Users might frown upon the fact that they need additional software to play back our clips. However, *Google Video*'s download option, for instance, is limited to a version targeted at portable video players. [37] Also, we could make an extra transcoding pass targeted solely at downloads; this is however not a priority.

Thus, we can set up a *fully open source* streaming video platform.

One thing we have not taken into account is *DRM* or *digital rights management*. The copyright issue can get rather complicated, but we manage to sidestep it to some extent: once transcoded, we want our videos to be public, end of story. However, we can expect the odd DRM-protected piece of source material, for which *ffmpeg2theora* will be unable to retrieve a license. The best thing we can do to avoid this is educate our users.

Finally, now seems like a good time to decide on a name for the software; after all, pretty soon, we are going to be confronted with the problem of picking file names and the like. Contracting the phrase '**o**pen **vi**deo **s**haring' (and/or '**s**treaming') a bit, we end up with *Ovis*. Not only is that name pretty catchy, it is also the Latin word for 'sheep'. We all love sheep, don't we?

5 REQUIREMENTS ANALYSIS

By now, we have decided on how to convert user-supplied video material and how to present the resulting clips. We have also mentioned that the front-end for both these aspects will be a Web application. In this chapter, we will address some initial considerations regarding its development.

5.1 Features

Let us start by writing up the operations that the Ovis demo will support.

- **User management.** To link content to users, we will set up a trivial user base. We will call logged in users 'members' and others 'guests'.
- **Clip upload.** Members will be able to post video files on the platform.
- **Clip management.** Members will have access to a simple overview of their clips. Because the demo should focus on issues related directly to streaming video, we will not be implementing generic features such as the modification and deletion of video clips.
- **Clip playback.** Both visitors and members will be able to browse and watch the clips that members have uploaded to the platform.

5.2 Software Platform

As our goal is to eventually integrate the libraries that support the Web application with *Dokeos*, we will adhere to Dokeos's characteristics as closely as possible. Therefore, we will develop a *PHP 5* application with a *MySQL* back-end. Our Web server will run *Apache*.

5.3 File Size

HTML forms are highly unsuitable for uploading large files, and unfortunately, video files nearly always fit into that category. Even if the Web server is configured to allow large `POST` requests and long timeouts, a

standard form is far from ideal. After all, it requires that the Web browser keep running the entire time. Furthermore, resuming uploads is not possible.

We will take a different approach. Rather than using an HTML form for submitting new clips, users will be requested to launch a *Java Web Start* application from the browser. This technology allows a Java application to be started directly from a URL and keeps the application running, even if the Web browser is closed. [38] Again, Java's platform independence is a great benefit. Users are likely to have the Java Runtime Environment installed anyway, as we will supply the Cortado applet.

5.4 Upload Protocol

Now, since we cannot use HTTP for uploading files, we will need to select a different protocol. Rather than developing our own protocol, we will use *FTP*. Because FTP is a very common protocol, we can easily use existing software on both ends of the connection. We will select our server software in the next section. The client software will be discussed in Section 12.1.

While using FTP makes large uploads more reliable, it does have a downside. Because we are no longer using HTML forms, we cannot send a clip's metadata, such as its title, along with the request. We will circumvent this by developing a *Web service*, which our upload application will send the metadata to when an upload completes. Evidently, the user will have to supply the metadata in the application itself.

5.5 FTP Accounts

As FTP authentication information is sent across the network in plain text, it would be unwise to reuse existing user accounts. Furthermore, in Section 13.7, we will see that Java Web Start forces us to expose the information rather blatantly as well.

Consequently, right before the Java Web Start application is launched, a temporary FTP account will be set up for the current user. The account will automatically expire after a predefined period—say, a week. The credentials associated with the account will be passed directly to the Java Web Start application, thus relieving the user of any additional effort.

This way, if worst comes to worst, a user who manages to obtain the FTP authentication information will only be able to access the upload directory of the member in question.

5.6 FTP Server

Having decided on the FTP protocol, we will need to select FTP server software. Our FTP daemon of choice is the open source *ProFTPD* [39]. Apart from being a popular choice with system administrators, ProFTPD can use a *MySQL* back-end for user management. This means that we will be able to share the same database between the front-end and the FTP server.

5.7 Undesired Segments

Not all users are capable of authoring[5] a video clip. Therefore, we can expect users to upload clips containing material that is of no interest to them, resulting in vast amounts of unnecessary upload data.

As we will be supplying a Java application, we could have it interpret video files. We could even turn it into a full-blown authoring solution. Unfortunately, as mentioned earlier, the Java Media Framework leaves a lot to be desired. In addition, developing video authoring software is a rather large task, presumably unsuitable for this project. It would no doubt be interesting to display a preview of the clip inside the upload application, but this would probably involve using a different platform, thus abandoning Java's platform independence.

Looking at alternatives, we could direct our users to video authoring solutions. However, we want to avoid confronting them with the issues of containers and codecs, bit rates, etc.

As a compromise, the upload application will give users the opportunity to specify the start and end times of segments that are relevant to them—presuming they are able to obtain these timestamps from their preferred media player software. This way, we will still have to deal with partially superfluous uploads, but at least we will be able to limit the transcoding process, as well as eventual storage and playback, to relevant segments.

5.8 Chapter Marks

Even with undesired material removed, clips supplied by our users may be long. In this case, they may want to divide the clip into several chapters. The Cortado applet has a simple *JavaScript API*, which allows us to easily seek in the clip, on the client side.

Unfortunately, due to this project's limited time frame, support for chapter marks will only be included in the Dokeos version. We will see that its infrastructure greatly simplifies this task.

5.9 Transcoding Time

Transcoding can take a substantial amount of time. Consequently, when a member finishes uploading a clip, it will be added to the *transcoding queue*. This queue will be processed in the background, transcoding subsequent clips. Once a clip has been transcoded, it will become publicly available. When transcoding is complete, we can, for instance, notify the clip's author via e-mail.

5.10 Transcoding Profiles

Because users' bandwidth can greatly vary, we will include the possibility to configure application-wide *transcoding profiles*, so as to be able to offer several versions of a clip, targeted at different baud rates. Thus, when clips are transcoded, a version will be created for each transcoding profile. When they are played back, we will provide the means to switch between profiles.

5 'Authoring' is the term generally used to describe the process of preparing video material for publication.

Each profile will define the maximum resolution of a video clip, as well as quality settings for the audio and video streams. For video streams, `ffmpeg2theora` accepts a numeric quality between 0 and 10; in the case of audio, it is between –2 and 10. In addition, we will impose a maximum on the number of audio channels, e.g. to down-mix surround sound to stereo or mono.

5.11 Maintaining Aspect Ratios

By defining a maximum video resolution in transcoding profiles, we make the problem a little bit more complex. Consider the following diagrams:

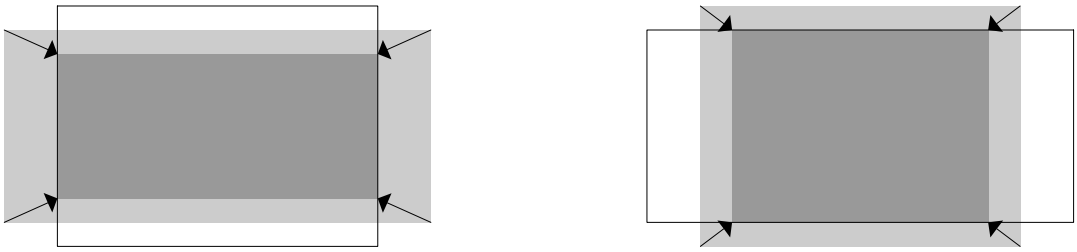

Figure 5.1: Maximum video resolution exceeded in one dimension

The outer, light gray area represents the video frame we are trying to shrink, while the outline corresponds to the maximum dimensions specified by our transcoding profile.

In the left picture, the video clip is too wide for the frame. Hence, we need to lower its width, and adjust its height accordingly. Thus, we obtain the inner, dark gray area. In the right picture, it is the other way around: the video clip is too high, so we will lower its height and its width will follow.

It is also possible that both the clip's width and its height exceed the frame's dimensions. How the clip is shrunk down will depend on how its dimensions relate to the frame's:

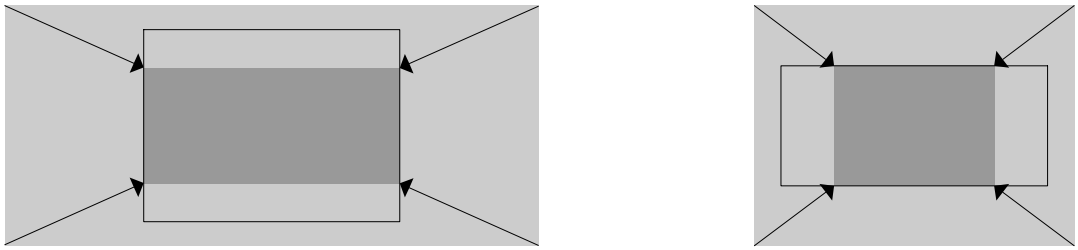

Figure 5.2: Maximum video resolution exceeded in both dimensions

In all four cases, the resizing method can be deduced from what is known as the *aspect ratio* of both the clip and the frame: width divided by height. If the clip's aspect ratio exceeds the frame's, then the clip is relatively wider than the frame. Thus, we need to decrease its width and consequently adjust its height accordingly; otherwise, it is the other way around.

Note that a fifth, rather trivial case exists: if the video clip can be fully contained inside the reference frame, then no resizing is necessary.

5.12 Cortado Limitations

As we will be using the Cortado applet to play back our clips, we will need to generate some HTML code for it. This code must contain, among other things, the applet's dimensions. Unfortunately, we cannot merely copy the maximum dimensions specified in the transcoding profile. Tests revealed that the Cortado applet will stretch the image to fill the video frame, even if its `keepAspect` parameter is set; this behavior is inconsistent with the documentation [40].

To circumvent this, we will store each clip's aspect ratio in the database. We can then determine the applet's dimensions on the fly, by repeating the logic described above. This way, we do not need an entry for each {*clip*, *profile*} tuple, and we only need to hold on to one extra value.

5.13 Thumbnail Images

Most streaming video portals add some visual identity to each clip by creating a thumbnail image, i.e. a fixed-size snapshot of one of the clip's frames. We will follow these Web sites' lead and work out a way to create such an image.

6 BASE CONFIGURATION

This chapter will formally identify the set of servers that our system will run on, as well as the supporting software on each of the machines. Over the next couple of chapters, we will then discuss the installation and further configuration of that software.

6.1 Server List

To increase flexibility, we will assign each individual task to a server entity. Naturally, a single machine may be responsible for multiple tasks, e.g. via loopback connections or by directly accessing the appropriate files. The eventual configuration is illustrated in Figure 6.1.

- **Application server.** Web server that runs our Web application, including the Web service.
- **Database server.** Contains the database for the Web application.
- **Upload server.** FTP server that accepts video file uploads from known users. Consults the database server for authentication information.
- **Source file server.** Stores incoming uploads. Thus, the upload server writes to it.
- **Transcoding server.** Performs all transcoding operations. Consults the transcoding queue on the database server and retrieves files to transcode from the source file server.
- **Destination file server.** Stores transcoded video clips. Thus, the transcoding server writes to it.
- **Streaming server.** Web server that loads video files from the destination file server and streams them to connected viewers.

This configuration does not account for additional load balancing by spreading each responsibility across multiple machines. This is left as an exercise for the astute reader.

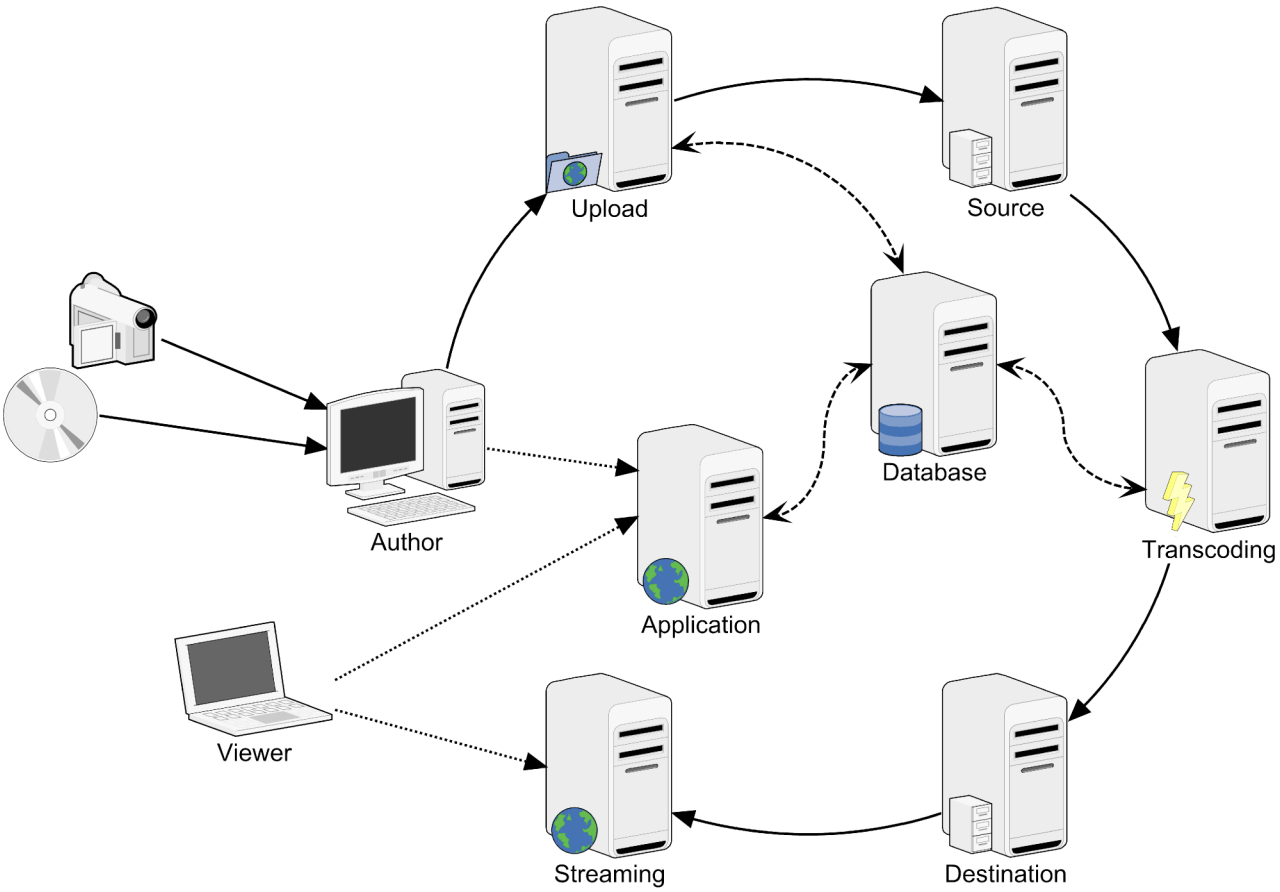

Figure 6.1: Overview of platform servers

6.2 Package Management

From this chapter onwards, we will be installing and configuring software. Because many, if not most Dokeos installations run *Debian GNU/Linux* [41], we will assume to be starting from a minimal Debian installation with networking capabilities. More specifically, we will use the *testing* branch, which is codenamed *lenny* at the time of this writing.

Consequently, we will list relevant packages for Debian's *Advanced Packaging Tool (APT)* and explain how to adapt the default configuration for our needs. The process may vary for users of other operating systems.

6.3 Server Interoperability

Obviously, our servers will be interacting. There will be two types of connections between servers: file system shares and database connections.

In Figure 6.1, solid lines are used to represent the life cycle of an uploaded video file; to transfer files between the servers involved, we will set up file system shares. Dashed lines, on the other hand, represent database connections. Finally, dotted lines indicate client-server interaction via HTTP; the configuration of those servers will be covered in Chapter 9.

6.3.1 File System Sharing

A straightforward solution for sharing file system paths over a local area network is the *Network File System (NFS)*. For those unfamiliar with configuring NFS, we will now take a look at a reference scenario. The process described here is based on [42].

In this scenario, the transcoding server will gain access to the destination file server. While we will configure read/write access, NFS can also be configured for read-only access.

We will call the transcoding server *transcoding.example*, *example* being the local domain. The destination file server will be *storage.example*. Transcoded files will be in the directory `/etc/ovis/out` on *storage.example*.

Like many protocols, NFS relies on a client-server model. The machine where the shared path resides is the server. Thus, in our case, the server is *storage.example*, making *transcoding.example* the client.

6.3.1.1 Server Configuration

On the NFS server machine, *storage.example*, we need to install the NFS daemon. On a Debian GNU/Linux machine, we can do so by typing

```
# apt-get install nfs-kernel-server
```

Now, we can define our share. We open up `/etc/exports` in a text editor and add the line

```
/etc/ovis/out transcoding.example(rw,sync,no_subtree_check)
```

This exposes `/etc/ovis/out` to the host *transcoding.example* with read/write access. The options `sync` and `no_subtree_check` need to be specified to avoid ambiguity warnings. Additional client addresses may be added to the entry, each preceded by whitespace. It goes without saying that great care should be taken when allowing any sort of remote access.

To make the NFS daemon aware of our changes, we issue the command

```
# exportfs -a
```

6.3.1.2 Client Configuration

Now, *transcoding.example* needs to be configured as an NFS client. Machines running Debian GNU/Linux should first install the required software by issuing the command

```
# apt-get install nfs-common
```

Next, we choose and create a mount point.

```
# mkdir /mnt/ovis/out
```

We are now able to `mount` the share. However, we probably want it to be permanent, so we will add an entry to `/etc/fstab` instead:

```
storage.example:/etc/ovis/out /mnt/ovis/out nfs rw 0 0
```

And voilà, the share can now be mounted.

```
# mount /mnt/ovis/out
```

6.3.1.3 Share Identification

The file system shares that support our setup are listed in Table 6.1. Depending on the configuration, one or more shares may reside on the same machine, making them obsolete.

NFS client (mounting)		NFS server (mounted)		Writable
Upload server	`/mnt/ovis/in`	Source file server	`/var/ovis/in`	Yes
Transcoding server	`/mnt/ovis/in`	Source file server	`/var/ovis/in`	Yes
Transcoding server	`/mnt/ovis/out`	Destination file server	`/var/ovis/out`	Yes
Streaming server	`/var/www/media`	Destination file server	`/var/ovis/out`	No

Table 6.1: Shared file system paths and associated mount points

6.3.2 Database Interaction

Several servers will require access to the same MySQL database. While most platforms and programming languages support MySQL, the syntax for connection parameters greatly varies between them. One library might require a data source name in a proprietary format, others might ask that the hostname and credentials be specified separately. Therefore, we will define a trivial XML-based file format for specifying data sources:

```
<datasource>
    <driver>mysql</driver>
    <host>mysql.example.com</host>
    <port>3306</port>
    <username>sheep</username>
    <password>supersecret</password>
    <database>ovis</database>
</datasource>
```

For the sake of flexibility, we will also add an optional element called `<prefix>`, which can be used to specify a global table name prefix. This way, the database can be shared with other applications if desired.

Since parsing XML is a common process these days, each part of the Ovis platform can easily obtain all the required information from the markup. This will allow us to reuse the exact same data source definition file on all servers that require access to the database.

7 DATABASE SERVER

In this chapter, we will see how to set up MySQL server software on a Debian system, as well as design the database for our application.

7.1 Installation

Installing MySQL Server on a Debian GNU/Linux machine is child's play:

```
# apt-get install mysql-server
```

This will both install and start the server. We can then fire up the MySQL console.

```
$ mysql -u root -p
```

Then, we can create our database and choose the credentials for establishing a connection. This will be the information we store in the data source definition file described in Section 6.3.2. We will allow that user to alter the database structure; this will come in handy when we supply an installation script later on.

```
CREATE DATABASE ovis;
GRANT CREATE, ALTER, DROP, INDEX, SELECT, INSERT, UPDATE, DELETE
    ON ovis.*
    TO 'sheep'@'192.168.1.%'
    IDENTIFIED BY 'supersecret';
```

In this example, we created the account so that the user can connect from any machine in our *192.168.1.0/24* local area network; alternatively, we could specify a host mask like `'%.example.com'`. If we want the user to be able to connect from the machine itself, e.g. because the database server and the Web server are on the same machine, we can use `'localhost'`. [43]

To simplify management of our database, we can install *phpMyAdmin* [44]. As this is a Web-based utility and we have yet to set up a Web server, we will get back to that in Chapter 9.

7.2 Database Model

The model of the Ovis demo's database is displayed in the entity-relationship diagram in Figure 7.1.

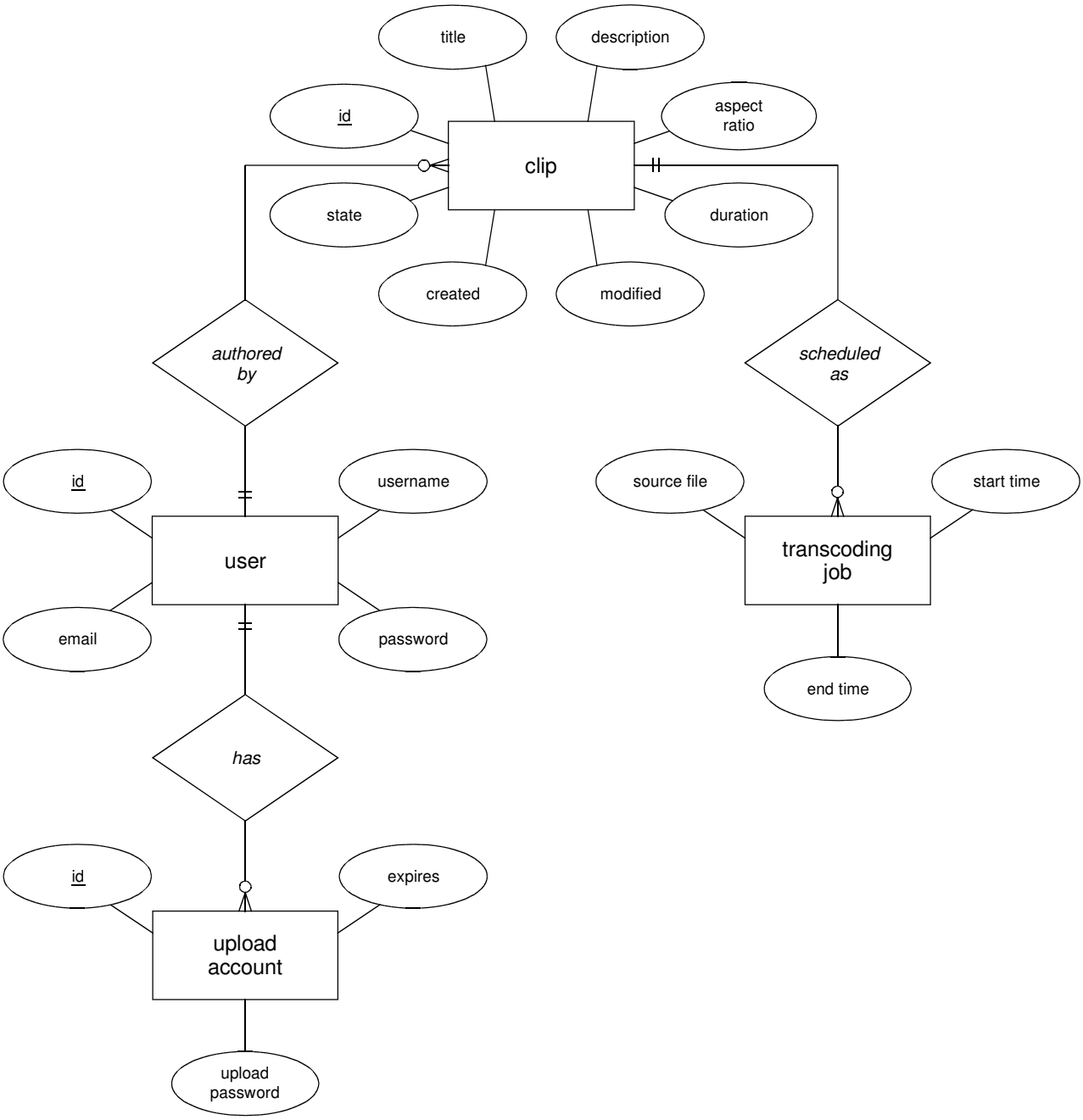

Figure 7.1: Entity-relationship diagram for core entities

For the most part, the diagram will probably be self-explanatory. Nonetheless, let us take a closer look at the individual entities in the model.

7.2.1 Users

The user table contains accounts for the demonstration platform; other applications will probably rely on a similar structure. As is customary, users' passwords are hashed before storage. The e-mail address of each user is kept as well; it will be used to contact the author of a clip after transcoding.

7.2.2 Upload Accounts

Each user account is associated with zero or more upload accounts. As discussed in Section 5.5, an expiry date accompanies each of these. Passwords are be automatically generated and stored without encryption.

7.2.3 Clips

Each clip refers to its author, i.e. the user who supplied it. The title and description, as entered by that user, are kept as well. Two timestamps indicate when the user uploaded the clip and when it was last modified.

Then we have some information about the file itself, namely the clip's duration in seconds and the video aspect ratio. These will not be filled in until transcoding takes place, as they require examining the file.

Finally, we will also keep track of a clip's state. It can be:

- queued for transcoding,
- in the process of being transcoded,
- fully transcoded and hence available for viewing, or
- affected by a transcoding error.

Note that there is no column for the clip's filename. We will simply form the associated filename by concatenating the clip's unique ID with the name of the transcoding profile used. In the case of a thumbnail image, it will be even simpler, as we can leave out the profile.

7.2.4 Transcoding Jobs

A transcoding queue entry refers to a clip that needs to be transcoded. Such queue entries will be removed when transcoding completes; thus, there will eventually be far more public clips than queue entries. Therefore, this information is stored separately.

Two optional fields indicate the start and end time, to allow users to select one or more segments for transcoding, instead of the entire clip.

This time, we do need to supply a source filename as well. If we were to assign a unique ID to each entry and deduce the source filename from it, we would not be able to use the same file as a source for multiple clips. We could use symbolic links, for instance, but let us keep it simple.

Strictly speaking, we would have to store source filenames in a separate table and refer to them from the transcoding queue. Again for the sake of simplicity, we refrain from doing so.

7.2.5 Transcoding Profiles

For convenience, we also store our transcoding profiles in the database. After all, several servers need to access the list. The table is modeled as follows:

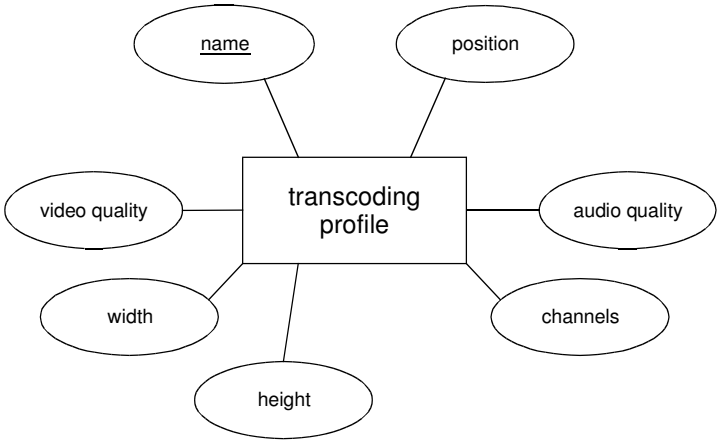

Figure 7.2: Entity-relationship diagram for transcoding profiles

Most of the column names directly map to the *ffmpeg2theora* options we discussed earlier. That leaves two new columns. The first is *name*, which obviously identifies profiles. The second is *position*, which is used to order them. We may, for example, want to sort our profiles by bandwidth usage. The first profile in this order will also be the default when playing back a clip.

7.3 Home Directories

Because we do not want users to access each other's uploaded files, each account will be given its own home directory. This is not apparent from the model above, as the path to a user's home directory will be constructed dynamically. To accomplish this, we will use a SQL *view*.

The view `ftp_accounts` will mirror the account table and add the `homedir` column to it. That column's value will be composed of a predefined path, followed by the numeric ID of the user. For instance, the user with ID *1001* would upload his files to `/mnt/ovis/in/1001`.

In Chapter 8, we will see how to configure ProFTPD to consult the view for authentication information.

8 FTP SERVER

Once we have our MySQL server up, we can install our FTP daemon ProFTPD and configure it to use the database for account information.

8.1 Installation

Installing ProFTPD on a modern Linux system should be a trivial process, as the software is generally available from the distribution's package repositories. On a Debian system, we can simply issue the command

```
# apt-get install proftpd
```

8.2 Configuration

Of course, there is the matter of further configuration. This section applies to Debian GNU/Linux with ProFTPD version 1.3.1; the configuration will be similar on other platforms. ProFTPD is well documented [45], so examining its manual is recommended.

Debian keeps ProFTPD's configuration files under `/etc/proftpd`. The main configuration file is called `proftpd.conf`, while SQL-related directives are traditionally kept in `sql.conf`.

First of all, the SQL module and MySQL driver need to be loaded.

```
LoadModule mod_sql.c
LoadModule mod_sql_mysql.c
```

Next, we tell the module to use MySQL and inform it how to connect to the database. We can reuse the account information from Section 7.1. Note that security can be increased by using an additional MySQL user account, which is only allowed the `SELECT` privilege on the `ftp_accounts` view.

```
SQLBackend mysql
SQLConnectInfo ovis@mysql.example.com sheep supersecret
```

We choose to store the FTP passwords without any encryption. Note that various encryption schemes *are* supported by ProFTPD. However, as discussed in Section 5.5, such a measure is unlikely to make our application any more waterproof.

```
SQLAuthTypes Plaintext
```

In our configuration, all users will belong to the same UNIX group and share the same user ID. Thus, their username will be the only thing setting them apart. We consult `/etc/passwd` and find that, in our case, the `ftp` user has ID *112* and belongs to group *65534*.

```
SQLDefaultUID 112
SQLDefaultGID 65534
```

Because all users now share the same user ID, they will be able to access each other's files. This is not what we want, so we will keep every user sandboxed in his home directory. For convenience, we will also have ProFTPD create home directories that do not exist.

```
DefaultRoot ~
CreateHome on
```

We can now instruct ProFTPD to use the database solely for user authentication, as the default group ID we configured applies to all of our users.

```
SQLAuthenticate users
```

Finally, we tell ProFTPD to retrieve authentication information from the account view, described in Section 7.2. We also specify its column names; we can omit the user ID and group ID, as well as the login shell.

```
SQLUserInfo ftp_accounts username password NULL NULL homedir NULL
```

Something we have not discussed yet is how to handle incomplete uploads. If we do not want users to be able to resume their uploads, we can have ProFTPD automatically remove the files.

```
DeleteAbortedStores on
```

If we choose not to enable automatic deletion, we will have to devise another method of cleaning up incomplete uploads. We could, for instance, periodically remove old files from the upload path.

8.3 Verification

Testing the configuration is trivial. We insert a row into the user table and start ProFTPD. Next, we attempt to log in, using our FTP client of choice. If successful, we verify that we cannot go up a level from our home directory, and that we can upload some files to it. If anything should go wrong, then the log files located in `/var/log/proftpd` may be of interest.

9 WEB SERVERS

In our configuration, we have two Web servers, which each serve a specific purpose: the streaming server and the application server. In both cases, the installation sequence is fairly trivial.

9.1 Streaming Server

The streaming server will be fairly spartan. Our first step is to install the Apache Web server. As discussed, alternatively, we could select the `lighttpd` package to save some resources.

```
# apt-get install apache2
```

Now, we can either configure some virtual hosts, or just place our media files under the document root. This is left as an exercise for the astute reader.

Additionally, we will install the Cortado applet on the streaming server. The applet must not reside on any other Web server, as Java applets are subject to a strict security policy which prevents them from accessing other hosts than the one they were served from.

Rather conveniently, the Cortado applet is available from Debian's repositories. All we need to type is:

```
# apt-get install cortado
```

This way, various versions will automatically be made available from the directory `/cortado` on our (Apache) Web server; in the case of lighttpd, manual configuration is required to enable it. We are only interested in the file `cortado-ovt.jar`, which offers support for **O**gg with **V**orbis and **T**heora.

Note that additional fine-tuning of the streaming server may greatly increase its performance. We will choose not to elaborate on this, as the configuration parameters heavily depend on the situation. The complexity of this matter would greatly exceed the scope of this project.

9.2 Application Server

Since the application server will act as a classic Web server, not much configuration is required. We will be using Apache 2 with support for PHP 5 and MySQL, so we type

```
# apt-get install apache2 libapache2-mod-php5 php5-mysql
```

Should further configuration of the Apache Web server be necessary, then the interested reader unfamiliar with this task is kindly directed to [46].

For the sake of convenience, we can easily install *phpMyAdmin* [44], which will allow us to manage our database using a Web interface.

```
# apt-get install phpmyadmin
```

After editing `/etc/phpmyadmin/config.inc.php` to our liking, we fire up our favorite Web browser and direct it to the URL `/phpmyadmin` on the application server.

10 TRANSCODING APPLICATION

In this chapter, we will discuss the development of the application responsible for transcoding user-supplied video files. We will identify several new dependencies as we go along. At the end of this chapter, we will then synthesize a list of required software for our transcoding server.

10.1 Basic Specifications

To process the transcoding queue sequentially, we will use a *Perl* script. Calling it simply `transcode` might make interfere with the Transcode suite discussed earlier in this document; thus, let us avoid all confusion and name the script file `ovis_transcode`.

Even though overhead would be minimal, the script will only be running as long as the transcoding queue has entries remaining. Thus, we will invoke it at fixed intervals; a *cron* job is ideal for this. Should the process end prematurely, then the cron job will make sure transcoding is resumed. However, invoking the script repeatedly might result in multiple simultaneous instances; we will discuss how to avoid this.

10.2 Initial Actions

This section will cover preparatory actions the script will take prior to the actual transcoding process. We will briefly discuss the implementation of each of these actions.

10.2.1 Option Parsing

The script will take a number of command line options, to override default paths, set a *nice* value for the transcoding process, etc. The `Getopt::Long` module [47] can be used to pass POSIX style options. It is commonly used in conjunction with `Pod::Usage` [48], which generates usage instructions.

In the following hypothetical code segment, we allow the user to turn on debugging output and specify a log file for the script. A typical argument list would be `--debug --log=mylog.txt`.

```
use Getopt::Long qw/GetOptions/;
use Pod::Usage qw/pod2usage/;
$log = 'log.txt';
$debug = 0;
GetOptions(
    'debug!' => \$debug,
    'log=s' => \$log
) or pod2usage(2);
```

In the code segment, we create some variables holding the options' values, which are subsequently passed (by reference) to `GetOptions()`. After the call, any command line options that were present will be reflected in the variables' values.

If the command line contains any invalid options, `GetOptions` will return a false value. In that case, the `pod2usage()` method will output usage instructions and exit with a return value of 2. In order to create the usage instructions, the `Pod::Usage` module merely outputs the documentation embedded in the script file, which uses Perl's standard *Plain Old Documentation* format [49].

Because of the exclamation mark after `debug`, the option may be negated. Thus, if the user wanted to explicitly turn off debugging, he could add the option `--nodebug`.

The equals sign after `log` denotes that the option is followed by a mandatory value; to make the value optional, we would use a colon. The letter `s` indicates that the value following the option is to be interpreted as a `s`tring; there is also support for `i`ntegers and `f`loating point numbers.

Options' names may also be abbreviated, provided that there is no ambiguity. For instance, if we added the option `localization` to the above declaration, the user would be able to abbreviate it to `--loc`, but not `--lo`, as the latter would conflict with the `log` option.

The `GetOpt::Long` module offers a number of additional features which are not relevant to this project, but which may be of interest nonetheless. The enthusiastic reader is directed to [47].

The reader interested in an exhaustive list of commands is kindly invited to type `ovis_transcode --help`.

10.2.2 File Locking

As mentioned, the script will be invoked repeatedly, meaning that we will end up with multiple simultaneous instances. As we do not want more than one transcoding task running at any given time, the code will have to check for other instances before starting the transcoding process.

A commonly deployed solution to this problem is to keep a write lock on a predefined file. If the script fails to obtain a write lock on that file, it can exit immediately. For convenience, the instance's process ID is generally stored in the lock file.

In Perl, we can implement such functionality using the `Fcntl` module [50]. Its `flock()` function requests a lock on a file handle and returns a true value upon success.

```
use Fcntl qw/:DEFAULT :flock/;
open(LOCKFILE, ">$filename") or die "Cannot open $filename: $!";
flock(LOCKFILE, LOCK_EX | LOCK_NB) or exit;
```

The `LOCK_EX` flag requests an *exclusive* lock, so only the process itself can access it. The `LOCK_NB` flag stands for *non-blocking*; if this flag were not set, the script would not be able to continue until it had successfully obtained the lock (and the call to `flock()` would always return a true value).

Finally, at the end of the script, we simply `close()` the file handle to release the lock.

10.2.3 XML Parsing

The script will obviously need to connect to the database server, as it contains the transcoding queue as well as the transcoding profiles. The information required to make this connection is stored in the XML document we discussed in Section 6.3.2.

The easiest way to parse the XML code is via the `XML::Simple` module [51]. With one simple call, we can turn our XML code into a Perl hash, representing the document tree.

```
use XML::Simple qw/XMLin/;
$hash_ref = XMLin("/path/to/datasource.xml");
```

If we were to serialize `$hash_ref` using the `Data::Dumper` module [52], we would get:

```
{
    'driver'   => 'mysql',
    'host'     => 'mysql.example.com',
    'port'     => '3306',
    'username' => 'sheep',
    'password' => 'supersecret',
    'database' => 'ovis'
}
```

Note that `XML::Simple` is far more versatile than one would initially expect; we are only using its most basic features. For an overview of the possibilities, the interested reader is directed to [51].

10.2.4 Database Interaction

Once we have extracted the database credentials, we can make the connection. The standard database layer for Perl is the `DBI` module [53].

To connect to a database using DBI, all we need is a compatible *data source name*. We can construct that from the contents of the XML document.

```
use DBI qw/:sql_types/;
$dsn = "DBI:$driver:host=$host;port=$port;database=$database";
$dbh = DBI->connect($dsn, $username, $password)
    or die("Connection failed: $DBI::errstr");
```

Next, much like with any other database layer, we can execute queries directly. Bound values may be passed as parameters. The number of rows affected by the query is returned. Note that if no rows were affected, the method returns `"0E0"`; the number `0` is reserved for errors, which can be caught with a simple `or` operator.

```
$rows = $dbh->do("DELETE FROM transcoding WHERE clip_id = ? OR state = ?",
    $clip_id, $state) or die($dbh->errstr);
```

Queries that return a row set must be prepared and subsequently executed. Next, the resulting rows can be fetched as a reference to either an array or a hash.

```
$sth = $dbh->prepare("SELECT * FROM clips WHERE state = ?")
    or die($dbh->errstr);
$sth->execute($state) or die($dbh->errstr);
while ($row = $sth->fetchrow_hashref()) {
    print "Clip ID: ", $row->{"clip_id"}, "\n";
}
$sth->finish();
```

In this example, we also used a *parameter*: the question mark acts as a placeholder for the value of the `state` column. Next, we passed that value to `execute()`. Had there been multiple question marks, then we would have had to pass each of their values, respecting their order in the query.

For a more conservative approach, we can also specify the type of each column. Instead of passing the values to `execute()`, we explicitly call `bind_param()` for each parameter. Thanks to the `:sql_types` tag after `use DBI`, we have access to constants which allow us to specify the type of a parameter. Thus, to assign a value to the first and only parameter, and subsequently execute the statement, we would type:

```
$sth->bind_param(1, $state, SQL_INTEGER);
$sth->execute() or die($dbh->errstr);
```

After the statement has been executed, we proceed by calling `fetchrow_hashref()` like in the example above.

Finally, it is considered good practice to explicitly close the connection when we are done.

```
$dbh->disconnect();
```

10.3 Queue Processing

10.3.1 File Inspection

The `ffmpeg2theora` executable can resize a video stream during transcoding, but not using the logic we described in Section 5.11. Therefore, we will need to examine the file prior to transcoding, calculate the destination width and height manually, and then pass them to `ffmpeg2theora`.

For this purpose, we will deploy *MPlayer*. The `-identify` switch triggers the output of information that can easily be parsed. We do need some additional options to speed up the process:

```
$ mplayer -identify -frames 0 -vo null -ao null -vc null /path/to/infile.avi
```

The `-frames` switch tells MPlayer how many frames to process, in our case none. To suppress audio and video output, we use `-ao null` and `-vo null`. We can also avoid loading a video codec by adding `-vc null`, as the required information about the video stream is part of the container itself. However, we do not add `-ac null` at this point, as the audio codec will tell us the number of channels. Finally, we need to pass the input filename; note that it will probably reside on a mounted network volume.

Among the output of the `mplayer -identify` command will be lines looking like

```
ID_VIDEO_WIDTH=640
ID_VIDEO_HEIGHT=480
ID_LENGTH=123.40
ID_AUDIO_NCH=2
```

which tell us the pixel width and height of the clip, as well as its duration and number of audio channels, which we will use later on. Incidentally, `ID_VIDEO_ASPECT` would tell us the aspect ratio directly, but since we need the clip's dimensions anyway, we can just do the arithmetic ourselves.

10.3.2 Transcoding

Once we have obtained the required information, we can finally invoke `ffmpeg2theora` to do the actual transcoding. The command line is fairly straightforward; a typical example would be:

```
$ ffmpeg2theora
    --audioquality 0 --channels 2 --videoquality 2 --width 320 --height 180
    --output /path/to/outfile.ogg
    /path/to/infile.avi
```

If we only wish to encode a certain range, we can add the `--starttime` and `--endtime` switches, each followed by a timestamp in seconds. Also note that we choose not to alter the clip's frame rate.

10.3.3 Thumbnail Generation

Finally, to create a thumbnail image for the video clip, we can again use MPlayer, be it with a slightly different set of switches. This time, the command line is:

```
$ mplayer -nosound -vo jpeg:outdir=/path -frames 1 -ss 12.34 /path/to/infile.avi
```

This time, we are only interested in the clip's video. Thus, we disable audio altogether by using the `-nosound` option, which is equivalent to `-ac null -ao null`, and set the video output driver to `jpeg`. This will generate an image file for each frame in the directory specified by the `outdir` option; we can also specify a numeric JPEG quality setting, but the default setting will do. We do not use the `-vc` switch, as we want MPlayer to determine the used video codec automatically.

Because we only need one thumbnail image, we use `-frames 1`. Rather than generating an image of the very first frame, we will seek to a time in seconds, passed using the `-ss` switch; the timestamp corresponding to the exact middle of the clip is easily deduced from the duration we obtained earlier.

Sadly, we do not have any control over the name of the file this command generates; it will always be `00000001.jpg`. As the script is limited to a single instance, this is not a major inconvenience.

There is however another problem: when using the `jpeg` output driver, the image is not resized, even if one specifies dimensions on the command line. Thus, rather than just renaming `00000001.jpg`, we will actually resize it using the logic described earlier, save the resulting image and dispose of the original. To simplify the task of resizing images, the `Image::Resize` Perl module [54] exists.

10.3.4 Post-Transcoding Actions

Once the script has transcoded a file, it can update the associated clip's record in the database, which will render the clip public. In addition, if no other queue entries share the current source file—remember that a file can act as a source for multiple segments—, we can remove it.

Recall that we decided to notify the author of a clip after transcoding. Rather than sending e-mails straight from the Perl script, we will take a generic approach. The `--onsuccess` option will allow one to specify the command to execute after a transcoding job completes. Similarly, the `--onfailure` option will handle hiccups. Whenever such a command is invoked, it will take the clip's identifier as a command line argument.

One might argue that the `--onsuccess` and `--onfailure` command line options could be aggregated into a single one, e.g. `--oncomplete`. After all, the command only receives the clip identifier. Therefore, it is likely that it will connect to the database to obtain additional information, including the clip's state. However, the separation allows for maximum extensibility. We could, for instance, add an option for a command to invoke before a transcoding job is started, and/or specify a different list of command line arguments in each case.

Finally, if any jobs remain, the script starts transcoding the next clip in the queue. Otherwise, the queue is empty, meaning it can safely exit—until the next invocation of the cron job.

10.4 Shared Code

In Chapter 11, we will develop two additional utilities which share some features with the transcoding script. As it makes sense to harbor common code in a separate library, some of `ovis_transcode`'s features are actually implemented in a *Perl module* called `Ovis`. It mostly contains static methods related to database interaction and logging. We will not elaborate on the development of Perl modules here; the interested reader is kindly directed to Perl's documentation on the subject, e.g. [55].

To import the module into our scripts, we simply add the line `use Ovis;` to them. However, there is a catch. Perl looks for modules in its *include path*, which is unlikely to contain the directory where `Ovis.pm` is located. This can be sorted out in a number of ways. For instance, we can

- move `Ovis.pm` to `/usr/local/lib/perl5`, the standard location for manually installed modules on Debian,
- add the path where `Ovis.pm` resides to the `PERLLIB` or `PERL5LIB` environment variable, or
- add the `use lib` pragma to all of the scripts, e.g.

```
use lib "/path/to/perl/modules";
```

Note that `use lib` should be a last resort, as the modifications will need to be repeated after an Ovis update.

10.5 Software Installation

10.5.1 Dependencies

Over the course of this chapter, we have named a couple of supporting packages for the transcoding script. The following command will take care of their installation on Debian:

```
# apt-get install
    mplayer ffmpeg2theora perl
    libxml-simple-perl libdbi-perl libdbd-mysql-perl
```

At the time of this writing, the `Image::Resize` Perl module is not available from Debian's APT repositories. However, it does have the *GD* library, which the module depends on. Thus, we install its Perl bindings:

```
# apt-get install libgd-gd2-perl
```

Then, as `Image::Resize` itself is listed on CPAN [54], we can install it using the CPAN shell:

```
# cpan Image::Resize
```

10.5.2 Scheduled Execution

When all the dependencies have been taken care of, we can set up the cron job. Those unfamiliar with cron can find more information at a number of sources, for instance [56].

Note that the script needs to run under a user account that has write access to both the source file server and the destination file server; we discussed how to use NFS for this in Section 6.3.1. A dedicated user account, e.g. `ovis`, is generally a good idea.

Now, say we want to execute `ovis_transcode` every 5 minutes as the `ovis` user. We start by typing:

```
# crontab -eu ovis
```

This fires up our text editor of choice, allowing us to the edit our personal *crontab* document. We then add the line below, save the document and exit the editor, which will schedule the job.

```
*/5 * * * * ovis_transcode
```

This entry assumes that the shell will be able to find `ovis_transcode` using the `$PATH` environment variable. Alternatively, we could just specify the full path. Also, command line options may follow.

11 SUPPORTING UTILITIES

In this chapter, we will briefly discuss the development of two additional utilities that support our platform. The first will be used for maintenance, the second will send out an e-mail after every transcoding job.

11.1 Janitor

In Section 5.5, we discussed the introduction of FTP accounts which expire after a given time period. While it is straightforward to check an account's expiry date upon access, we do not want the FTP account table to fill up with expired accounts. Thus, we will develop a simple utility to remove expired accounts.

In addition, if a clip's database record disappears, we want the associated video file to be removed as well. Thus, the same utility will also remove video files and thumbnail images that no longer have an associated database record. Note that the demonstration portal does not allow users to remove their clips; however, the Dokeos version of our software will rely on it, as will most third-party implementations, presumably.

The utility will be called `ovis_janitor`, for obvious reasons. Like the transcoding application, it will be a *Perl* script, which is invoked periodically using *cron*. As mentioned in Section 10.4, it will rely on `Ovis.pm`.

Upon execution, `ovis_janitor` will perform two tasks:

- remove records from the `upload_accounts` table whose `expires` field is less than the current time, and
- retrieve all clip IDs from the `clips` table and use them to remove stray files from the destination file server.

If we place the script on the transcoding server, we do not have to worry about dependencies or file system shares, as `ovis_transcode`'s requirements include them all. To increase performance and/or minimize network traffic, the script *could* of course be placed on a different server if necessary.

Thus, `ovis_janitor` will optionally take the path to a data source definition file, as well as the mount points for the directories containing video files and thumbnail images on the destination file server. Evidently, these command line options will again be parsed using the `Getopt::Long` module we discussed in Section 10.2.1.

11.2 Postman

Section 10.3.4 mentions options to `ovis_transcode`, which instruct it to execute any command after a transcoding job has been carried out. In the case of our demo, we want this command to send out an e-mail to the author of the clip in question. So, let us develop such a script.

Note that, as opposed to `ovis_janitor`, the postman script will be specific to the demonstration platform. Needless to say, though, that other installations would probably be able to use it as a starting point.

11.2.1 Basic Specifications

Yet again, we will write a simple *Perl* script, based on `Ovis.pm`; this one, we will call `ovis_postman`. We have already outlined its purpose: to retrieve useful information about the clip and its author from the database and to inform that person of the result via e-mail. Database access was covered in Section 10.2.4, so we only have to find out how we can send out the e-mail.

11.2.2 Sending E-Mail

A clean and object-oriented way to send e-mails from a Perl script is the `MIME::Lite` module. It automatically selects the most suitable method to send the message; on UNIX systems, this means invoking the system's `sendmail` MTA, while on Windows, the default SMTP server is contacted directly. [57]

Sending a simple text e-mail using `MIME::Lite` could not be more straightforward. We first create an object representing the message and then call its `send()` method.

```
use MIME::Lite;
$mail = new MIME::Lite(
    'From'    => 'Ovis Demo <demo@ovis.pwnt.be>',
    'To'      => 'john.doe@biology.university.edu',
    'Subject' => 'Success: "A Grazing Sheep"',
    'Data'    => 'Your video clip "A Grazing Sheep" is now available!'
);
$mail->send();
```

By explicitly passing a method to `send()`, we can override the default. Although this can cause compatibility issues, it may, for instance, be tempting to force the use of a specific SMTP server instead of `sendmail`:

```
$mail->send('smtp', 'smtp.example.com');
```

Note that, as its name suggests, `MIME::Lite` is capable of adding attachments to messages and sending rich text. Thus, we could, for instance, create a nicely styled message, including the thumbnail image that was generated, or even the Cortado applet itself. However, not everybody appreciates rich text e-mail.

11.2.3 Software Installation

To use `ovis_postman`, we will first need to install the `MIME::Lite` module. In addition, we will install *Postfix*, a well-known replacement for *Sendmail*, without the notoriously tedious configuration process.

```
# apt-get install libmime-lite-perl postfix
```

The actual configuration of Postfix greatly depends on the network infrastructure in question. Therefore, we will not elaborate on this particular task; introductory information may be found at [58], among others.

The next step is making `ovis_transcode` invoke `ovis_postman` after every transcoding job. In Section 10.3.4, we discussed the `--onsuccess` and `--onfailure` options, which we will now use.

If we were to pass `ovis_postman` to `ovis_transcode` directly, we would be overlooking something. Just like our other two Perl scripts, `ovis_postman` will be accessing the database. As such, it takes the path to a data source definition file, as described in Section 6.3.2, as a command line option.

If we are dissatisfied with the default path `/etc/ovis/datasource.xml`, we will need a trivial *wrapper* script. Let us aptly name it `ovis_postman_wrapper`—sounds esoteric, no? Its contents speak for themselves:

```
#!/bin/sh
/path/to/ovis_postman --datasource=/path/to/datasource.xml $1
```

Then, we can instruct `ovis_transcode` to invoke that when appropriate. Thus, the command line becomes

```
$ ovis_transcode [options]
    --onsuccess=/path/to/ovis_postman_wrapper
    --onfailure=/path/to/ovis_postman_wrapper
```

Note that the same wrapper script handles both events. After all, `ovis_postman` can use the clip's identifier to retrieve all the necessary information about it, including its state. Thus, based on that information, the script can send the appropriate notification to the clip's author.

12 UPLOAD APPLICATION

In this chapter, we will discuss the development of the Java application for uploading video clips to an Ovis-based streaming video platform.

12.1 Reusable Software

In Section 5.4, we decided to use the FTP protocol for uploading clips to the platform.

As FTP is common, initially, it would seem as though there are a number of open source Java FTP applications and applets, which could be adapted slightly to suit our needs: [59], [60] and [61], to name but a few. Unfortunately, closer examination reveals that these are either poorly designed or too complex. Therefore, borrowing a full-blown graphical user interface is out of the question.

Fortunately, we do not need to develop the raw FTP layer itself. A number of libraries exist for this purpose, as laid out in [62]. We will select the open source *Apache Commons Net* library [63], because the Apache Commons project is well respected and actively maintained.

Rather than focusing on uploading video clips straightaway, we will first develop a generic Java application for uploading files via FTP; this will be discussed in Section 12.2. The application will subsequently serve as the base of the video clip uploader. In Section 12.3, we will see how to accomplish this.

12.2 Generic FTP Upload

We will now elaborate on the development of a simple Java application for transferring a set of arbitrary files to an FTP server. The resulting application may be of use to those looking for a simple FTP upload client, both as an applet and as a standalone application.

As mentioned, the software will rely on the *Apache Commons Net* library. The application's GUI will be built using the standard *Swing* [64] toolkit. In addition, to provide a familiar look and feel, we will make use of free standardized icons from the *Tango Desktop Project* [65].

12.2.1 Main Scenario

A typical usage work flow will take place as follows:

1. The user enqueues a number of files.
2. The user indicates that he desires to start the upload process.
3. The application connects to the FTP server.
4. The application logs in.
5. The application changes the working directory if necessary.
6. The application uploads the enqueued files sequentially.
7. The application marks processed uploads as completed.

12.2.2 Alternate Scenarios

Firstly, the user may choose to *abort* the upload process at any time. We will use Java's standard thread model, which is documented in [66], to achieve this.

Secondly, the FTP protocol also supports *resuming* uploads, using the `APPE` command. Unfortunately, due to the limited time frame of this project, this subject will be left untouched.

12.2.3 Architecture

The FTP uploader is defined in the Java package `be.pwnt.ftp`. It contains a number of subpackages, the most important of which are

- `be.pwnt.ftp`, which contains the core class `Transfer`, as well as the `TransferList` interface,
- `be.pwnt.ftp.net`, which is in charge of everything related to network communication, and
- `be.pwnt.ftp.ui.swing`, which provides the Swing-based GUI.

Let us briefly discuss the contents of the `net` and `ui.swing` packages.

12.2.3.1 Networking Classes

The class `UploadClient` is the application's gateway to the Apache Commons Net libraries' FTP client. Its purpose is to log into an FTP server, change the working directory and upload a single file at a time. Using an *observer pattern*, it notifies its registered `TransferProgressListener` instances of uploaded byte amounts. An instructive example of a Commons-based FTP client can be found in [67]. For implementation details, the astute reader is kindly directed to the source code of `UploadClient`, which accompanies this document.

To transfer multiple files, `BatchUploadClient` is introduced. This class is similar to `UploadClient`, but is able to connect to a server and upload a `TransferList` with a single call to `transfer()`. It extends `BatchClient`, which, through a more elaborate observer pattern, notifies `BatchClientListener` instances of every event.

12.2.3.2 GUI Classes

The graphical upload application is started by loading the main class, `Uploader`. This merely creates a `JFrame` containing an `UploadPanel`. The latter is a `JPanel` which sets up Swing components, as is customary. It also creates a `BatchUploadClient` and acts as its `BatchClientListener`; as such, the upload application more or less implements a *model–view–controller* architecture.

The most important part of the GUI is the list of transfers. It is visualized using an extent of `JTable` called `TransferTable`. Its model is a `TransferTableModel`, which is not only a Swing `AbstractTableModel`, but also implements the `TransferList` interface. Thus, it is directly passed to the `BatchUploadClient`.

To display icons and progress bars in the transfer table, a set of `TableCellRenderer` classes is used. These are contained in the subpackage `be.pwnt.ftp.ui.swing.renderer`.

12.2.4 Speed Estimation

The FTP upload application keeps a *sliding window* of byte amounts transferred, which is updated at a fixed frequency. Thus, it remembers the most recently transferred amounts and, at fixed time intervals, discards the oldest among those amounts in favor of a newly added one. If the sliding window $\{ b_1\ b_2\ \ldots\ b_n \}$ is updated after every time interval t, the average connection speed v is readily obtained using the expression

$$v = \frac{\sum_{i=1}^{n} b_i}{n\,t}$$

Dividing the number of bytes remaining by this speed then yields an estimation of the required transfer time.

12.3 Video Clip Upload

As mentioned earlier, the generic FTP upload application we just discussed will now serve as the base of the application for uploading video clips. Thanks to the modular design of the generic upload application, we can seamlessly integrate the necessary classes.

12.3.1 Additional Features

There are three main features that Ovis's upload application adds to the generic FTP uploader. These are all implemented in the `be.pwnt.ovis.upload` package.

- The user can define a list of clip segments to create from the video file. By default, this list contains one segment, namely the entire clip.
- When files are added, they quietly receive a randomized destination file name. This way, chances of name collisions are slim. If the transfer fails, the name is again randomized, as a precaution.
- Upon completion of an upload, the Web service is notified. Additionally, to ensure that the Web service is available, its version information is requested at startup.

12.3.2 Architecture

The package model of Ovis's application for uploading video files is more or less similar to that of the generic upload application. Key packages that comprise the application are

- `be.pwnt.ovis.upload`, which contains core classes,
- `be.pwnt.ovis.upload.ui`, which extends the generic uploader's Swing GUI, and
- `be.pwnt.ovis.upload.soap`, which provides classes for accessing the Web service.

Let us now take a closer look at the classes contained in each of those packages.

12.3.2.1 Core Classes

We significantly extend the generic FTP upload application's model by introducing the `ClipSegment` class in `be.pwnt.ovis.upload`. Evidently, this will allow us to represent segment information, i.e. a segment's title, along with its start and end time. For convenience, those times are represented by the newly introduced class `Timestamp`, which easily switches between human-readable time representations and (internal) long integers.

The transfer of a file which may or may not describe multiple segments is modeled using the `ClipTransfer` class; evidently, it extends the generic `Transfer` class from Section 5.4. In addition, its constructor assigns the transfer a randomized destination file name; this way, its name will almost definitely be unique.

12.3.2.2 GUI Classes

The user interface for the clip upload application is displayed in Figure 12.2 at the end of this chapter. In the `be.pwnt.ovis.upload.ui` package, we find `ClipUploader`, which is the application's main class. Similar to its generic counterpart `Uploader`, it is merely a `JFrame` for a `ClipUploadPanel`. The panel in its turn is an extent of the generic `UploadPanel`, adding two features to it.

Firstly, until a transfer is initiated, the progress bar is replaced with a simple hyperlink. If the user either clicks that link or double-clicks the transfer, a `ClipPropertiesDialog` will pop up. This is a standard `JDialog`, which contains a (fairly complex) editable `JTable`, allowing the user to manage the segment list. The dialog window is displayed in Figure 12.3 at the end of this chapter.

Secondly, `ClipUploadPanel` is responsible for notifying the Web service. Recall that the generic `UploadPanel` implemented `BatchClientListener`. One of that listener class's methods is `transferStateChanged()`, which gets invoked whenever the state of a transfer in the `TransferList` changes. Thus, `ClipUploadPanel` overrides this method to contact the Web service whenever a transfer receives the 'completed' state.

Making contact with the Web service is expected to complete momentarily. Nonetheless, while waiting for the operation to complete, the GUI becomes unavailable. Therefore, the helper class `TaskAwaitingDialog` is introduced. This is another `JDialog`, which takes an implementation of the interface `AwaitedTask`, and waits for it to complete. If the awaited task is still running after a certain period of time, the dialog window pops up, waiting for the task to finish. This way, the user will not be bothered unless absolutely necessary.

12.3.2.3 Web Service Classes

Traditionally, creating a Web service client is left to a Java IDE for the most part. *Eclipse*, which is a popular choice for Java (and other) development, can take a WSDL document[6] and set up a SOAP client, provided that we first install its *Web Tools Platform*. [69]

As documented in [70], we simply supply the URL to the WSDL document and let Eclipse generate the supporting classes. This also adds the required *Apache Axis2* [71] libraries to our project. We can omit some of those libraries, as well as large parts of the resulting code, as we will not be using all the functionality the generated client has to offer. This will limit the download size a bit.

Note that we have yet to develop the Web service itself; this will be discussed in Section 13.8. Nonetheless, we can already note that the Java package `be.pwnt.ovis.upload.soap`, containing the client side to this Web service, contains two classes. The first is `Proxy`, which facilitates access to the Web service. The second is called `Segment`; it is a serializable *JavaBean* class, which represents segment information.

12.4 Signed JAR Files

As discussed in Section 5.3, we will be making use of Java Web Start to launch our application straight from the Web browser. Our application will require access to the local file system, to obtain the contents of the files to upload. Consequently, Java Web Start requires that we sign *all* of our JAR files, in order to allow users to verify their authenticity. [72]

The process is described in [73]. JAR files can be signed using *keys*. The authenticity of these keys is verified by *digitally signing* the files using *X.509 certificates*. In this section, we will look at how to generate a key for ourselves and use it to sign the JAR files.

12.4.1 Generating Keys

First, we need to have our own *key*. Keys are stored in a *keystore*, which is protected by a *password*. Each key is identified by an *alias*.

Keystores can be managed using the `keytool` utility. Its most basic invocation goes

```
keytool -genkey
```

After answering the questions, we will have a keystore called `.keystore` in the current user's home directory. It will contain a single key, identified by the current username. We can specify an alternate alias using the `-alias` switch and override the keystore path using the `-keystore` switch.

Obviously, `keytool` can do a lot more. The interested reader is kindly directed to the full list of commands, available by invoking `keytool -help`.

6 *Web Service Definition Language (WSDL)* documents formally describe a Web service's interface. [68]

12.4.2 Signing JAR Files

Now that we have our key, we can use it to sign the JAR file. The utility to use here is `jarsigner`. If we want to sign `unsigned.jar` using the key with the alias `bob`, we issue

```
jarsigner -signedjar signed.jar unsigned.jar bob
```

Note that using the same path for the unsigned and the signed JAR file will throw an error.

Like `keytool`, this utility offers the `-keystore` switch. The password for the keystore and the key can also be passed on the command line, using the `-storepass` and `-keypass` switches, respectively.

12.4.3 Certificate Authorities

The simple process described above allows us to sign our JAR files. However, there is a catch. If we launch the application from Java Web Start—we will see how in Section 13.7—, we end up with a warning dialog like the one displayed in Figure 12.1.

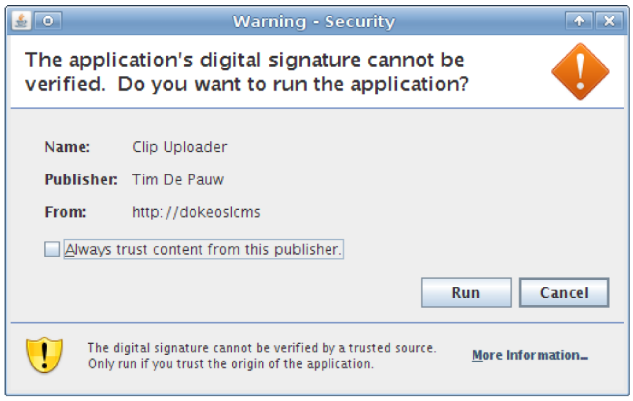

Figure 12.1: Java Web Start authenticity warning

The reason for this is that we are using a *self-signed* certificate. To avoid getting the warning, we would have to obtain a digital signature from a trusted *certificate authority*. We will not elaborate on this common process; the interested reader is kindly directed to [74].

12.5 Impressions

Since a picture is allegedly worth more than a thousand words, we conclude this chapter with two screen captures of the clip upload application.

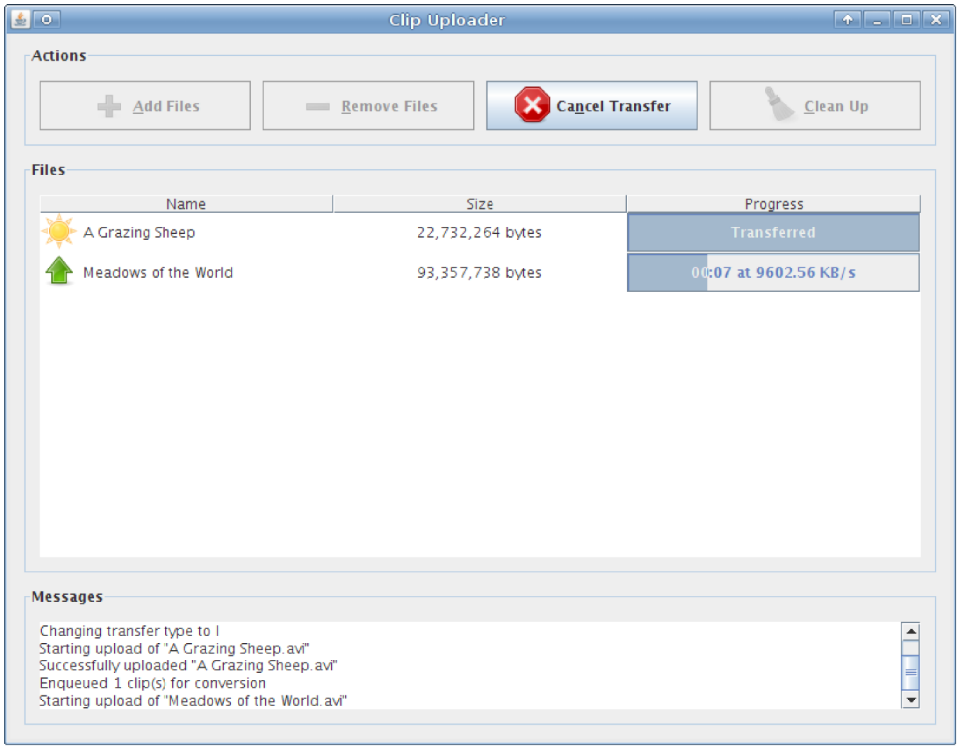

Figure 12.2: Clip upload application in action

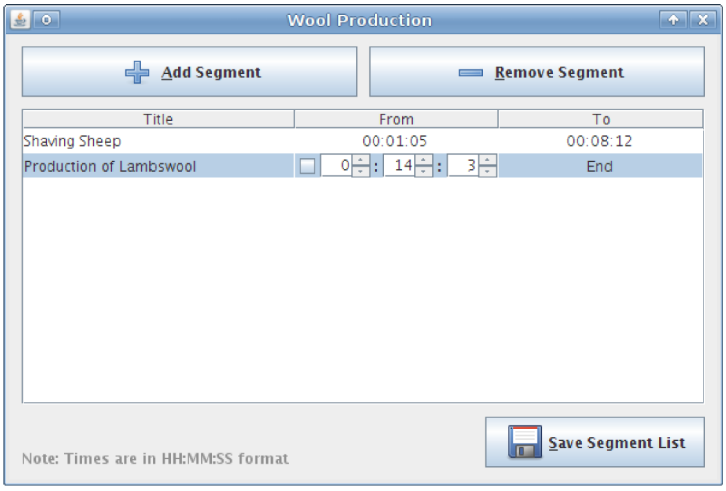

Figure 12.3: Segment editing in the clip upload application

13 WEB APPLICATION

Our servers are running, our transcoding application is waiting for input and our upload application is ready to be deployed. Hence, it is finally time to put it all together and start developing the Web front-end.

In addition, we will take this opportunity to experiment with features newly introduced in PHP 5, which may also be of interest for future Dokeos versions.

13.1 Class Names

A common problem in PHP is that there is no direct mapping between classes and file names. As such, every source file that references classes defined elsewhere must `require` the relevant source files. This rather tedious task can be avoided.

Every time PHP encounters an unknown class name, it passes that name to the `__autoload()` function. [77] We will define the function in such a way that there is a direct relation between the class name and the path to the associated source file. For instance, the class `Ovis_Foo_Bar` would automatically be loaded from the path `Ovis/Foo/Bar.class.php`. As PHP does not support name spaces[7], this does mean our class names will turn out fairly long. Setting PHP's include path [78] appropriately will also improve code readability.

There is one major downside to the `__autoload()` technique. Once the function has been defined, there is no way to extend it. This means that, if two sets of classes each rely on their own `__autoload()` function, they cannot be used simultaneously without again redefining the function, so it combines the functionality of both. This is just another shortcoming of PHP that developers have to live with.

13.2 XML Parsing

We will again have to parse the data source definition file from Section 6.3.2. In Perl, we used `XML::Simple`, which we described in Section 10.2.3. PHP offers a similar API, called *SimpleXML* [79].

7 Name space support was planned for PHP 5 at first, but then delayed until version 6. [75] [76]

To parse our XML file, we first read it into a string, which we then pass to `SimpleXMLElement`'s constructor.

```
$string = file_get_contents("/path/to/datasource.xml");
$element = new SimpleXMLElement($string);
```

This will parse the file's contents. Then, much like with `XML::Simple`, we can access the child nodes' values as properties of the `SimpleXMLElement`. For instance, we would obtain the database name as follows:

```
$database = $element->database;
```

13.3 PHP Data Objects

For good measure, we will make abstraction of the data layer using the PHP extension *PHP Data Objects (PDO)* [80]. A number of alternatives exist, most notoriously *MDB2* [81] and *ADOdb* [82]. PDO, however, ships with PHP by default and offers a fairly clean interface. Note that when we installed MySQL support for PHP in Section 9.2, we also implicitly added it to PDO.

PDO's basic approach is similar to that of its Perl equivalent DBI, which was discussed in Section 10.2.4. The first step is to create a connection object by instantiating the `PDO` class.

```
$options = array(
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_PERSISTENT => true
);
$dbh = new PDO(
    "$driver:host=$host;port=$port;dbname=$database",
    $username, $password,
    $options
);
```

Apart from the standard connection information, we pass some options to the constructor. While a lot of developers are not aware of this, PHP 5 has a full exception model. [83] By setting the `PDO::ATTR_ERRMODE` option to `PDO::ERRMODE_EXCEPTION`, we enable exception-based error reporting. Using `set_exception_handler()`, we can then define a single exception handler for the entire application. [84] The `PDO::ATTR_PERSISTENT` option makes connections persistent, which can improve performance.

Once we have our database connection, we have 3 object methods at our disposal for executing queries. The first is the trivial `exec()` object method, which is suitable for queries that do not return a row set.

```
$dbh->exec("DROP TABLE users");
```

The second method is `query()`, which provides equally trivial support for queries that do return one or more rows. As the `PDOStatement` class implements the `Traversable` interface, PDO's result sets can be iterated over using the `foreach` statement.

```
foreach ($dbh->query("SELECT * FROM clips") as $row) {
    echo $row["clip_id"], "<br/>";
}
```

The most versatile method is to use prepared statements. First, using the `prepare()` method, we obtain a `PDOStatement` object. In the example below, we also use a *named parameter* called `:source_file`.

```
$sth = $dbh->prepare("SELECT * FROM transcoding WHERE source_file = :source_file");
```

The obtained statement may then be used one or more times. Each time we want to use it, we first assign a value to each named parameter. Next, we `execute()` the statement and `fetch()` the resulting records.

```
$sth->bindParam(":source_file", $source_file);
$sth->execute();
while ($row = $sth->fetch()) {
    echo $row["clip_id"], "<br/>";
}
```

Note that `bindParam()` takes two additional parameters, namely the parameter's type (a constant from [85]) and its maximum length. If omitted, they are deduced from the variable. Thus, a safer approach would be:

```
$sth->bindParam(":source_file", $source_file, PDO::PARAM_STR, 255);
```

In the example above, we treated `$row` as an associative array. Alternatively, a number of alternate *fetch modes* are available, which some may prefer. They are activated by specifying `fetch()`'s first parameter. [86]

Once we are done with a statement or connection object, we simply destroy the variable and let its destructor take care of freeing up resources:

```
unset($sth);
```

Note that PDO does not provide the means to limit a row set's size (i.e. `LIMIT x, y` in MySQL), or to quote identifiers (e.g. database, table and column names) in a driver-specific fashion. Therefore, if one is unwilling or unable to adapt one's queries to support different database vendors, MDB2 is likely to be a better choice. However, as the MDB2 developers themselves admit, this sort of abstraction will sometimes cause more problems than it solves. [87]

13.4 QuickForm

Dokeos uses the `HTML_QuickForm` [88] package for an object-oriented approach to HTML forms. We will use its successor `HTML_QuickForm2` [89], which is a PHP 5 rewrite. While it is still in alpha testing, it is functional and offers a few new concepts worth investigating.

The documentation for QuickForm2 is largely unfinished, but the development wiki provides some insight into the API. [90] In addition, the package comes with some useful examples. [91] [92]

QuickForm's basic idea remains the same in this revision. The form is an object, which contains several form element objects. Upon submission, the form's `validate()` method will not return true until each individual element validates. The validation API itself, however, has changed quite a lot. An important improvement is the ability to chain validation rules. For instance, consider a text field that can either be left empty, or filled in with at least 4 characters. This would translate to the following compact statement:

```
$field->addRule("empty")
    ->or_($field->createRule("minlength", "Too short!", 4));
```

Similarly, there is an `and_()` method, to require that both conditions be met. A single statement can chain any number of rules, as long as operator precedence is taken into account. In addition, code readability can be an issue with complex rules, like some examples in [92].

Just like in the original QuickForm, several predefined rules are available. For example, we may create rules based on regular expressions or implement our own callback function.

13.5 Model-View-Controller

While several *MVC frameworks* exist for PHP [93], we will be setting up our own basic MVC implementation; this way, our code's legibility will not be burdened by the introduction of an additional feature set. We are not developing a considerably large application anyway, and the Dokeos code base does not use any MVC frameworks either at this point.

First off, all classes that are only related to the demonstration portal (and not part of the underlying libraries) will have a name starting with `Ovis_Demo`. This will allow us to clearly separate the demonstration portal from the underlying libraries, and even distribute the libraries individually.

Classes belonging to the *model* will all be contained in the libraries. Thus, they will simply start with `Ovis`. An example is the class representing clips, `Ovis_Clip`.

The *view* classes will all have names starting with `Ovis_Demo_View`; for instance, `Ovis_Demo_View_Register` would be the registration page. In addition, the classes will all extend the abstract class `Ovis_Demo_View` itself, thus making various UI-related methods available directly from the view object.

Finally, the *controller* classes have names starting with `Ovis_Demo_Controller`. As is often the case, many of their methods simply delegate the task to the data manager.

13.6 URL Rewriting

To make URLs more readable, as well as to optimize them for indexing by search engines, a lot of Web sites rely on the `mod_rewrite` Apache module [94]. This module provides the ability to redirect URLs internally, based on regular expressions.

By creating a file called `.htaccess` in the application's root directory, we can apply some additional Apache directives to it. [95] The `mod_rewrite` module adds a number of these directives.

As `mod_rewrite`'s feature set is rather extensive, we will refrain from covering it in great detail. Instead, we will briefly discuss the directives required for our application. The reader interested in harnessing the rewrite module's true potential is invited to examine [96].

The first directive is necessary to initialize the rewrite engine.

```
RewriteEngine On
```

Next, we will set the base URL for rewriting. This is not necessary in all cases, but can avoid problems. Note, however, that the path will need to be modified if the application is not directly under the server root.

```
RewriteBase /
```

We will redirect every request for a nonexistent path to `index.php` using the following directives:

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . index.php%{REQUEST_URI} [L]
```

When a path is requested, we first ensure that it is neither a regular file, nor a directory. If both conditions are met, the path is matched against our (trivial) regular expression. The value of `REQUEST_URI` is an absolute path, so it will always start with a forward slash. Consequently, we can directly append it to the URL, storing it in the `PATH_INFO` environment variable for `index.php`.

Then, `index.php` can take over. Using a *factory pattern* in the `Ovis_Demo_View` class, the requested path will be split up and interpreted as a view URL, followed by optional arguments.

For instance, if we were to request the URL `http://example.com/foo_bar/hello/world`, an instance of the class `Ovis_Demo_View_FooBar` would automatically be created. The object's argument list would subsequently be set to `array('hello', 'world')`. Finally, its `display()` method would be called, rendering the page.

For implementation details, the avid reader is invited to examine the source code of the view classes, which accompanies this document.

13.7 Java Web Start

In Section 12.4, we signed all our JAR files, so we could launch the clip uploading application through Java Web Start and allow it access to the local file system. But how do we invoke Java Web Start?

As explained in [72], Java Web Start is launched when a *JNLP* document is encountered. *JNLP* stands for *Java Network Launching Protocol* and is Sun's protocol which defines the launching mechanism for Java Web Start applications. A JNLP document is an XML document that defines the application's characteristics.

By serving documents with the `application/x-java-jnlp-file` MIME type, we can tell the Web browser to launch Java Web Start—provided that the Java Runtime Environment has been installed. Just to be safe, we also add a `Content-Disposition` HTTP header to suggest a file name that ends in `.jnlp`.

The general structure of a JNLP document is as follows.

```
<jnlp spec="1.0+"
        codebase="http://example.com/" href="web_start/1001/temp456">
    <information>
        <!-- General information about the application -->
    </information>
    <security>
        <!-- Security settings for the application -->
    </security>
    <resources>
        <!-- Resources that the application will use -->
    </resources>
    <application-desc main-class="be.pwnt.ovis.upload.ui.ClipUploader" />
</jnlp>
```

The document's root element contains the URL to the code base, and a relative URL to the JNLP file itself. Because we cannot pass FTP authentication any other way, and we do not want to burden our users with supplying it manually, it is included in the URL; in Section 5.5, we discussed why this is acceptable. Thus, the code above would result in an FTP session with the username `1001` and the password `temp456`.

In the `<application-desc>` element, we have already filled in the name of the application's main class; upon execution, its `main()` method is invoked. Next, let us look at the other child elements.

First, we have some some basic information about the application.

```
<information>
    <title>Clip Uploader</title>
    <vendor>Tim De Pauw</vendor>
    <homepage href="http://ovis.pwnt.be/"/>
    <description>Clip Uploader</description>
    <description kind="short">Uploads media clips</description>
</information>
```

Then come the application's security settings. Without this segment, the application would not have access to local files. Thus, it is also the main reason why we had to sign our JAR files in Section 12.4.

```
<security>
    <all-permissions/>
</security>
```

Finally, the document lists the resources that the application relies on.

```
<resources>
    <j2se version="1.5+" href="http://java.sun.com/products/autodl/j2se"/>
    <jar href="http://example.com/jar/be.pwnt.ovis.jar"/>
    <!-- … and more JAR URLs … -->
    <property name="be.pwnt.ovis.upload.hostname" value="upload.example.com"/>
    <!-- … and more system properties … -->
</resources>
```

The first resource is the Java platform itself, upon which a minimum version is imposed. Our application is compatible with Java 5 and up.

Then come the most important resources: the list of JAR files that comprise our application. In our case, there are quite a few; we only list one, as other entries are obviously similar.

The third and last type of resource is a system property, equivalent to those ordinarily passed to `java` using the `-D` switch. We use system properties to pass the FTP connection information, as well as the URL to the Web service. Again, we only list one of these.

13.8 Web Service

The `Ovis_Demo_View_WebService` class will expose our Web service, contacted by the clip upload application whenever an upload completes.

PHP comes with a *SOAP extension* [97], which allows us to implement a SOAP server. A common approach is to expose an object's methods via the server. Once we have our object, we assign it to a `SoapServer` and let that handle the request. Constructing the server also requires that we specify its URI.

```
$server = new SoapServer(null, array('uri' => $uri));
$server->setObject($object);
$server->handle();
```

Note that *all* of `$object`'s methods are exposed via the Web service. Therefore, we will use a *façade* class to ensure that only relevant methods become available.

As mentioned in Section 12.3.2.3, a WSDL document is required to automatically generate a Java Web service client. One would expect that WSDL generation could be automated in PHP, much like in Java EE or .NET. However, the available implementations leave a lot to be desired:

- The SOAP extension mentioned above is unable to generate WSDL altogether.
- Despite strong connections to the Apache project, *WSO2 WSF/PHP* [98] is severely limited in terms of parameter types.
- The more obscure *Webservice Helper* [99] handles non-native parameter types better, but the WSDL code it produces appears to be invalid.

Thus, the final version of the Web service simply contains a hard-coded WSDL document. The interested reader is kindly directed to the source code of the `Ovis_WebService` PHP class.

13.9 Software Installation

13.9.1 Dependencies

We discussed the installation of Apache on Debian in Section 9.2. However, before we can set up our Web application, we need to take care of additional dependencies introduced over the course of this chapter.

The `mod_rewrite` module is installed with Apache by default, but not enabled. To activate it, we simply need to create a symbolic link and restart the server.

```
# ln -s /etc/apache2/mods-available/rewrite.load /etc/apache2/mods-enabled/
# /etc/init.d/apache2 restart
```

The `HTML_QuickForm2` package is not available from Debian's repositories, but we can easily obtain it using the PEAR installer [100]. As this is the first time we encounter that installer in this document, let us first make sure that we have it on our system.

```
# apt-get install php-pear
```

Now, to actually install the `HTML_QuickForm2` package, we issue the command below. Note the suffix `-alpha`, which confirms that we want to install a bleeding edge package.

```
# pear install HTML_QuickForm2-alpha
```

As previously discussed, these instructions only apply to Debian-based systems. Other platforms may require the explicit installation of other software introduced in this chapter.

13.9.2 Application

Once the requirements have been met, we can install the Web application in three easy steps.

1. Place the source files inside the Web server's document root.[8]
2. Edit `/inc/settings.inc.php` to configure the platform.
3. Run `/util/install.php` either from the command line or in a Web browser.

The `install.php` script will simply set up the database tables. Similarly, `uninstall.php` will remove them. It is therefore not a good idea to expose these two scripts via the Web server. As a security measure, a `.htaccess` file in the `/util` directory blocks all Web access to that path.

13.10 Impressions

To conclude this chapter, let us again take a look at some screen captures of the fruit of our labor. A set of video clips that are freely available from the *Internet Archive* [101] are used as sample content, namely [102], [103], [104], and [105].

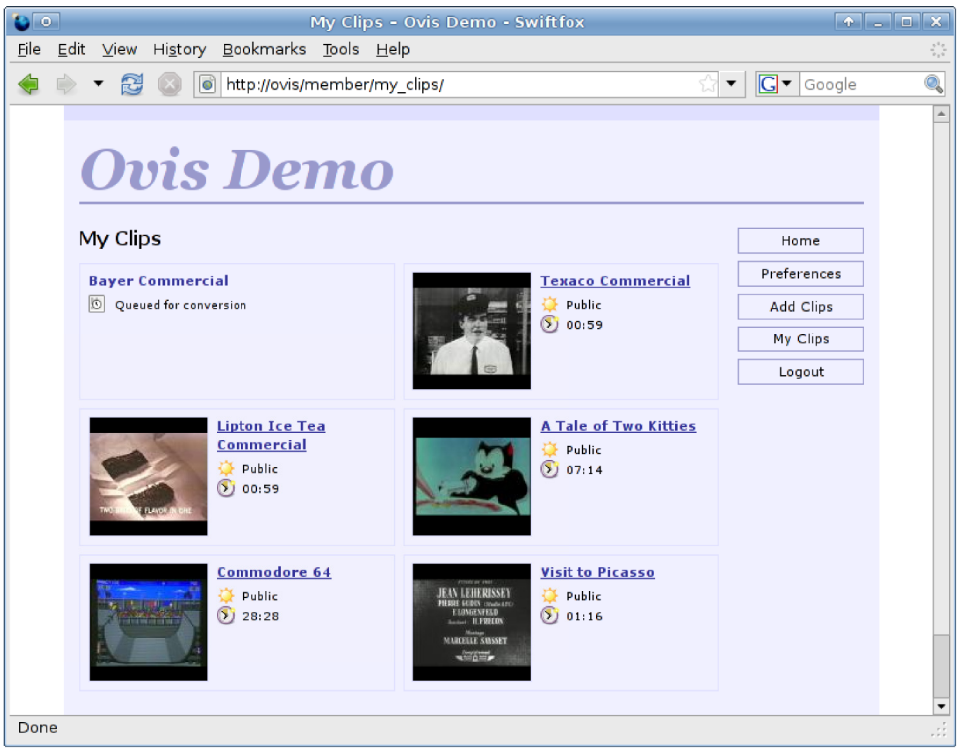

Figure 13.1: Personal clips at the demonstration portal

8 If the application is placed elsewhere, the `RewriteBase` directive discussed in Section 13.6 must reflect this.

Figure 13.2: Clip playback at the demonstration portal

14 DOKEOS INTEGRATION

We have arrived at phase two, in which we will be integrating the Ovis video streaming libraries with the PointCarré e-learning environment. This chapter should also be instructional to those wanting to incorporate Ovis into their own software, as it will list the actions required to do so.

14.1 Dokeos LCMS

As mentioned, PointCarré is based on Dokeos. This immediately presents an important choice: currently, there are two entirely separate development branches of Dokeos, which we will have to decide between. There is the original code, deployed in production environments, but there is also the more recent *Dokeos LCMS*, which stands for *Learning Content Management System*. [106]

Dokeos LCMS is an extensive rewrite of the Dokeos platform in clean object-oriented PHP 5. Whereas the original Dokeos is heavily course-centric, Dokeos LCMS revolves around *learning objects*, which are abstract representations of entities that may aid the learning process. Learning objects are standardized and occur in many e-learning environments these days. Applied to some common Dokeos features, virtually anything can become a learning object: an announcement, a shared file, a forum post, etc.

In the Dokeos LCMS implementation, a learning object has a *type* (e.g. 'shared file'), a number of *default properties* which are common for all types (e.g. the learning object's title or creation date) and a number of *type-specific properties* (e.g. a shared file's filename). Dokeos LCMS's modular design provides supporting code to manage learning objects and their types transparently. On the front-end, users have access to a new feature called 'My Repository', where they can organize all the learning objects that they have created.

Sadly, Dokeos LCMS is far from finished. Nonetheless, the development is making steady progress and the code is in a usable state. Moreover, yours truly has firsthand experience with developing its core classes.

Therefore, the streaming video platform will be based on the LCMS branch. At the time of this writing, the intention is to deploy a separate Dokeos LCMS installation alongside PointCarré in the near future, to run the streaming video platform and gradually migrate other features to it.

14.2 Required Steps

Over the past couple of chapters, we have outlined the steps required to set up an Ovis-based video sharing platform. For convenience, let us first summarize them. Note that their order has been slightly adapted.

1. **Hardware allocation.** Selecting the machines that our platform will run on.
2. **File system sharing.** Mounting the required volumes between machines.
3. **Database installation.** Setting up a MySQL server and database.
4. **FTP server configuration.** Setting up ProFTPD with MySQL-based authentication.
5. **Task scheduling.** Setting up cron jobs for `ovis_transcode` and `ovis_janitor`.
6. **Streaming server configuration.** Setting up Apache to serve video files and the Cortado applet.
7. **Application server configuration.** Setting up Apache to provide the Web front-end.

In what follows, we will examine to what extent the process will need to be adapted to function within a Dokeos LCMS environment. Even though references to previous chapters will be presented when relevant, it is assumed that the reader has familiarized himself with Ovis's architecture before tackling this chapter.

14.2.1 Hardware Allocation

In Chapter 6, we identified the different responsibilities to identify to our hardware. If we already have Dokeos LCMS running, there are two machines less to worry about. Dokeos and Ovis will share a database server (or servers), and the Dokeos Web server will also be Ovis's application server.

Technically, Dokeos LCMS supports a variety of database vendors. We will only consider MySQL, as it is Dokeos's default, and Ovis relies on ProFTPD integrating with it. However, most, if not all the information presented in this document is likely to apply to other vendors as well.

In what follows, it is assumed that each of the machines is running Debian lenny and has been incorporated into a local area network.

14.2.2 File System Sharing

Section 6.3.1 explained how to mount NFS volumes. The reader is hereby kindly referred to Figure 6.1 and Table 6.1, which may be of assistance in identifying the required mount operations.

14.2.3 Database Installation

As mentioned, we are relieved of the task of setting up a MySQL server. Additionally, the most convenient approach will be to let Ovis use Dokeos's database. To avoid conflicts, we will simply prefix our table names with `ovis_`, separating them from Dokeos's. Thus, when creating the data source definition file introduced in Section 6.3.2, we will merely need to specify the `<prefix>` option.

Now, let us define the tables which we will be adding to the Dokeos LCMS database. Figure 7.1 showed the design of the database for the demonstration platform. We will make two important changes to it:

- As Dokeos LCMS installations have their own user base, evidently, we will abandon our rudimentary user table in favor of the native one.
- Instead of storing clip information in a regular table, we will use what is perhaps the most potent feature of Dokeos LCMS: learning objects. If we define a learning object type for video clips, users will be able to publish clips in courses, use them to liven up their portfolio, etc. How to actually define such a learning object type will be discussed in Chapter 15. For now, suffice to say that we will not need the clip table.

Evidently, we will not burden system administrators with manually creating the tables. Also in Chapter 15, we will discuss where that should happen.

14.2.4 FTP Server Configuration

In Chapter 8, we discussed how to set up ProFTPD on a Debian system and subsequently configured it to retrieve authentication information from a MySQL database. When Ovis is integrated with Dokeos LCMS, this process will be identical, as we have just established that the `upload_accounts` table and `ftp_accounts` view will both remain intact.

14.2.5 Task Scheduling

Setting up cron jobs is a rather generic process, so the information in Section 10.5.2 still applies.

However, because of the changes to our database structure, some of the SQL queries in `ovis_transcode` and `ovis_janitor` will no longer apply. Because editing those scripts would make maintenance rather unpleasant, we will sneak an additional feature into them.

Specifically, `ovis_transcode` and `ovis_janitor`'s queries can be overridden by setting environment variables. The relevant environment variables all start with `OVIS_QUERY_`. For instance, `OVIS_QUERY_GET_PROFILES` may be defined to override the query for retrieving transcoding profiles. Both scripts come with a list of queries, which is displayed when the `--help` option is passed. We could of course use standard command line options to pass the queries as well. However, by using a totally different method, we somewhat emphasize the fact that overriding queries is an advanced feature of the scripts.

Now, to actually override the necessary queries for Dokeos LCMS, we first need some insight into its database structure. After all, the queries we need to override are related to the clip table we replaced earlier, meaning that queries on clips will become queries on the native learning object tables.

Luckily, the design is easy to understand. Earlier on, we mentioned that learning objects have two types of properties: default ones and type-specific ones. In the case of a video clip, an example of a default one would be its title, whereas its aspect ratio would be a type-specific one; we will get back to that in Chapter 15.

Anyway, Dokeos LCMS stores default properties in the generic `repository_learning_object` table, whereas type-specific ones get their own table, e.g. `repository_video_clip`. The two are linked via the `id` column.

14.2.5.1 Transcoding Application

The `ovis_transcode` script relies on two queries that use the clip table. Evidently, we are still dealing with Perl's DBI, so query parameters are again represented by question marks. Additionally, the reserved string `$prefix$` will be replaced with the table name prefix, defined in the data source definition file.

- The first query we need to adapt is `OVIS_QUERY_UPDATE_CLIP`, which updates a video clip's state, aspect ratio and duration. By default, the query is defined as:

```
UPDATE $prefix$clips
SET state = ?, aspect_ratio = ?, duration = ?
WHERE id = ?
```

  In our case, all three properties are in the type-specific table, as is the clip's ID. However, in Chapter 15, we will see that `state` is a reserved name in Dokeos LCMS; consequently, the column representing the clip's state is called `conversion_state` instead. Thus, we get:

```
UPDATE repository_video_clip
SET conversion_state = ?, aspect_ratio = ?, duration = ?
WHERE id = ?
```

- The second query, `OVIS_QUERY_GET_NEXT_JOB`, is a bit more complex. As its name suggests, it retrieves the next transcoding job. It is expected to return the clip's ID, the ID of its author, the name of the source file, and the start and end timestamp in milliseconds, if any. The default value is:

```
SELECT c.clip_id, author, source_file, start_time, end_time
FROM $prefix$clips AS c
JOIN $prefix$transcoding AS t
ON c.clip_id = t.clip_id
ORDER BY c.clip_id
LIMIT 1
```

  The author of the clip is actually the owner of the learning object, a default property. All the other data is still kept in the table of transcoding jobs. Consequently, the new query reads:

```
SELECT t.clip_id, o.owner AS author, t.source_file, t.start_time, t.end_time
FROM repository_learning_object AS o
JOIN $prefix$transcoding AS t
ON o.id = t.clip_id
ORDER BY o.created
LIMIT 1
```

14.2.5.2 Janitor

In the case of `ovis_janitor`, there is only one query we need to override, namely `OVIS_QUERY_GET_CLIP_IDS`. The script uses this query to retrieve all the clips' IDs from the database. Evidently, by default, it reads

```
SELECT clip_id FROM $prefix$clips
```

Thus, applied to Dokeos LCMS, we get the trivial query

```
SELECT id FROM repository_video_clip
```

14.2.6 Streaming Server Configuration

It should be no surprise that a streaming server for the Dokeos LCMS version of Ovis serves the exact same purpose as its generic counterpart. Thus, the installation process described in Section 9.1 still applies.

14.2.7 Application Server Configuration

In Section 9.2, we set up a Web server to run Ovis. As it turns out, the Ovis libraries only require software which is also required by Dokeos LCMS. As such, to turn a Dokeos LCMS Web server into an application server for Ovis, no additional actions are required.

Note that, in Section 13.9, we discussed the installation of additional dependencies on the application server. However, these applied only to the demonstration portal, not to the Ovis libraries.

15 DOKEOS DEVELOPMENT

Now that we have equipped our existing Dokeos LCMS infrastructure with supporting tools, it is time to write the glue code which will actually integrate Ovis with the code base.

15.1 Learning Object Types

In the previous chapter, we briefly discussed the purpose of learning objects. As mentioned, our goal will be to convert Ovis's clip table to a set of learning objects. In order to do this, we need to define a new learning object type and make Dokeos LCMS aware of it.

15.1.1 Default Properties

As shown in Figure 15.1, all learning objects share a common set of *default properties*, which are used to store general information about the object. In addition, every learning object has an *owner*, which is usually the user who created it. Using the *parent* property, learning objects can be arranged in tree structures.

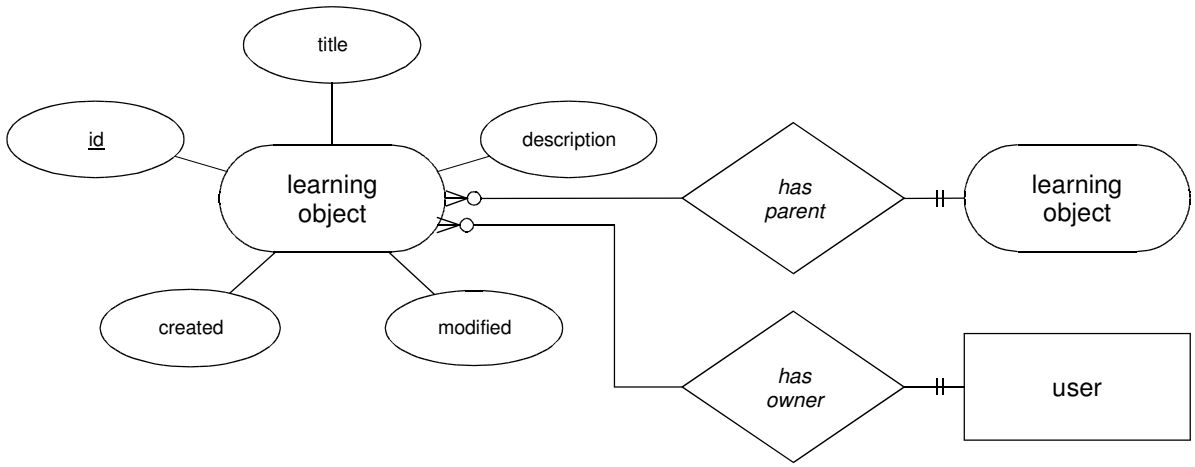

Figure 15.1: Generic entity-relationship diagram for learning objects

Two of the default learning object properties are not displayed in the figure, as they are mostly for use by Dokeos LCMS's internals. Those properties are:

- **State.** A numeric constant indicating the learning object's state. This property is used to indicate that a learning object has been moved to the recycle bin.
- **Display order index.** An integer which imposes an order on learning objects that share the same parent.

Note that in the figure, rounded rectangles distinguish learning object types from standard entity types. This convention will be maintained throughout this chapter.

15.1.2 Additional Properties

Evidently, for most of the learning object types used by Dokeos LCMS, the default set of properties will be insufficient. For this reason, *extended* learning object types may extend the default set with any number of properties, specific to the type. These are called *additional properties*. Roughly speaking, the data type of such a property must correspond to a standard SQL type.

For instance, the *link* learning object type builds upon the generic type *learning object* by adding the string-typed property *URL*, as displayed in Figure 15.2.

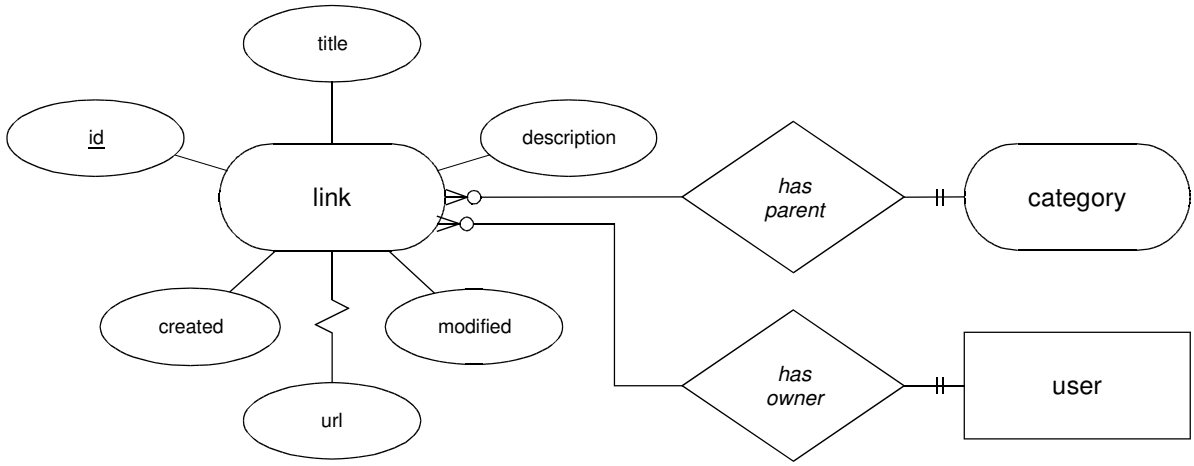

Figure 15.2: Entity-relationship diagram for the 'link' learning object type

In this figure, a resistor-like connector indicates that the property is specific to the learning object type; this is another convention which we will maintain. In addition, we introduce the *category* learning object type, which allows users to organize their learning objects; as such, it can be compared to a file folder.

15.2 Video Clip Learning Objects

From the database model in Section 7.2, we can see that many of the properties of a video clip directly map to the default properties listed above; note that a learning object's *state* is entirely different from a video clip's *conversion state*. The only properties that are not available are the clip's aspect ratio, duration and conversion state, which make those the *video clip* type's properties. The resulting type is illustrated in Figure 15.3.

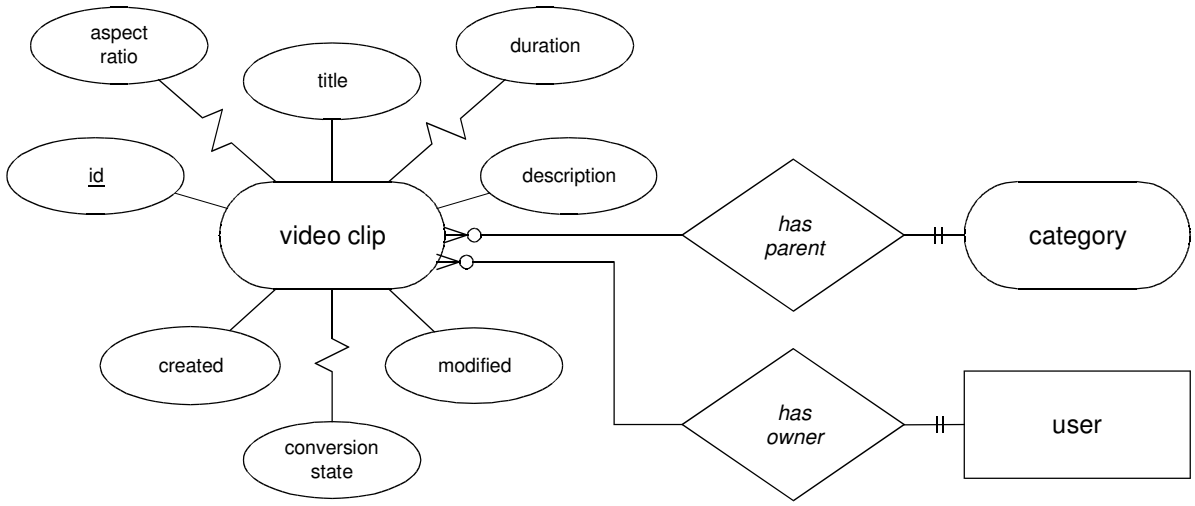

Figure 15.3: Entity-relationship diagram for the 'video clip' learning object type

Now, to add our new learning object type to Dokeos LCMS, we need to create a set of files, all contained in a new directory called `/repository/lib/learning_object/video_clip`. Upon initialization, a factory pattern will make sure that the files are automatically parsed, thus making the platform aware of the type.

15.2.1 Storage Unit

In the case of extended types, the first file we need is a file describing the *storage unit* required for holding on to type-specific properties. This is a simple XML file, which basically outlines the database table that needs to be created. Additionally, indexes may be defined; we will index the conversion state column to speed up the retrieval of public clips. Consequently, the file `video_clip.xml` will read:

```
<object name="video_clip">
    <properties>
        <property name="id" type="integer" unsigned="1" notnull="1"/>
        <property name="aspect_ratio" type="float" unsigned="1" notnull="0"/>
        <property name="duration" type="integer" unsigned="1" notnull="0"/>
        <property name="conversion_state" type="integer" unsigned="1" notnull="1"/>
    </properties>
    <index name="conversion_state">
        <indexproperty name="conversion_state"/>
    </index>
    <index name="id" type="primary">
        <indexproperty name="id"/>
    </index>
</object>
```

Incorporating the `id` column may seem like an odd choice, as it is mandatory. Indeed, the storage unit file format is likely to change in a future version of Dokeos LCMS.

15.2.2 Learning Object Class

The next thing we need to do is create the *learning object class* itself. The file is named `video_clip.class.php` and defines the class `VideoClip`. This class must extend the `LearningObject` base class, which, among other things, supplies it with methods for managing both types of properties.

Although not explicitly required, it is considered good practice to supply constants and accessor methods that correspond to the type-specific properties, if any. Thus, we get the following code:

```
class VideoClip extends LearningObject {
    const PROPERTY_ASPECT_RATIO = 'aspect_ratio';
    function get_aspect_ratio() {
        return $this->get_additional_property(self::PROPERTY_ASPECT_RATIO);
    }
    function set_aspect_ratio($aspect_ratio) {
        return $this->set_additional_property(self::PROPERTY_ASPECT_RATIO, $aspect_ratio);
    }
    // … plus the other two properties …
}
```

15.2.3 Display Class

When the Dokeos LCMS environment is instructed to generate an HTML representation of any learning object, it uses its *display class*. The PHP class `VideoClipDisplay` is defined in `video_clip_display.class.php`. It extends `LearningObjectDisplay` and offers methods for generating various representations of a `VideoClip` object; in our case, that means playing back the clip using the Cortado applet, as well as providing direct download links. We will not elaborate on this class's source code, as it is assumed to be straightforward.

15.2.4 Form Class

Similarly, the class `VideoClipForm`, defined in `video_clip_form.class.php`, extends `LearningObjectForm`. However, unlike the display class, the *form class* for a learning object type can get fairly complicated.

`LearningObjectForm` is based on `FormValidator`, which is Dokeos's wrapper class for `HTML_QuickForm`. Recall that we briefly discussed the latter in Chapter 13. As such, the general idea is to add QuickForm elements to the form object. Depending on the situation, either one of the object methods `build_creation_form()` and `build_editing_form()` is called; both add form elements for the default learning object properties. Thus, the form developer can override the methods to add elements for the type-specific properties.

Upon form submission, each of the elements' validation rules are checked. If all the elements validate, either `create_learning_object()` or `update_learning_object()` is called, again depending on the situation. Thus, the form class is responsible for creating or updating the learning object. Note that the developer of the class is not responsible for setting the learning object's default properties.

In the case of `VideoClipForm`, the creation-related methods will be slightly irregular. After all, when the user indicates that he wishes to create a video clip, that will merely start our clip upload application. Thus, we will hide the default form elements by *not* calling `parent::build_creation_form()`, and move the functionality of `create_learning_object()` to a supporting class.

Modification-related methods, on the other hand, do not even need to be overridden. If we leave them as they are, `LearningObjectForm` will provide the means to enter a title, description, etc. for the video clip. We obviously do not want users to modify its aspect ratio, for instance. This does mean that users will be unable to upload a new video file into an existing video clip learning object, which seems sensible.

We will again refrain from displaying the resulting class, as it would lead us too far. The interested reader is directed to the source code, which accompanies this document.

15.2.5 Version Control Classes

Something we have not discussed yet is that Dokeos LCMS's learning objects offer *version control*. To enable it, we would create the additional classes `VideoClipDifference` and `VideoClipDifferenceDisplay`. However, we will postpone video clip versioning until a future iteration. Consequently, the function `is_versionable()` in `VideoClip` returns `false`.

15.3 Ovis Integration

Now, we can *glue* the Ovis libraries into the Dokeos code base. All the required code will be placed under the directory `/common/ovis`. Let us go over the files located there.

15.3.1 Clip Uploader

Under `/common/ovis/jar`, we will place the JAR files required to run the clip upload application developed in Chapter 12. In order to establish a trust relationship, it is recommended that the archives be signed by the organization to which the e-learning environment belongs, as outlined in Section 12.4.

15.3.2 Data Source Definition

Under `/common/ovis/inc`, we will store files that are not directly accessible via the Web server. One of those is the data source definition file, `datasource.xml`, from Section 14.2.3.

15.3.3 Initialization Script

Any Dokeos script that relies on Ovis will merely need to incorporate `/common/ovis/inc/init.inc.php` to load all the necessary items. Specifically, `init.inc.php` in its turn loads `settings.inc.php`, which contains configuration values for Ovis. Next, it loads `ovis.inc.php`, which initializes the libraries and allows for the automatic loading of Ovis classes, as discussed in Section 13.1.

Note that the learning object classes we introduced in Section 15.2 also rely on the initialization script. For instance, `VideoClipDisplay` uses the static class `Ovis_Utilities` to generate HTML for the Cortado applet, and `VideoClipForm` uses the convenience method `Ovis_UploadAccount::get()` to obtain an upload account for the user, which is then passed to Java Web Start.

15.3.4 Data Manager

Upon initialization, the PHP constant `OVIS_DATA_MANAGER` is set to `"DokeosOvisDataManager"`. By doing so, the Ovis libraries are instructed to use the given class instead of the default `Ovis_Data_Manager`. The data manager is maintained using a *singleton pattern*: invoking `Ovis_Data_Manager::get_instance()` will return an instance of `DokeosOvisDataManager`.

The class `DokeosOvisDataManager`, defined in `/common/ovis/inc/dokeos_ovis_data_manager.class.php`, is a simple extent of `Ovis_Data_Manager`. It overrides two functions.

- While we defined the `VideoClip` learning object class earlier on, the Ovis libraries all rely on the internal `Ovis_Clip` class. Consequently, the data manager's `create_clip($ovis_clip)` function turns `$ovis_clip` into a learning object and saves it. For convenience, the code to convert an Ovis clip into a learning object is harbored in the `VideoClip` class, as the static method `import()`.
- Just like for the demonstration portal, the default data manager's `set_up()` function is extended to install two sensible transcoding profiles by default. The system administrator is of course free to alter these.

15.3.5 Utilities

To set up the default tables required by Ovis, as well as the default transcoding profiles we just mentioned, two scripts are provided, identical to those that come with the demonstration portal. In `/common/ovis/util`, the platform administrator will find `install.php` and `uninstall.php`.

Future versions of Dokeos LCMS should probably incorporate this functionality into the installer. For now, as Ovis requires quite a bit of extra configuration, manually invoking `install.php` is preferred.

15.3.6 Java Web Start

Ovis provides the functionality to generate the JNLP document from Section 13.7. Consequently, a user agent can request `/common/ovis/jnlp.php`, passing the FTP username and password as standard query string parameters. This will cause `Ovis_Utilities::print_jnlp()` to be invoked on the server side. On the client side, the document will then be passed to Java Web Start, as described in Section 13.7.

15.3.7 Web Service

Similarly, the Web service developed in Section 13.8 is located at the URL `/common/ovis/soap.php`. All this script needs to do is create an `Ovis_WebService` object and call its `run()` method. Everything else is handled internally by the Ovis libraries.

15.4 Video Clip Publication

All the learning objects created by a user are said to be in that user's *repository*. Consequently, by browsing to the *repository manager*, users can organize their learning objects, much like they would organize files on their personal computer. With the functionality we have added thus far, users will be able to add video files to their repository, change their title and description, and organize them for personal use.

Evidently, we also want to give users the opportunity to share video clips. More specifically, we want a course administrator to be able to publish a video clip in one or more of his courses.

15.4.1 WebLCMS

Dokeos LCMS comes with various *repository applications*. These are basically a set of modular classes, found under `/application`, which add functionality to the platform.

The repository application which mimics the functionality of the original Dokeos, namely a tool set to take part in a variety of course-centric activities, is called the *WebLCMS*, which is short for *Web-based Learning Content Management System*. Evidently, it comes with Dokeos LCMS by default.

15.4.2 Video Tool

The WebLCMS in its turn is comprised of *repository tools*. Each tool more or less corresponds to a learning object type. For instance, the forum tool allows a course administrator to link forums to his courses. Thus, our next step will be to develop the *Video Tool* for the WebLCMS application.

A WebLCMS tool is just a class which extends `RepositoryTool`. Like its siblings, the tool will be placed in the directory `/application/lib/weblcms/tool`. There, we create a subdirectory called `video`. When a user launches the tool, the WebLCMS will automatically load `video_tool.class.php` and instantiate `VideoTool`. Finally, as `VideoTool` is a `RepositoryTool`, its `run()` method will subsequently be invoked.

Thus, in the `run()` method, we output the HTML that the tool should generate. At the very least, this means calling the functions `display_header()` and `display_footer()`, inherited from `RepositoryTool`. Of course, we want our tool to do a bit more. It will have three main features, the first two of which are standard ones found in most of the existing repository tools.

- **Video clip publication.** If the user is a course administrator, he will be presented with the standard Dokeos LCMS interface for publishing learning objects. Adding this interface to our repository tool boils down to creating a `LearningObjectPublisher` for the `video_clip` type, and outputting the return value of its `as_html()` function. Everything else is handled internally.
- **Video clip browsing.** To view a list of learning objects of a certain type, published within a course, we first create an extent of `LearningObjectPublicationBrowser`. The class `VideoClipBrowser` in its turn relies on one of Dokeos LCMS's many `LearningObjectPublicationListRenderer` classes, which allow the developer to present a set of published learning objects in a variety of ways. For displaying a simple *list* of video clips,

we extend `ListLearningObjectPublicationListRenderer` to create `VideoClipPublicationListRenderer`. One of that renderer's tasks is to place a thumbnail image next to a video clip.

- **Video clip playback.** If the user clicks on the title of a video clip in the list, the clip will be played back. Evidently, this functionality uses the `VideoClipDisplay` class from Section 15.2. As such, the presentation of a video clip will use the same familiar interface throughout the entire Dokeos platform.

15.5 Cue Points

Back when we outlined our goals, in Section 5.8, we mentioned support for chapter marks. We also claimed that they would be easier to maintain if we refrained from adding the functionality to Ovis, and relied on the Dokeos core directly instead.

15.5.1 Cue Point Learning Objects

Indeed, much like we defined the video clip type in Section 15.2, we can define a new learning object type for chapter marks. Consequently, not only will database tables be maintained automatically, but, if we slightly modify the code, users will also be able to publish individual chapters in their courses.

We do need to take one important restriction into account. As mentioned, the Cortado applet allows us to seek to a certain time using JavaScript. However, there is no way to instruct the applet to stop playback at a given time in the clip. Consequently, rather than calling our learning object type a *chapter mark*, we will use the term *cue point*, which indicates its purpose more clearly.

15.5.2 Type Definition

The rather trivial design of the *video clip cue point* learning object type is displayed in Figure 15.4.

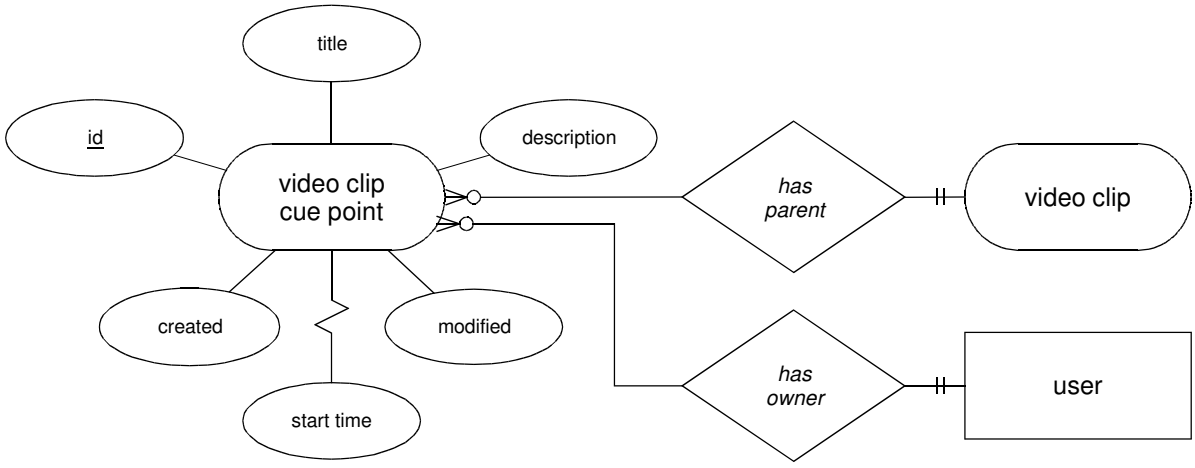

Figure 15.4: Entity-relationship diagram for the 'video clip cue point' learning object type

Section 15.2 showed how to define a new learning object type. Thus, we now repeat this process, albeit under a slightly different guise.

Firstly, we do not want users to be able to create instances of the type manually. To achieve this, we override the function `is_master_type()` from `LearningObject`, where it returns `true`. By letting the function return `false` in `VideoClipCuePoint`, we make Dokeos aware of the fact that the type is part of a more complex one.

Secondly, as Dokeos LCMS currently does not offer an API for modeling relations between learning object types, the introduction of the function `get_cue_points()` in `VideoClip` seems convenient. It requests a sorted set of cue points from the data manager, and returns them as an array.

Finally, the cue point type does not require a form class. Instead, the form belonging to a clip will allow users to modify its list of cue points. When the method `update_learning_object()` is called on the form object, it will update both its associated clip object and the clip's underlying cue point objects.

15.5.3 Form Design

As mentioned, users will be able to edit the list of cue points associated with a video clip from the form we introduced in Section 15.2.4. As such, the related functionality belongs in the `VideoClipForm` class.

Creating a friendly user interface for editing cue points is harder than it seems. We want to obtain two values per cue point: its title and its start time. The number of cue points is not known in advance, so the user must be given the opportunity to dynamically add and remove form elements. To enter the start time, we probably want to provide the user with three separate fields: one for hours, one for minutes and one for seconds; also, we should probably jump to the next field when a reasonable number has been filled in.

To achieve all this, the best thing we can do is develop a new QuickForm element class for entering times. Next, a second element class should represent a group of such time entry elements, each accompanied by a standard text field for the title, as well as the means to add such a pair to the group.

Although all this is absolutely feasible, due to the limited time frame of this project, we will take a different approach. Instead, we will add a standard HTML text area to the form, in which users can enter cue point specifications using the following format:

```
1:39 = Sheep takes its first bite of grass
2:01.081 Sheep chews the grass
1:09:22 = Sheep wags its tail
```

Note the use of different formatting varieties, which are all supported. In the second line, we see that the equals sign is optional, and that times may be specified to the millisecond. In the third line, we also add the number of hours—obviously, we are dealing with a fairly long clip here.

Upon form submission, the value of the text area is parsed, the start times are converted into a long integer representing the number of milliseconds, and the corresponding cue point objects are created. When building the form for an existing video clip, the text area is automatically filled with a human-readable representation of its list of cue points.

15.5.4 Playback

Our final step is to adapt the `VideoClipDisplay` class to display a list of segments, each of which call the JavaScript function `doSeek()` on the Cortado applet. Thus, we first call `get_cue_points()` on the clip object, after which we print a list of hyperlinks to a JavaScript statement.

If we assign the object for the Cortado applet an ID (e.g. `<object id="player">`), we can obtain it using `getElementById()`. Then, its `doSeek()` function takes a `double` between 0 and 1, which represents a relative position in the clip. Thus, in PHP, we `echo` something like:

```
'<a href="sorry_javascript_required.html" '
    . 'onclick="getElementById(\'player\').doSeek('
    . ($cue_point->get_start_time() / $video_clip->get_duration())
    . '); return false;">'
    . htmlspecialchars($cue_point->get_title())
    . '</a>'
```

This code snippet has two intricacies that are often overlooked. Firstly, many developers are inclined to write

```
<a href="javascript:getElementById(id).doSeek(pos);">
```

While this will work as long as JavaScript is enabled in the Web browser's preferences, it is not considered sensible. By using a regular URL in the `href` attribute, a fallback action can be specified, provided that the `onclick` handler return `false`. If `return false` were omitted, the JavaScript code would still be executed, but the Web browser would subsequently navigate to the URL as well.

Secondly, although tangentially related at best, calling `htmlspecialchars()` is necessary to encode reserved characters as their equivalent HTML entity. Specifically, replacements are made as listed in Table 15.1.

Character	HTML entity
&	&amp;
"	&quot;
<	&lt;
>	&gt;

Table 15.1: Reserved characters in HTML and their corresponding entity codes

If a developer fails to encode HTML output properly, the implications can be dramatic, since it allows users to enter arbitrary code. Not only will this almost always result in a malformed HTML document, but it also allows malicious users to spread JavaScript exploits and the like.

It goes without saying that `htmlspecialchars()` is called numerous times throughout the code base. To avoid oversights in this department, future versions of Dokeos could perhaps rely on the popular template engine *Smarty* [107] or a more advanced MVC solution.

15.5.5 Mozilla Browser Compatibility

Due to the bug described in [108] and [109], recent versions of the Java plug-in for Mozilla browsers do not allow socket connections that are the result of a JavaScript call. This behavior started with Java 6 Update 3. Users affected by it will receive the error message "Not allowed" when they attempt to jump to a cue point.

Fortunately, the issue will be fixed in Java 6 Update 10. At the time of this writing, that version is in beta testing. After installing beta build 23 from [110], we can indeed jump to cue points using Mozilla Firefox 3 Release Candidate 1, on both Windows XP Service Pack 3 and Ubuntu Linux 8.04. As all of the Mozilla Foundation's Web browsers share a common plug-in architecture, we should be in the clear.

Because we can expect Java 6 Update 10 to be released shortly, we will refrain from attempting to find a workaround for users of previous versions. Instead, we will recommend that they update their software. Those who are reluctant to do so may temporarily turn to a different Web browser, or decide not to use the cue point functionality altogether.

15.6 Localization

To localize the Dokeos LCMS classes we just discussed, an API is available. However, it is set to undergo a thorough revision. Therefore, we will not discuss it in great detail.

As it stands, the `/languages` directory contains a directory for every available language. In these directories are PHP scripts such as `repository.inc.php` and `weblcms.inc.php`; the former contains strings related to the learning object repository, the latter is specific to the WebLCMS application. By extending these files with key/value pairs—using a peculiar syntax we will not describe here—, new locale strings can be added.

To obtain a string from the file which is currently loaded, we use `Translation::get($key)`. Since the API currently cannot handle more than one file simultaneously, most of the strings related to our new classes are duplicated in both of the files mentioned above.

15.7 Impressions

At long last, we have implemented the ability to share video clips on the PointCarré platform. Let us take a moment to observe the result.

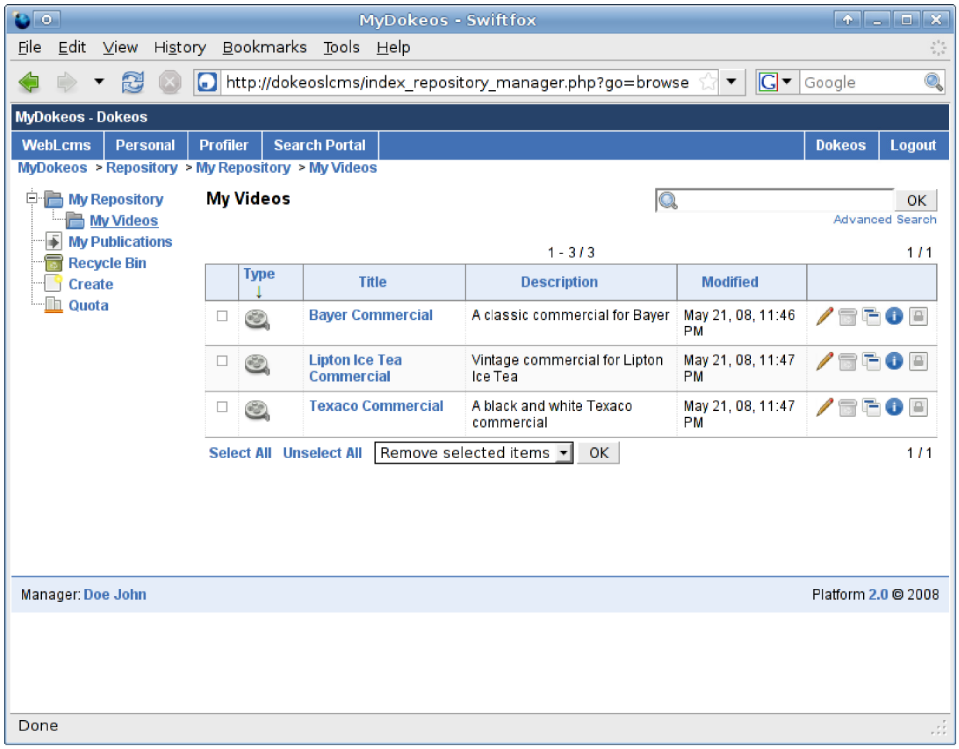

Figure 15.5: List of video clips in the user's repository

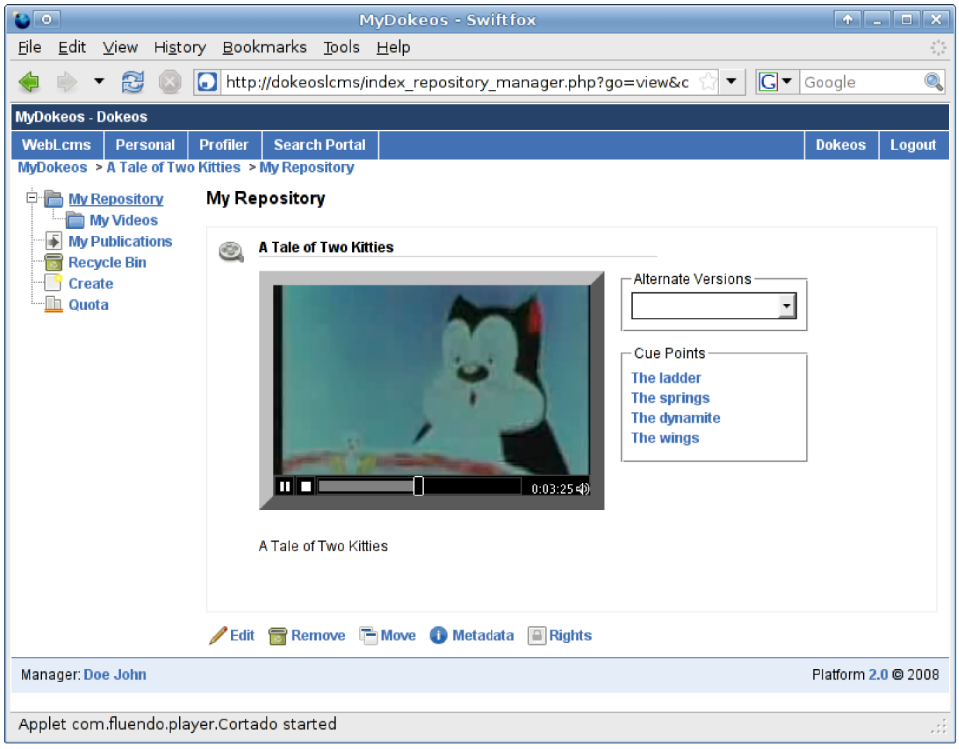

Figure 15.6: Playback of a video clip in the user's repository

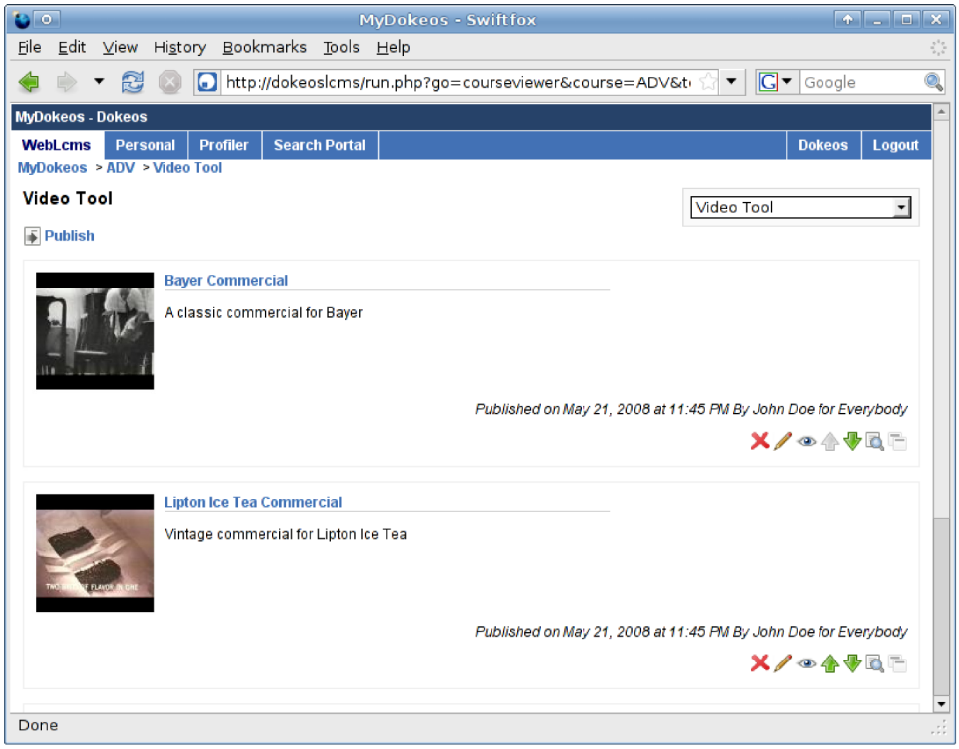

Figure 15.7: List of video clips published in the current course

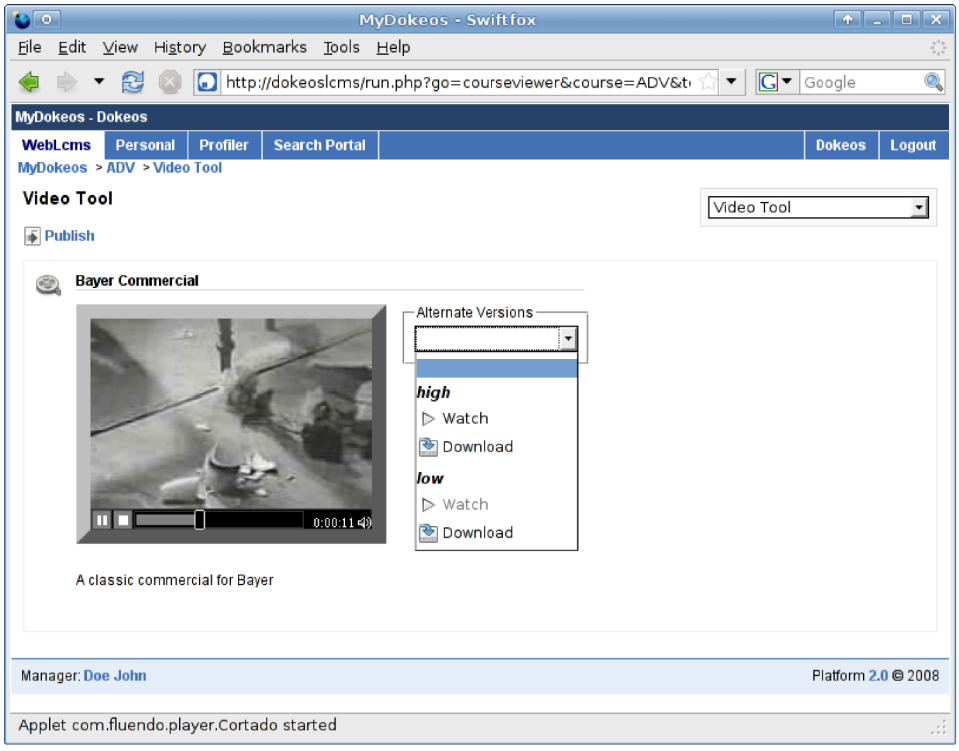

Figure 15.8: Playback of a video clip from the Video Tool

16 CONCLUSION

So, did this project attain its goals? Let us make a summary of what we have accomplished over the course of this research and compare that to what we initially set out to achieve.

First, we learned how to set up a collection of machines to support a streaming video platform. It consists of a Web server that runs the front-end, an FTP server for uploading video material, a machine in charge of transcoding video material, a video streaming server, a database server, and two file servers that store the files involved in the process. We reserved the possibility to assign several of these roles to one physical machine.

Next, we supplied the software for all the machines involved. Building upon free and open source software, we added the required capabilities to our infrastructure. We discussed the development of various software components, which together form our video sharing solution.

The first of these components is a Perl script that, relying on existing video processing software, sequentially transcodes a queue of user-provided video material. We implemented the possibility to define various quality settings and to create thumbnail images. As the script interacts with the database server, authors are kept up to date on their material's status. Additionally, this information can be supplied via e-mail.

In order to allow users to upload video clips to the platform, we developed an easy-to-use Java FTP client. The application allows users to specify information about the clips, which is posted to the platform using a Web service. By keeping the logic related to video clips separate from the rest of the application, a trivial FTP upload application was created as well.

Then, a PHP class library was implemented, which can be used to support a Web application for streaming video. As a proof of concept, such an application was developed, allowing users to publish video clips and play back those that others have supplied. Using Java Web Start technology, the application for uploading video clips is started directly from the Web browser.

Finally, we proceeded to integrate the newly created set of PHP classes with the Dokeos LCMS e-learning environment. By defining learning object types for video clips as well as cue points, we successfully extended Dokeos LCMS with the ability to publish video clips in courses, learning paths, portfolios, andsoforth.

Thus, our two main goals have been reached. Firstly, we have eliminated intervention by OSC personnel: teachers wanting to share video material with their students will be able to do so autonomously. Secondly, our solution has been fully integrated with the PointCarré code base.

Moreover, the solution we have developed can fairly easily be deployed outside the Vrije Universiteit Brussel infrastructure. Not only can institutions that rely on Dokeos make use of it, but we even provide a fully generic tool set, which allows for integration with any PHP application. Additionally, those interested in an autonomous solution can turn to our proof of concept and use it as a starting point for their own application.

Can we justify favoring our own solution over existing free-of-charge video sharing Web sites? By harnessing Dokeos's rich feature set, we introduced fine-grained access control, as well as a clutter-free interface, which prevents any distraction from the learning process. In addition, not only is our software ad-free, but it is even free and open source. Thus, it is safe to say that our effort has paid off.

Even though we have fulfilled our objectives, it should be noted that there is room for improvement. For one, support for cue points is specific to the Dokeos version of our software. Also, while fully functional, the FTP upload application is somewhat limited in the field of error handling. Finally, not every part of the platform will scale as well as the next; if one wishes to assign a server responsibility to multiple machines, it remains to be seen to what extent the software will need to be adapted.

Nonetheless, we can look back upon a fruitful development cycle. Using the components discussed in this document, a highly usable video sharing solution can be set up, both inside and outside Vrije Universiteit Brussel grounds, relying on nothing but free and open source software. Happy watching!

REFERENCES

[1] *Wikipedia, The Free Encyclopedia*. (2004, July 22). FL: Wikimedia Foundation, Inc. Retrieved April 17, 2008, from http://www.wikipedia.org/

[2] Container format (digital). (2008, March 17). In *Wikipedia, The Free Encyclopedia*. Retrieved April 17, 2008, from http://en.wikipedia.org/w/index.php?title=Container_format_%28digital%29&oldid=198767092

[3] Codec. (2008, April 13). In *Wikipedia, The Free Encyclopedia*. Retrieved April 17, 2008, from http://en.wikipedia.org/w/index.php?title=Codec&oldid=205303484

[4] Comparison of audio codecs. (2008, April 15). In *Wikipedia, The Free Encyclopedia*. Retrieved April 17, 2008, from http://en.wikipedia.org/w/index.php?title=Comparison_of_audio_codecs&oldid=205870063

[5] Psychoacoustics. (2008, April 18). In *Wikipedia, The Free Encyclopedia*. Retrieved April 19, 2008, from http://en.wikipedia.org/w/index.php?title=Psychoacoustics&oldid=206429170

[6] Windows Media Audio. (2008, May 16). In *Wikipedia, The Free Encyclopedia*. Retrieved May 21, 2008, from http://en.wikipedia.org/w/index.php?title=Windows_Media_Audio&oldid=212847032

[7] Vorbis. (2008, May 20). In *Wikipedia, The Free Encyclopedia*. Retrieved May 21, 2008, from http://en.wikipedia.org/w/index.php?title=Vorbis&oldid=213691782

[8] Comparison of video codecs. (2008, April 6). In *Wikipedia, The Free Encyclopedia*. Retrieved April 17, 2008, from http://en.wikipedia.org/w/index.php?title=Comparison_of_video_codecs&oldid=203825238

[9] Windows Media Video. (2008, May 13). In *Wikipedia, The Free Encyclopedia*. Retrieved May 21, 2008, from http://en.wikipedia.org/w/index.php?title=Windows_Media_Video&oldid=212047806

[10] VP7. (2008, March 31). In *Wikipedia, The Free Encyclopedia*. Retrieved May 21, 2008, from http://en.wikipedia.org/w/index.php?title=VP7&oldid=202377671

[11] Comparison of container formats. (2008, April 10). In *Wikipedia, The Free Encyclopedia*. Retrieved April 17, 2008, from http://en.wikipedia.org/w/index.php?title=Comparison_of_container_formats&oldid=204724025

[12] Voice over Internet Protocol. (2008, April 29). In *Wikipedia, The Free Encyclopedia*. Retrieved April 29, 2008, from http://en.wikipedia.org/w/index.php?title=Voice_over_Internet_Protocol&oldid=209027698

[13] Real-time Transport Protocol. (2008, April 11). In *Wikipedia, The Free Encyclopedia*. Retrieved April 17, 2008, from http://en.wikipedia.org/w/index.php?title=Real-time_Transport_Protocol&oldid=205003080

[14] Real time control protocol. (2008, January 28). In *Wikipedia, The Free Encyclopedia*. Retrieved April 17, 2008, from http://en.wikipedia.org/w/index.php?title=Real_time_control_protocol&oldid=187583586

[15] Real Time Streaming Protocol. (2008, April 1). In *Wikipedia, The Free Encyclopedia*. Retrieved April 17, 2008, from http://en.wikipedia.org/w/index.php?title=Real_Time_Streaming_Protocol&oldid=202494752

[16] Schulzrinne, H., & Rao, A. (1998, April). *RFC 2326 Real Time Streaming Protocol (RTSP)*. Retrieved April 17, 2008, from http://tools.ietf.org/html/rfc2326

[17] *MPlayer – The Movie Player*. (2008, April 15). Retrieved April 17, 2008, from http://www.mplayerhq.hu/design7/news.html

[18] *FFmpeg*. (2008, April 2). Retrieved April 17, 2008, from http://ffmpeg.mplayerhq.hu/

[19] *Transcode Wiki*. (2008, January 27). Retrieved April 17, 2008, from http://www.transcoding.org/cgi-bin/transcode

[20] Gerber, J. (2008, February 19). *ffmpeg2theora*. Retrieved April 17, 2008, from http://www.v2v.cc/~j/ffmpeg2theora/

[21] The Apache Software Foundation. (2008, March 9). *The Apache HTTP Server Project*. Retrieved April 17, 2008, from http://httpd.apache.org/

[22] *lighttpd fly light*. (2008, April 16). Retrieved April 17, 2008, from http://www.lighttpd.net/

[23] *Icecast.org*. (n.d.). Retrieved April 17, 2008, from http://www.icecast.org/

[24] *GStreamer: open source multimedia framework*. (2008, April 3). Retrieved April 17, 2008, from http://gstreamer.freedesktop.org/

[25] Fluendo. (2008, February 5). *Flumotion*. Retrieved April 17, 2008, from http://www.flumotion.net/

[26] The VideoLAN team. (2008, April 6). *VLC media player*. Retrieved April 17, 2008, from http://www.videolan.org/vlc/

[27] Apple Inc. (n.d.). Streaming Server. In *Apple Developer Connection (Open Source)*. Retrieved April 17, 2008, from http://developer.apple.com/opensource/server/streaming/

[28] Apple Inc. (n.d.). Streaming Server. In *QuickTime*. Retrieved April 17, 2008, from http://www.apple.com/quicktime/streamingserver/

[29] Wijering, J. (2008, April 4). *JW FLV Media Player*. Retrieved April 22, 2008, from http://www.jeroenwijering.com/?item=JW_FLV_Player

[30] Sun Microsystems, Inc. (n.d.). *Java Media Framework API (JMF)*. Retrieved April 17, 2008, from http://java.sun.com/products/java-media/jmf/

[31] Fluendo. (2008, February 5). *Streaming applet for Ogg formats*. Retrieved April 17, 2008, from http://www.flumotion.net/cortado/

[32] Wikimedia Foundation, Inc. (2008, March 23). *Wikimedia Foundation*. Retrieved April 17, 2008, from http://www.wikimedia.org/

[33] Hickson, I., & Hyatt, D. (2008, April 15). *HTML 5*. Retrieved April 17, 2008, from http://www.w3.org/html/wg/html5/

[34] HTML 5. (2008, April 11). In *Wikipedia, The Free Encyclopedia*. Retrieved April 17, 2008, from http://en.wikipedia.org/w/index.php?title=HTML_5&oldid=204986284

[35] DirectX. (2008, April 16). In *Wikipedia, The Free Encyclopedia*. Retrieved April 17, 2008, from http://en.wikipedia.org/w/index.php?title=DirectX&oldid=206037322

[36] *ffdshow tryouts*. (n.d.). Retrieved April 17, 2008, from http://ffdshow-tryout.sourceforge.net/

[37] Google. (n.d.). *Google Video*. Retrieved April 17, 2008, from http://video.google.com/

[38] Sun Microsystems, Inc. (n.d.). *Java Web Start Technology*. Retrieved April 17, 2008, from http://java.sun.com/products/javawebstart/

[39] The ProFTPD Project. (n.d.). *The ProFTPD Project*. Retrieved April 17, 2008, from http://www.proftpd.org/

[40] Fluendo. (2006, November). /cortado/trunk/README. In *Flumotion*. Retrieved April 17, 2008, from https://code.fluendo.com/flumotion/trac/browser/cortado/trunk/README

[41] SPI. (2008, April 21). *Debian – The Universal Operating System*. Retrieved April 22, 2008, from http://www.debian.org/

[42] Smith, C. (2006, May 2). *Linux NFS-HOWTO*. Retrieved April 19, 2008, from http://nfs.sourceforge.net/nfs-howto/

[43] MySQL AB. (n.d.). GRANT Syntax. In *MySQL 5.0 Reference Manual (SQL Statement Syntax)*. Retrieved April 22, 2008, from http://dev.mysql.com/doc/refman/5.0/en/grant.html

[44] phpMyAdmin Devel Team. (2007, August 21). *phpMyAdmin*. Retrieved April 22, 2008, from http://www.phpmyadmin.net/

[45] The ProFTPD Project. (n.d.). *The ProFTPD Project: Project Documentation*. Retrieved April 17, 2008, from http://www.proftpd.org/docs/

[46] Apache Documentation Group. (2008, February 21). *Apache HTTP Server Documentation*. Retrieved April 22, 2008, from http://httpd.apache.org/docs/

[47] Vromans, J. (2008, February 22). Getopt::Long. In *perldoc.perl.org (Core modules)*. Retrieved April 22, 2008, from http://perldoc.perl.org/Getopt/Long.html

[48] Appleton, B. (2008, February 22). Pod::Usage. In *perldoc.perl.org (Core modules)*. Retrieved April 22, 2008, from http://perldoc.perl.org/Pod/Usage.html

[49] Wall, L., & Burke, S.M. (2008, February 22). perlpod. In *perldoc.perl.org (Language reference)*. Retrieved April 29, 2008, from http://perldoc.perl.org/perlpod.html

[50] Allen, J. (2008, February 22). Fcntl. In *perldoc.perl.org (Core modules)*. Retrieved April 17, 2008, from http://perldoc.perl.org/Fcntl.html

[51] McLean, G. (2006, July 20). *XML::Simple*. Retrieved April 17, 2008, from http://www.mclean.net.nz/cpan/

[52] Sarathy, G. (2003, August 24). Data::Dumper. In *perldoc.perl.org (Core modules)*. Retrieved April 29, 2008, from http://perldoc.perl.org/Data/Dumper.html

[53] Sterin, I. (n.d.). *Perl DBI*. Retrieved April 17, 2008, from http://dbi.perl.org/

[54] Ruzmetov, S. (n.d.). Image::Resize – Simple image resizer using GD. In *CPAN*. Retrieved April 22, 2008, from http://search.cpan.org/~sherzodr/Image-Resize/Resize.pm

[55] perlmod. (2008, March 21). In *perldoc.perl.org (Language reference)*. Retrieved May 10, 2008, from http://perldoc.perl.org/perlmod.html

[56] Garrels, M. (2007, September 20). Scheduling processes. In *Introduction to Linux*. Retrieved April 22, 2008, from http://tldp.org/LDP/intro-linux/html/sect_04_04.html

[57] Eryq, & Orton, Y. (n.d.). MIME::Lite – low-calorie MIME generator. In *CPAN*. Retrieved May 16, 2008, from http://search.cpan.org/~rjbs/MIME-Lite/lib/MIME/Lite.pm

[58] Postfix. (2008, March 17). In *Debian Wiki*. Retrieved May 16, 2008, from http://wiki.debian.org/Postfix

[59] Hansmann, D. (2007, December 19). *JFtp – Open Source FTP, SFTP, NFS and SMB file transfer client and more*. Retrieved April 19, 2008, from http://j-ftp.sourceforge.net/

[60] Petrovicova, B. (2005, April 24). *Java FTP Client Library*. Retrieved April 19, 2008, from http://jvftp.sourceforge.net/

[61] Zhao, D. (n.d.). ZUpload Java Upload Applet. In *SourceForge.net (Projects)*. Retrieved April 19, 2008, from http://sourceforge.net/projects/zupload/

[62] Norguet, J. (2006, March 6). Update: Java FTP libraries benchmarked. In *Java World*. Retrieved April 19, 2008, from http://www.javaworld.com/javaworld/jw-03-2006/jw-0306-ftp.html

[63] The Apache Software Foundation. (2008, March 5). Net. In *Apache Commons*. Retrieved April 19, 2008, from http://commons.apache.org/net/

[64] Sun Microsystems, Inc. (2008, March 18). Trail: Creating a GUI with JFC/Swing. In *The Java™ Tutorials*. Retrieved April 22, 2008, from http://java.sun.com/docs/books/tutorial/uiswing/

[65] *Tango Desktop Project*. (2007, October 17). Retrieved April 22, 2008, from http://tango.freedesktop.org/Tango_Desktop_Project

[66] Sun Microsystems, Inc. (2008, March 18). Lesson: Concurrency. In *The Java™ Tutorials (Essential Classes)*. Retrieved April 22, 2008, from http://java.sun.com/docs/books/tutorial/essential/concurrency/

[67] The Apache Software Foundation. (n.d.). ftp. In *Commons Net 1.5.0-SNAPSHOT Reference (org.apache.commons.net.ftp)*. Retrieved May 18, 2008, from http://commons.apache.org/net/xref/examples/ftp.html

[68] Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001, March 15). *Web Service Definition Language (WSDL)*. Retrieved April 29, 2008, from http://www.w3.org/TR/wsdl

[69] The Eclipse Foundation. (n.d.). Web Tools Platform (WTP) Project. In *Eclipse.org*. Retrieved April 22, 2008, from http://www.eclipse.org/webtools/

[70] Smart, J.F. (2006, September 26). Eclipse Callisto Project Profile: Web Tools Platform. In *DevX*. Retrieved April 22, 2008, from http://www.devx.com/webdev/Article/32534

[71] The Apache Software Foundation. (2007, October 11). Apache Axis2. In *Apache Web Services Project*. Retrieved April 22, 2008, from http://ws.apache.org/axis2/

[72] Sun Microsystems, Inc. (2008, March 18). *Lesson: Java Web Start*. Retrieved April 17, 2008, from http://java.sun.com/docs/books/tutorial/deployment/webstart/

[73] Sun Microsystems, Inc. (2008, March 18). Signing and Verifying JAR Files. In *The Java™ Tutorials*. Retrieved April 19, 2008, from http://java.sun.com/docs/books/tutorial/deployment/jar/signindex.html

[74] Sun Microsystems, Inc. (2007, December 18). keytool. In *JDK 6 Documentation*. Retrieved April 22, 2008, from http://java.sun.com/javase/6/docs/technotes/tools/solaris/keytool.html

[75] Rethans, D. (2005, November 22). *Minutes PHP Developers Meeting, name spaces*. Retrieved April 22, 2008, from http://www.php.net/~derick/meeting-notes.html#name-spaces

[76] Good, N.A. (2008, May 6). The future of PHP. In *developerWorks (Open source)*. Retrieved May 12, 2008, from http://www.ibm.com/developerworks/opensource/library/os-php-future/

[77] The PHP Group. (2008, April 18). Autoloading Objects. In *PHP Manual (Classes and Objects (PHP 5))*. Retrieved April 22, 2008, from http://www.php.net/manual/en/language.oop5.autoload.php

[78] The PHP Group. (2008, April 18). set_include_path. In *PHP Manual (PHP Options/Info Functions)*. Retrieved April 22, 2008, from http://www.php.net/manual/en/function.set-include-path.php

[79] The PHP Group. (2008, May 9). SimpleXML. In *PHP Manual (XML Manipulation)*. Retrieved May 12, 2008, from http://www.php.net/manual/en/book.simplexml.php

[80] The PHP Group. (2008, April 18). PDO. In *PHP Manual (Abstraction Layers)*. Retrieved April 22, 2008, from http://www.php.net/manual/en/book.pdo.php

[81] Alberton, L., Convissor, D., Coallier, D., Fazelzadeh, A., Smith, L., & Cooper, P. (n.d.). MDB2. In *PEAR (Packages)*. Retrieved April 22, 2008, from http://pear.php.net/package/MDB2

[82] Lim, J. (2006, March 13). *ADOdb Database Abstraction Library for PHP (and Python)*. Retrieved April 22, 2008, from http://adodb.sourceforge.net/

[83] The PHP Group. (2008, April 18). Exceptions. In *PHP Manual (Language Reference)*. Retrieved April 22, 2008, from http://www.php.net/manual/en/language.exceptions.php

[84] The PHP Group. (2008, April 25). set_exception_handler. In *PHP Manual (Error Handling and Logging Functions)*. Retrieved April 29, 2008, from http://www.php.net/manual/en/function.set-exception-handler.php

[85] The PHP Group. (2008, May 9). Predefined Constants. In *PHP Manual (PHP Data Objects)*. Retrieved May 15, 2008, from http://us.php.net/manual/en/pdo.constants.php

[86] The PHP Group. (2008, May 9). PDOStatement->fetch. In *PHP Manual (The PDOStatement class)*. Retrieved May 15, 2008, from http://us.php.net/manual/en/pdostatement.fetch.php

[87] The PEAR Documentation Group. (2008, April 20). Quoting and escaping. In *PEAR Manual (MDB2)*. Retrieved April 22, 2008, from http://pear.php.net/manual/en/package.database.mdb2.intro-quote.php#package.database.mdb2.intro-quote.quote-identifiers

[88] Mansion, B., Borzov, A., Daniel, A., Rust, J., Schulz, T., & McClain, R. (n.d.). HTML_QuickForm. In *PEAR (Packages)*. Retrieved April 22, 2008, from http://pear.php.net/package/HTML_QuickForm

[89] *QuickForm2*. (2007, July 5). Retrieved April 22, 2008, from http://quickform.mamasam.com/wiki/

[90] API. (2007, May 15). In *QuickForm2*. Retrieved April 22, 2008, from http://quickform.mamasam.com/wiki/api

[91] File Source for basic-elements.php. (2007, June 30). In *PEAR (HTML_QuickForm2)*. Retrieved April 22, 2008, from http://pear.php.net/package/HTML_QuickForm2/docs/latest/__filesource/fsource_HTML_QuickForm2__HTML_QuickForm2-0.2.0docsexamplesbasic-elements.php.html

[92] File Source for builtin-rules.php. (2007, October 15). In *PEAR (HTML_QuickForm2)*. Retrieved April 22, 2008, from http://pear.php.net/package/HTML_QuickForm2/docs/latest/__filesource/fsource_HTML_QuickForm2__HTML_QuickForm2-0.2.0docsexamplesbuiltin-rules.php.html

[93] Model-view-controller, PHP. (2008, April 16). In *Wikipedia, The Free Encyclopedia*. Retrieved April 17, 2008, from http://en.wikipedia.org/w/index.php?title=Model-view-controller&oldid=206086178#PHP

[94] The Apache Software Foundation. (2008, March 12). Apache Module mod_rewrite. In *Apache HTTP Server Version 2.2 Documentation (Modules)*. Retrieved April 22, 2008, from http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

[95] Apache Documentation Group. (2008, February 8). .htaccess. In *Apache HTTP Server Version 2.2 Documentation (How-To / Tutorials)*. Retrieved April 22, 2008, from http://httpd.apache.org/docs/2.2/howto/htaccess.html

[96] Apache Documentation Group. (2008, March 12). Apache mod_rewrite. In *Apache HTTP Server Version 2.2 Documentation (Modules)*. Retrieved April 22, 2008, from http://httpd.apache.org/docs/2.2/rewrite/

[97] The PHP Group. (2008, April 18). SOAP. In *PHP Manual (Web Services)*. Retrieved April 22, 2008, from http://www.php.net/manual/en/ref.soap.php

[98] WSO2 Inc. (n.d.). WSO2 Web Services Framework for PHP. In *WSO2 Oxygen Tank*. Retrieved April 22, 2008, from http://wso2.org/projects/wsf/php

[99] Kingma, D. (n.d.). *Webservice Helper*. Retrieved April 22, 2008, from http://www.jool.nl/new/1,webservice_helper.html

[100] The PEAR Documentation Group. (2008, April 20). Command line installer. In *PEAR Manual (User Guide)*. Retrieved April 22, 2008, from http://pear.php.net/manual/en/guide.users.commandline.cli.php

[101] Moving Image Archive. (2008, April 24). In *Internet Archive*. Retrieved April 24, 2008, from http://www.archive.org/details/movies

[102] Commodore 64. (1988). In *Internet Archive (Computer Chronicles)*. Retrieved April 24, 2008, from http://www.archive.org/details/CC517_commodore_64

[103] Haesaerts, P. (1950). Visite à Picasso. In *Internet Archive (Academic Film Archive of North America)*. Retrieved April 24, 2008, from http://www.archive.org/details/afana_visit_to_picasso

[104] Clampett, R., & Foster, W. (1942, November 21). Merrie Melodies: A Tale of Two Kitties. In *Internet Archive (Animation Shorts)*. Retrieved April 24, 2008, from http://www.archive.org/details/TaleofTwoKitties

[105] Classic Television Commercials. (n.d.). In *Internet Archive (Classic TV)*. Retrieved April 24, 2008, from http://www.archive.org/details/ctvc

[106] LCMS. (2008, January 28). In *Dokeos*. Retrieved April 17, 2008, from http://www.dokeos.com/wiki/index.php/LCMS

[107] New Digital Group, Inc. (2008, February 12). *Smarty: Template Engine*. Retrieved May 25, 2008, from http://www.smarty.net/

[108] Sun Microsystems, Inc. (2007, October 26). Bug 6622150. In *Sun Developer Network (Bug Database)*. Retrieved May 18, 2008, from http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6622150

[109] Sun Microsystems, Inc. (2008, February 29). Bug 6669818. In *Sun Developer Network (Bug Database)*. Retrieved May 18, 2008, from http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6669818

[110] Sun Microsystems, Inc. (n.d.). Java SE 6 Update 10 Beta Overview. In *Sun Developer Network (Java SE Early Access Downloads)*. Retrieved May 18, 2008, from http://java.sun.com/javase/downloads/ea/6u10/6u10beta.jsp