UiO : Department of Informatics
University of Oslo

MovieCutter

A system to make personalized video summaries from archived video content.

Vegard Aalbu

Master's Thesis Spring 2014

MovieCutter

Vegard Aalbu

May 2014

Abstract

During a conversation you think of some scenes in movies that will illustrate what you are talking about. What if you can instantly access these movie scenes, customize them, and then make your point? This is possible using adaptive video streaming technologies. A video resource is cut into many small video segments to be played back in sequence. The same resource is also encoded in multiple qualities and a client player can switch between these qualities to ensure smooth playback. This adaptiveness optimizes streaming over the Internet.

The thesis will investigate how adaptive streaming can be used to create a montage from events in a video archive. We will use textual metadata resources like subtitles and chapters to search for and present specific scenes within movies. A video resource using adaptive streaming consists of small video segments that can be rearranged out of chronological order. This makes it possible to rearrange movie events in a playlist before being played back via adaptive streaming. We have made a system that reedits multiple movies in a matter of seconds. Not happy with how the movie ended? We present a recipe on how to create your own directors cut.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my supervisor professor Pål Halvorsen, professor Carsten Griwodz and Håkon Kvale Stensland for guidance, proofreading and helpful suggestions. I also would like to thank Vamsidhar Reddy Gaddam for providing helpful suggestions and encouragement.

Chapter 1

Introduction

1.1 Background

We begin this thesis with a story about two friends. Fourteen years ago, a participant in the research team moved overseas for a few years. A friend wanted to give him a goodbye present. This friend had edited a four hour long VHS [195] tape from his video collection. It was a personalized montage of his favorite scenes from different movies. To our knowledge, he did not have access to professional video editing equipment. He had probably used two VHS machines to create the tape. It was thoughtfully edited, and we believe he had spent quite a lot of time creating it.

Montages of different movie scenes can be used for several purposes. Our story is an example of people wanting to share their favorite movie scenes with friends. This is similar to a mixed tape in music. A selection of songs from different artists can be played back at a party or on a road trip. It is a simple technique to trigger emotions and to help us tell a story about ourselves or others.

The broadcasting industry has been using this technique throughout history to make audio or video summaries for all receiving devices. People are used to radio and television channels. They passively receive what the broadcasting network decides that you will experience while tuning in to their channel. In other words, they tell the story. For example, a music channel plays popular songs or a film review program show samples of different movies. Sometimes they broadcast something you want to experience again.

For such needs, you can get your own copy of a product. Technology has played a major part in how people get access to such material. An early audio format was the LP vinyl record that became a standard in 1948 [158]. Later technologies were the magnetic tape audio cassettes [131] and VHS [195] for video. From the late 1990s [192] and until now, we buy optical discs like DVD's [135] and Blu-ray's [129]. Audio and video technology improvements have made it simpler for people to share their favorite media content with friends.

A technology leap from analog to digital media has made access to such material much easier. Today, we can download almost any movie or song from the Internet. Web pages like Spotify [100] offer easy access to favorite songs and the possibility to create audio playlists on the fly. Songs can be played back at home, at parties or other social occasions. "You can listen to the right music, wherever you are" [100]. Video on demand (VOD) [199] services such as Netflix makes it possible "to watch movies or television series anytime anywhere" [86].

Some content are more popular than other. Numerous websites have pop-cultural references to different movies and songs. For example, the Internet Movie Database (IMDb) [52] includes a lot of information about movies and television series. Learning about a particular movie or television show, we sometimes want to watch a specific dialogue between characters, or a certain scene within a movie or program. We can of course playback a DVD or VOD movie to locate that specific scene. Such an approach is time-consuming, especially since we instead can use services like Youtube [208]. This video-sharing website offers access to a large video archive. "One hundred hours of video are uploaded to YouTube every minute. Totally over six billion hours of video are watched by more than 1 billion unique users each month" [209]. These videos are uploaded by individuals or media corporations to be viewed and shared with other users. Uploads are edited manually by the contributor and what you playback is a result of their storytelling. Such a video might or might not be what you are looking for.

Watching videos over the Internet requires streaming [188]. This technology works by sending a large video file in sequence from a server to end users. Streaming video requires a lot of data to be transported to consumers potentially equipped with different receiving devices. The goal being that the consumers receive the streamed video in the best quality available for their device. "The resulting user experience is one of reliable, consistent playback without stutter, buffering or "last mile" congestion" [211]. To achieve this, one key component is the ability to stream video at multiple bit rates [128]. This means the video is encoded [198] in multiple qualities. Then, it is necessary to evaluate the consumers local bandwidth and CPU condition. The result of this evaluation decides at what quality the video is streamed to the consumer. However, factors such as local bandwidth and CPU condition will change during a video playback. It is also necessary to dynamically reevaluate local conditions, to be able to switch seamlessly between different qualities being streamed to the consumer. Today, this is solved using adaptive streaming [94].

Several companies are developing adaptive streaming technologies. Companies like Apple [13, 14, 31, 95], Microsoft [75, 93, 211] and Adobe [5] are competing for the best technology. At the same time, they collaborate with standard groups like the 3rd Generation Partnership Project (3GPP) [1] and the Moving Pictures Experts Group (MPEG) [28, 99, 117] for interoperability.

For example, Internet Information Services Smooth Streaming (IIS Smooth Streaming) is Microsoft's approach to adaptive streaming. This protocol was introduced in 2008. That year, they demonstrated a prototype version of Smooth Streaming by delivering live and on-demand streaming content of events such as the Beijing Olympics and the Democratic National Convention [93]. This prototype version relied on segmenting long video files into small multi-second parts [122]. Such segments are called video segments, fragments or chunks.

"The concept is remarkably simple: if a Smooth Streaming manifest is like a playlist of video/audio fragments which live in the cloud, then editing, merging and appending multiple Smooth Streaming sources should be as simple as re-arranging entries in a manifest. Since Smooth Streaming video is just a series of fragments and each 2-sec fragment must be downloaded separately, it's completely irrelevant whether fragments are downloaded/played in a sequential or non-sequential order – the end result plays equally smooth either way. This is something that hasn't been possible with Windows Media or any other streaming media technology until now. It's a total game-changer" [212].

Nonlinear narrative [168] is a storytelling technique where events can be portrayed out of chronological order. Classical movies like Orson Welles [55]' Citizen Kane uses a non-chronological flashback narrative. The film begins with a news reel detailing Kane's life for the masses, and then from there, we are shown flashbacks from Kane's life [46]. Another example is the movie Pulp Fiction directed by Quentin Tarantino [57]. In this movie, we learn about "the lives of two mob hit men, a boxer, a gangster's wife, and a pair of diner bandits" [56]. These four seemingly unrelated stories are interwoven together in a non-linear fashion. The main character is killed in the middle of the movie, but later in the movie we are presented with a scene before he got killed. This kind of story telling can be confusing, but also creates new meaning. In this particular example, it could be interesting to rearrange Pulp Fiction in a linear narrative. Experimenting with the chronological order of a movie can give unexpected and promising results.

Using segmented video for streaming purposes offer new possibilities. For example, we can choose to stream only a particular scene within a movie. If we have access to a large video archive, we can combine this scene with scenes from other movies. Using an accurate search engine on a video archive, we can actually create our own version of a movie or a montage of movie scenes on the fly.

Video summaries from movies can also be used for other purposes. Imagine you are telling a story. Talking about a topic a few movies scenes come to mind. You want to use these movie scenes as examples, but you are not sure if your audience know these particular scenes. You need a tool to instantly access these scenes. Using adaptive streaming and text-based metadata about movies, we will find out how this is possible.

1.2 Problem Definition / Statement

Remember the story about the VHS montage? Our overall goal is to create a similar montage in a matter of seconds. An end user should be able to do arbitrary searches for content, and on the fly generate a personalized video playlist. To achieve this, we have to research the different components our system will consist of:

- We will compare different adaptive streaming technologies. Using adaptive streaming requires a specific workflow. We will explain how this is done, and why it performs better than traditional streaming.
- We will investigate how we can use video segment ranges for storytelling purposes. Imagine a movie as an indexed horizontal axis where every index is a video segment. Adaptive streaming usually presents video segments in sequence along this axis. We challenge this way of thinking by adding, removing and rearranging video segments to create new meaning.
- We will investigate different approaches for searching in video resources. Video segments have a duration of a few seconds each. Is it possible to map metadata [118] to the corresponding video segments? We will present a prototype [63, 64] where this kind of mapping has successfully been developed using sporting events. Then, we will investigate how this can be done with movie metadata events.
- We will use different searching techniques to access particular scenes within movies. This includes a text-based search engine using wild cards. It will also be possible to locate movies using genre selection.
- We will develop a web based user interface using these components. This website will have access to a video archive with a collection of movies for demonstration purposes.
- This proof of concept [174] system will be user evaluated. Feedback from this evaluation will help us to improve the system.

1.3 Limitations

We will present and compare three different streaming technologies. Two of these are proprietary technologies [175], while the third technology is an international standard [153]. Interoperability is one of the challenges with adaptive streaming technologies and web development in general. This will be addressed in chapter 2. In chapter 3, we implement our system using Apple's HTTP Live Streaming (HLS). This adaptive streaming technology works natively in the Safari [16] web browser. Other web browsers requires plug-ins such as Adobe Flash [4] based video players [68]. We limit our prototype to the Safari browser.

A video archive contains prerecorded video resources. Adaptive streaming technologies include support for live broadcasts over the Internet [155] as well. We will describe how adaptive streaming technologies support live streaming in chapter 2, but we only need to implement the VOD capabilities of HLS in this system.

We will present different metadata search options. Our search engine will use two different metadata resources. However, an advanced search engine is not the focus of this prototype. We will only provide enough search options to demonstrate proof of concept.

Our system searches a video archive containing movies for demonstration purposes only. This is not a thesis focusing on digital rights management (DRM) [133]. In chapter 2, we describe how it is possible to encrypt adaptive streaming technologies. However, we have not implemented encryption in this prototype. Our system's website is password protected and is only intended for proof of concept.

The prototype's website will be used to test how this system handles adaptive streaming. We limit the adaptive streaming capabilities to two different video qualities. This is enough to demonstrate that the technology adapts to changes in CPU usage and network traffic conditions. Our chosen bit rates are only intended for noticing when a client player changes picture quality. The system is not optimized for scalability. This is outside the scope of this thesis.

We will use a landscape dimension as style rule in the website. This should present the website properly on for example laptop devices. Using techniques such as responsive web design [181] to include all receiving devices, is not the focus of this prototype.

1.4 Research Method

Our research method for this thesis will be the design paradigm method of the ACM classification [26]. We will create a design and implement a system for demonstration of proof of concept. Hence, functionality expected by some users might not be included in this system. We will conduct a user survey using a 7-point balanced keying Likert scale to evaluate the usefulness of this system. "When responding to a Likert questionnaire item, respondents specify their level of agreement or disagreement on a symmetric agree-disagree scale for a series of statements. Thus, the range captures the intensity of their feelings for our system" [157].

1.5 Main Contributions

We have created a system for extracting events within a video archive. This means we can search for a specific scene within a movie and extract that particular event. An end user can then create a personalized video summary. By dragging multiple events into a playlist, an end user can rearrange and customize the durations of events. Afterwards, this video summary can be played back using adaptive streaming.

Extracting accurate events is possible using our metadata extraction component. We parse text-based documents containing time codes and a textual description of an event. Metadata resources such as subtitle and chapter events are downloaded from online movie databases. During parsing, extracted metadata are stored in a relational database.

Our search component includes a text based search engine using wild cards. It is also possible to search for movies and chapters using genre selection. An extracted event is then presented with a storyboard thumbnail image and information about that particular event. Such information includes event type, duration and a textual description.

An end user can preview an extracted event by dragging it into the video player in the user interface. Alternatively, an end user can also create a personalized video summary using the system's playlist component. By dragging multiple events into a playlist, an end user can rearrange and customize the durations of events. Either way, the system creates manifest files to playback events. A manifest file is a playlist of file path references to video segments on the server. This function calculates one or many events included in a request from a client device. Then, it creates three temporary manifest files to prepare a video summary for adaptive streaming. A variant manifest file contains some adaptive logic and the file paths references to other manifest files of a low and high bit rate. The URL of the variant manifest file is returned to the client video player for play back.

We have performed a user study among both technical and non-technical persons to evaluate this system. During this survey, the participants were asked to create video montages and give feedback about the system's functionality. Overall, people were positive to our system. All general questions in the survey received a high score in the Likert scale. We scored especially high on statements like "It is fun to use", "It responds fast to queries and playback" and "It plays back accurate video scenes".

Low latency and accuracy were two of our primary goals with this system. We have implemented a running prototype that can reedit multiple movies in a matter of seconds. The system is able to present specific scenes within a movie by mapping accurate metadata with video segments. These events from multiple movies can then be rearranged in a playlist before being played back in the video player. The video player changes between different video qualities using adaptive streaming. On the basis of subtitle and chapter metadata resources, we can create our own directors cut.

1.6 Outline

In chapter 2, we present various related works. We will introduce adaptive streaming as a result of earlier streaming technologies. Several adaptive streaming technologies are compared. Next, we describe video summary systems using internal and external technologies. Finally, we research possible movie metadata resources. We investigate online databases in search for metadata resources containing both time codes and textual descriptions.

In chapter 3, we present the design and implementation of the system. We will extract metadata from resources found in online databases. This data will be mapped to a range of video segments. These events can be queried and presented in a user interface. It will be possible to rearrange movie scenes in a playlist and to playback the video summary.

In chapter 4, we present a survey where users have tested our prototype. We also evaluate the system's latency and the accuracy of mapping metadata resources to video segment ranges. Based upon what we learned from these evaluations, we discusses further improvements of the system.

In chapter 5, we conclude the thesis. This chapter is a short summary of what we achieved. We describe the final result and present an outlook for how this system can be used in future work.

Chapter 2

Related Work

We want to develop a system that makes personalized video summaries from achieved video content. This system will require different components such as an adaptive streaming technology, and a searching technique that can locate specific video segment ranges to be played back to an end user. In this chapter we will look at related and usable technologies in this respect. First, we will compare different adaptive streaming technologies to investigate which streaming technology is best suited for our needs. Later, we compare internal and external approaches for creating video summaries. Finally, we research possible movie metadata resources.

2.1 Streaming evolution

"The world's first live streaming event happened on 5 September 1995" [210]. ESPN SportsZone streamed a live radio broadcast of a baseball game. Thousands of subscribers worldwide experienced this match using a technology developed by a Seattle-based company named Progressive Networks. They later changed the company name to RealNetworks [178]. Their main competitor was Microsoft, and they both tried to dominate this new technology market. Microsoft won this competition because of its Windows Media technologies [205], but the world was not quite ready for streaming solutions. "Even though the prospect of streaming media over the internet had always excited technology nerds and CEOs alike, streaming media's childhood years were primarily marred by pragmatic problems such as how to successfully stream watchable video over 56k modem lines" [210].

In the mid-2000s, Windows Media's market share dropped in favor of Macromedia's [160] popular Flash [4] player. This company later became part of Adobe Systems [2]. "The Flash player was a successful combination of interactivity, Web 2.0 and streaming media. However, issues such as bandwidth, scalability and reach was holding the streaming media industry back" [210].

At the same time, the vast majority of the Internet traffic was Hypertext Transfer Protocol (HTTP)-based [150], and proxy solutions like content delivery networks (CDN) [132] were used for scalability. This ensured delivery of popular content to large audiences. Traditional streaming relying on proprietary protocols and non-firewall friendly transport protocols like UDP, faced a challenge in this environment. In 2007 the company Move Networks [77] introduced a technology that would change the industry [210].

Move Networks used adaptive HTTP-Streaming. This new way of thinking uses components of earlier streaming technologies. Being a hybrid of these technologies, it is helpful to describe the main concepts of these technologies to fully understand how adaptive streaming work.

2.1.1 Traditional streaming

Real-Time Streaming Protocol (RTSP) [42] is an example of a traditional streaming protocol. Traditional streaming is defined as stateful. "A stateful service is one where subsequent requests to the service depend on the results of the first request. Stateful services are more difficult to manage because a single action typically involves more than one request, so simply providing additional instances and load balancing will not solve the problem" [88]. The streaming server keeps track of which clients are connected to it. During a session, the client communicates its state to the server by commands like play, pause or closing the connection. Figure 2.1 illustrates this workflow.

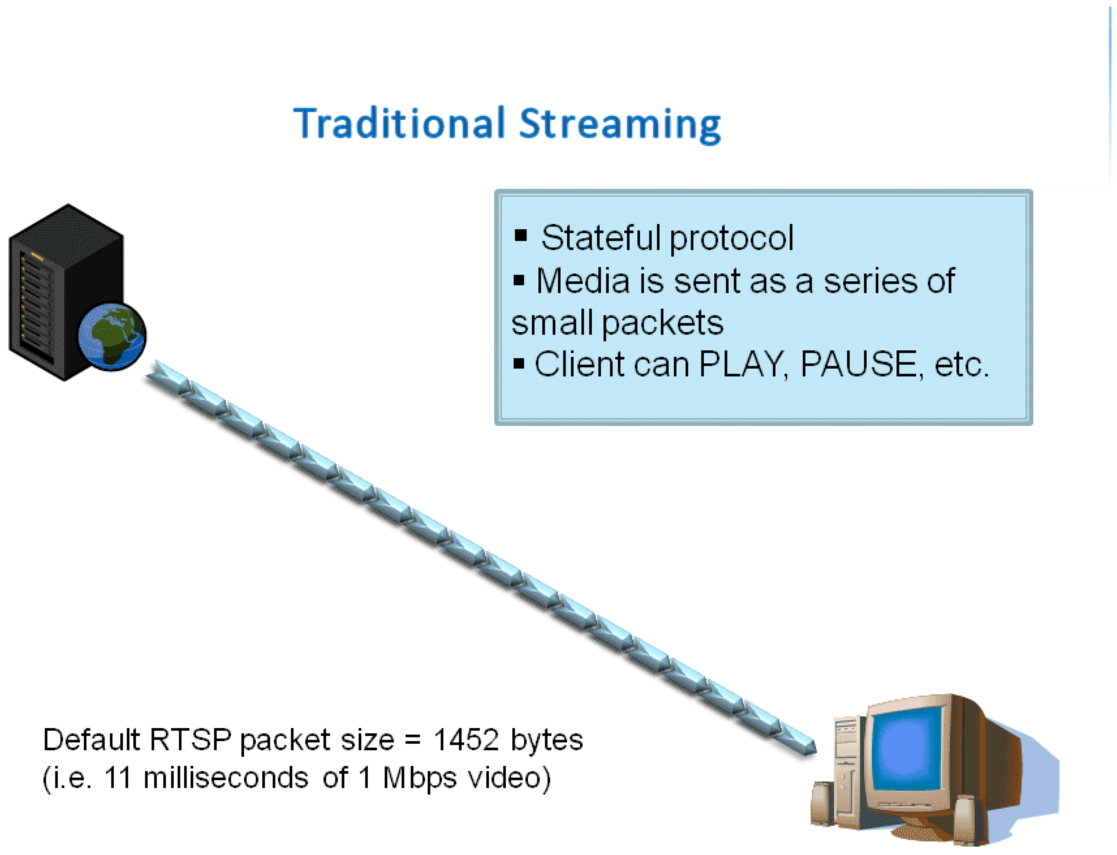

Traditional Streaming
▪ Stateful protocol
▪ Media is sent as a series of small packets
▪ Client can PLAY, PAUSE, etc.
Default RTSP packet size = 1452 bytes
(i.e. 11 milliseconds of 1 Mbps video)

Figure 2.1: Traditional Streaming [211]

The server streams small packets using the Real-Time Transport Protocol (RTP) [177] format. "Default packet size is 1452, meaning a streamed packet contains for example 11 milliseconds of encoded 1 megabit per second video" [211]. Packets can be transmitted over either UDP or TCP transport layer protocols. Neither protocol is ideal for streaming purposes. Firewalls or proxies can block UDP packets because of unfamiliar port numbers. TCP packets on the other hand, are re-sent until received, which can lead to increased latency.

The server sends data packets to the client at a real-time rate only. For example, "a video encoded at 500 kliobits per second (KBPS) is streamed to clients at approximately 500 KBPS" [211]. The server only sends enough data to fill the client buffer. This is typically between 1 and 10 seconds.

2.1.2 Progressive download

Progressive download has another approach to media delivery. It is a video file being downloaded from a HTTP Web server. Playback from client media players are allowed while the download is still in progress. If both the client and the server support the HTTP 1.1 specification [115], it is possible to seek positions in the video file in a non-linear fashion. The client can send requests for video that has not been downloaded yet. "Popular video sharing web sites on the web today including YouTube [208], Vimeo [114] and MySpace [83] almost exclusively use progressive download" [211].

Progressive download streams video differently than traditional streaming. The server keeps transferring a file until it is completely downloaded by the client. This means the server will send more data than a client buffering in traditional streaming. Eventually, the entire video is downloaded to your browser cache. "The downside being if the client do not want to watch the rest of the movie. -if 30 seconds into a fully downloaded 10 minute video, you decide that you don’t like it and quit the video, both you and your content provider have just wasted 9 minutes and 30 seconds worth of bandwidth" [211].

2.2 Adaptive HTTP-Streaming

Adaptive streaming acts like traditional streaming, but is based on HTTP progressive download. Media download is performed as a long series of small downloads, rather than one big progressive download. Figure 2.2 illustrates this workflow.

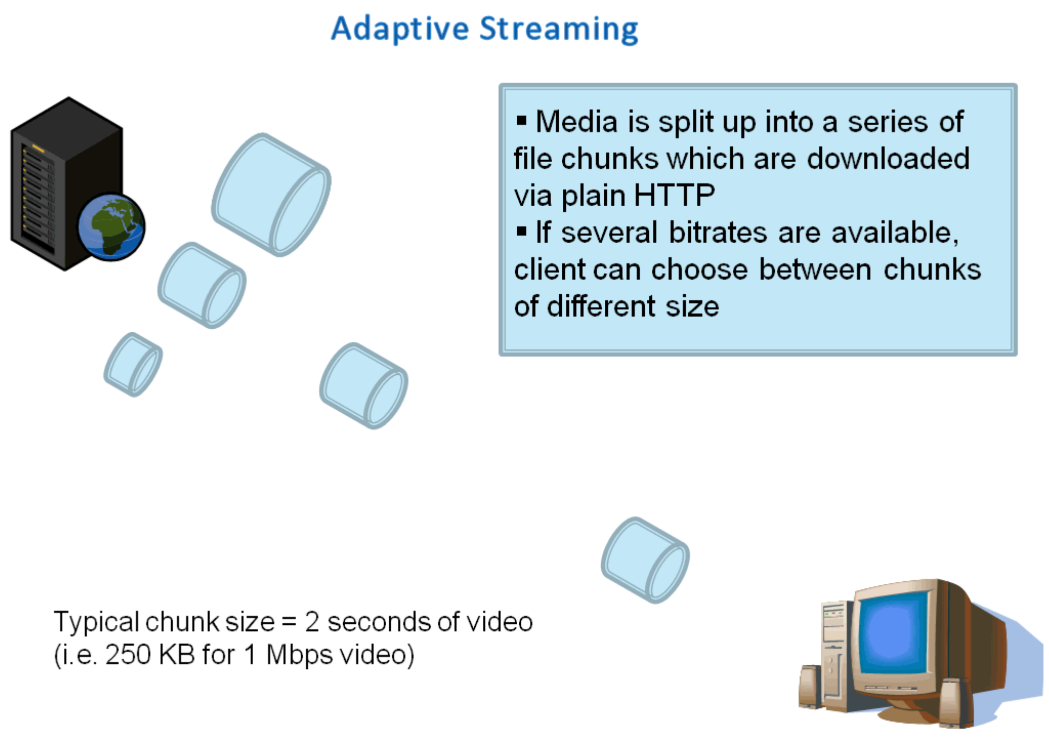

Adaptive Streaming
▪ Media is split up into a series of file chunks which are downloaded via plain HTTP
▪ If several bitrates are available, client can choose between chunks of different size
Typical chunk size = 2 seconds of video
(i.e. 250 KB for 1 Mbps video)

Figure 2.2: Adaptive Streaming [211]

Streaming video requires a lot of data to be transported to end users with different receiving devices. The goal being that consumers receive the streamed video in the best quality available for their device. To achieve this, one key component is the ability to stream video at multiple bit rates. This means a video resource is encoded in multiple qualities. The chosen bit rate for playback is transported using HTTP.

2.2.1 The HTTP component of adaptive streaming

There has been a shift away from classic streaming protocols like RTSP to HTTP progressive download. There are several reason why HTTP is preferred in the streaming media industry.

- It is cheaper. Web download services have been traditionally less expensive than media streaming services offered by CDNs and hosting providers. It is easier to move HTTP data to the edge of the network, thus being closer to end users.
- Media protocols does not work well with firewalls and routers. They usually use UDP sockets over unusual port numbers. HTTP-based delivery goes through port 80, which is a familiar port.
- Web cache treat media files like any other file. No special proxies or caches are necessary.

The Internet was built on HTTP and optimized for HTTP delivery. Why not adapt media delivery to the Internet instead of trying to adapt the entire Internet to streaming protocols" [211]?

HTTP is a stateless protocol. "A stateless service is one that provides a response after your request, and then requires no further attention. To make a stateless service highly available, you need to provide redundant instances and load balance them" [88]. This means the web server does not keep track of the client state like a traditional streaming protocol. Each HTTP request is handled as a standalone one-time session. To be able to stream video using HTTP, the client needs to keep track of it's own state. If an end user presses pause or play, the client needs to know what to require next from the server. This shift from server to client state is beneficial for both content provider and end user. The content provider traditionally had to decide a video quality for all end users depending on the end user with the less bandwidth. This was an unfair solution for end users with higher bandwidth.

A video stream is a large collection of images put together. Each image has a certain size and quality. Receiving devices has different requirements to quality, depending on the device available resources and the end user preferences. Each client is most capable of figuring out its local bandwidth and CPU consumption. The result of this evaluation decides at what quality a video resource is streamed from the server to the consumer. However, factors such as local bandwidth and CPU condition will change during a video playback. It is also necessary to dynamically reevaluate local conditions. The client video player logic should then be able to request a different quality from the server and switch seamlessly between qualities during playback. "The resulting user experience is one of reliable, consistent playback without stutter, buffering or "last mile" congestion" [211].

2.2.2 Video fragmentation and I-frames

When transporting files via the Internet, the files need to be fragmented [139] in order to fit into packets. Hence, it seems like a good idea to fragment a video of for instance one minute into multiple parts of some seconds each. Thus, the video/audio source is cut into many short segments with an encoding of the desired delivery format. These segments are typically 2 to 10 seconds long. The length of the segmented chunks are usually along video codec friendly group of pictures (GOP) boundaries. GOP is a group of successive pictures within a coded video stream. When compressing large video sources for transport, a usual technique is to reduce redundancy information in the images the video resource is made of. GOP consists of different kind of picture frames, for example MPEG-2 [163] uses three frame types I, P, and B frames [143]. GOP always begins with a reference picture, an I-frame. This is also known as the key frame. Every video segment contains at least one I-frame. The I-frame is the only MPEG-2 frame type which can be fully decompressed without any reference to frames that precedes or follows it. It is also the most data-heavy, requiring the most disk space.

2.2.3 Server and client communication

The encoded segments are hosted on a HTTP web server. These video chunks are then downloaded in a linear fashion using HTTP progressive download. Since the video resource was encoded and segmented without gaps or overlaps between the video chunks, the client video player can playback video segments in a linear order that appears as seamless video. The size of the client buffer is also fully customizable.

A segment can be encoded at multiple aspect ratios and bit rates. Web servers usually deliver data as fast as network bandwidth allows them to. The client can estimate user bandwidth and CPU usage and decide to download larger or smaller chunks ahead of time. "This is the 'adaptive' part of adaptive streaming, as the mobile client can choose between different bit rates/resolutions and adapt to larger or smaller chunks automatically as network conditions change" [94].

2.2.4 Benefits for distributor and end user

Adaptive streaming is beneficial for the content distributor. This technology uses HTTP caches/proxies. Thus, cheaper than specialized servers at each node. Another benefit is better scalability and reach to end network, since it can dynamically adapt to inferior network conditions. Clients can adapt to the content instead of content providers guessing what bit rate is most likely to be accessible for all clients.

There are also benefits for end users. Adaptive streaming supports fast start-up and seek times. It initializes at the lowest bit rate before changing to a higher bit rate playback. As long as the user meets the minimum bit rate requirements, there should be no buffering, disconnects or playback stutter. Client based adaption examines the local bandwidth and CPU capabilities to request the best quality available.

2.2.5 Industry support

Apple [13, 14, 31, 95], Microsoft [75, 93, 211] and Adobe [5] are competing for the best proprietary adapting streaming technology. At the same time, they collaborate with standard groups such as 3GPP [1] and MPEG [99,117]. Other companies like Google [37] and Netflix [86] also participate in implementing an interoperable adaptive streaming technology [28].

Microsoft and Adobe both uses plug-in programs in web browsers, being the Silverlight [74] player and the Flash [4] player. Apple's technology is supported natively in their Safari [16] web browser and other Apple products. We will look into how these companies approaches adaptive streaming. Since Microsoft and Adobe both uses plug-ins, we have chosen to focus on only one of them.

2.3 HTTP live streaming

HLS is Apple's approach to adaptive streaming. The technology supports live broadcasts and prerecorded content. Playback is intended for iOS devices - including iPhone, iPad, iPod touch, Apple TV and on desktop computers. "HLS was originally unveiled by Apple with the introduction of the iPhone 3.0 in mid-2009" [95]. In earlier iPhone versions no streaming protocols were supported natively. Apple has proposed HLS as a standard to the Internet Engineering Task Force (IETF) [109]. The protocol has gone through several iterations.

2.3.1 HLS overview

The protocol lets you stream video and audio to any supported Apple product including Macs with a Safari web browser. Safari plays HTTP live streams natively as the source for the HTML5 video element [148]. The workflow begins with an input stream being encoded to a supported format. Then, the HLS protocol streams audio and video as a series of small media files. These segmented files have a recommended duration of ten seconds [17]. The technology supports multiple alternate bit rates for each video fragment. The client software can switch streams intelligently as network bandwidth changes [13]. An index file, also referred to as a playlist or manifest, shows the client the URLs of the media segmented files. Figure 2.3 illustrates the workflow.

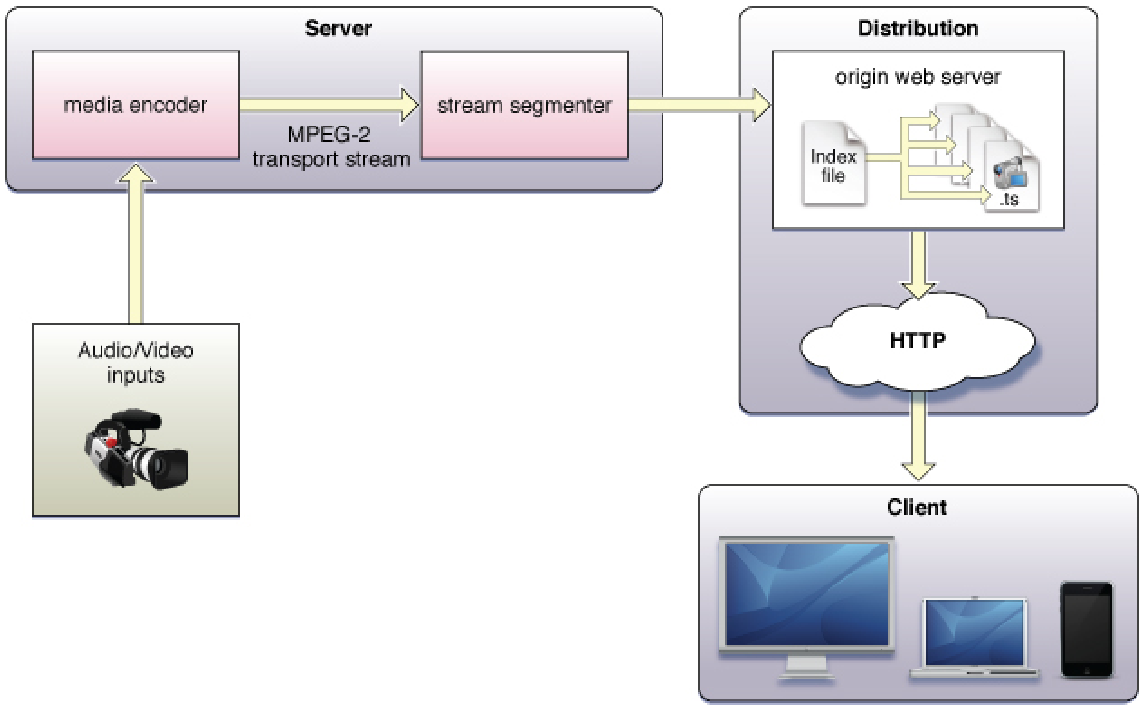


Figure 2.3: HLS transport stream [13]

2.3.2 HLS architecture - server, distribution and client

For live purposes, the server component receive input streams and prepare the media for distribution. This includes encoding and encapsulation in a format suitable for delivery. For VOD purposes, it stores the preprepared media. The server component is distributed from a standard web server like other world wide web resources. It accepts client requests and deliver the media and associated resources to the client. It is recommended to configure the web server to specify MIME-type associations for .m3u8 index files and .ts segment files. Because HLS uses HTTP, this kind of streaming is automatically supported by nearly all edge servers, media distributors, caching systems, routers, and firewalls [13]. HTTPS [149] can also be used for adaptive streaming, but is not recommended. Such requests often bypass web server caches, causing all requests to be routed directly to the web server. Hence, defeating the purpose of edge network distribution systems.

The client component determines the available quality of media to request. Then, these resources are downloaded and finally reassembled in order to present a continuous stream to an end user.

2.3.3 MPEG-2 Transport Stream

Typically, a hardware encoder receive audio-video input and encodes it as H.264 [145] video and Advance Audio Coding (AAC) [123]. Then, the encoded media is put inside a MPEG-2 transport stream (MPEG-2 TS) [165] for audio-video, or MPEG elementary stream [136] for audio-only. Note that HLS does not make use of Apple's Quicktime [176] .mov file format, which is the basis for the ISO MPEG [164] file format, later known as MPEG-4. "The MPEG-4 file format specification was created on the basis of the QuickTime format specification published in 2001" [113]. Apple thought that the MPEG-2 TS format was more widely used and better understood by the broadcasters who would ultimately use HLS [13]. They focused on the MPEG-2 TS format.

Note that MPEG-2 TS should not be confused with MPEG-2 video compression [196]. "The transport stream is a packaging format that can be used with a number of different compression formats. Only MPEG-2 transport streams with H.264 video and AAC, AC-3 [107] or MP3 [162] audio is supported at this time for audio-video content" [13].

2.3.4 Developer tools

Several tools for setting up HLS are available from the Apple developer website [15]. Tools include a media stream segmenter and a file stream segmenter, a stream validator, an id3 tag and a playlist generator.

- The segmenter tools are for live streaming and prerecorded media. The media stream segmenter receives an MPEG-2 TS as input and creates a series of equal-length files from it. The media file segmenter receives an encoded media file as input, wraps it in an MPEG-2 TS format, and produces a series of equal-length files from it. Both tools can generate index files, encrypt media content, produce encryption keys, optimize the files by reducing overhead, and create the necessary files for automatically generating multiple stream alternates [13].
- The media stream validator examines the output of the segmenter tools and tests whether they will work with HLS clients. It simulates an HLS session and verifies that the index file and media segments conform to the HLS specification. The validator performs several checks to ensure reliable streaming.
- The variant playlist creator creates the primary or master index file. It uses the output of the media file segmenter to create references to other index files of different quality. This is necessary to allow the player to switch between different bit rates.
- The metadata tag generator makes ID3 metadata tags. "ID3 is a metadata container most often used in conjunction with the MP3 audio file format. It allows information such as the title, artist, album, track number, and other information about the file to be stored in the file itself" [151]. These tags can either be written to a file or inserted into outgoing stream segments.

2.3.5 Segmenter and .m3u8 index file

The MPEG-2 TS is broken into a series of short media files with the extension .ts by a segmenter tool. Typically, a single input stream will be transcoded to several output resolutions depending on the types of client devices using the stream. "For example, an input stream of H.264/AAC at 7 Mbps can be transcoded into four different profiles with bit rates of 1.5Mbps, 750K, 500K, and 200K. These will be suitable for devices and network conditions ranging from high-end to low-end, such as an iPad, iPhone 4, iPhone 3, and a low bit rate version for bad network conditions" [95]. Figure 2.4 illustrates an example of multiple bit rates.

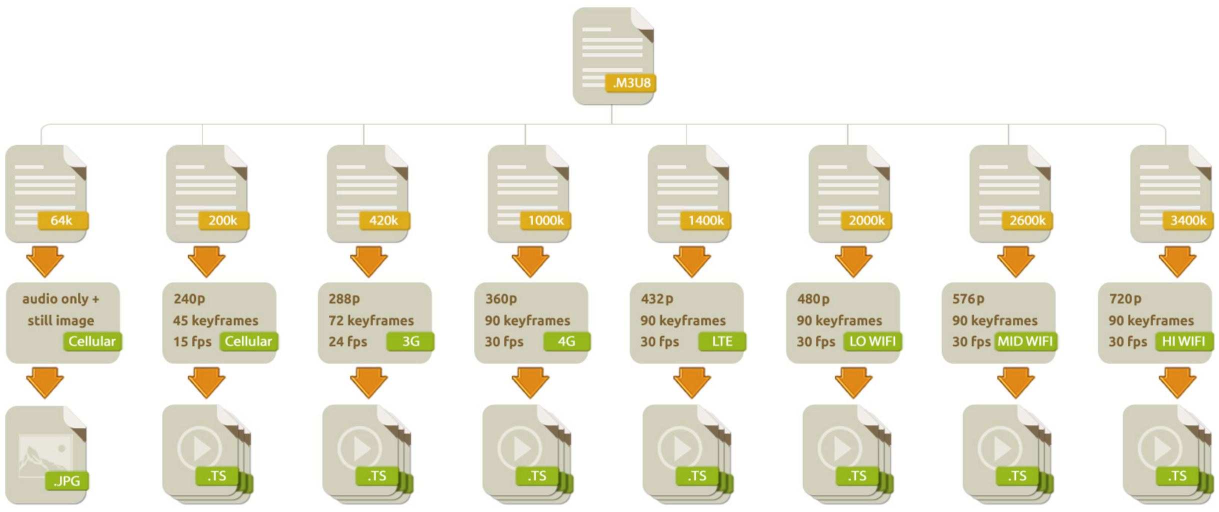

.M3U8
64k
200k
420k
1000k
1400k
2000k
2600k
3400k
audio only + still image Cellular
240p 45 keyframes 15 fps Cellular
288p 72 keyframes 24 fps 3G
360p 90 keyframes 30 fps 4G
432p 90 keyframes 30 fps LTE
480p 90 keyframes 30 fps LO WIFI
576p 90 keyframes 30 fps MID WIFI
720p 90 keyframes 30 fps HI WIFI
.JPG
.TS

Figure 2.4: HLS bit rates [13]

A segmenter tool also creates an index file to keep track of each media segment. Otherwise, the stream might be presented to an end client out of order. It is basically a playlist containing a list of references to segmented video files on the server. Note that we want to manipulate this index file in order to play selected parts of a video resource. We elaborate on how to do this in chapter 3, section 3.3.

The following example illustrates a manifest file containing three unencrypted 10-second video segment files:

```
#EXT-X-VERSION:3
#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:1

# Old-style integer duration; avoid for newer clients.
#EXTINF:10,
http://a.media.com/chunk0.ts

# New-style floating-point duration; use for modern clients.
#EXTINF:10.0,
http://a.media.com/cunck1.ts
#EXTINF:9.5,
http://a.media.com/cunck2.ts
#EXT-X-ENDLIST
```

The #EXT-X-TARGETDURATION:10 parameter defines the video segment duration. Newer versions of the protocol can use decimals when defining segments. This is useful since not all video resources fit inside a fixed duration constraint. For example, the last segment of a video segment range might not have an exact duration of ten seconds.

2.3.6 Adding timed metadata with the segmenter

Timed metadata is used to insert descriptive data into a media stream at a given time offset. It can be inserted into individual segments or all segments after a given time. For example, "you can add the current batter's name and statistics to video of a baseball game" [13]. Metadata needs to be formatted in an .id3 [151] or image file, so that a segmenter tool can automatically insert the metadata into every media segment. Metadata is persistent once it has been inserted into a media segment. For example, if a live broadcast is to be reused for VOD playback, the metadata is still there. The id3tag generator is used for adding timed metadata to a live stream.

The file segmenter tool requires a metadata macro file. A timing parameter to insert the metadata, the type of metadata, and the metadata file path are all included into each line of the macro file. For example, an audio stream macro file can look like this:

```
3.5 picture  /meta/image.jpg
8 id3 /meta/title.id3
```

2.3.7 Client device

The client downloads the index file using an Uniform Resource Locator (URL) address that locates the manifest file on the server. The index file specifies for the client where to get the stream chunks, decryption keys and any alternate streams available [13]. For a given stream, the client software then fetches each stream chunk in a linear fashion.

Once the client has enough media downloaded and buffered, the client player displays the media without any pauses or gaps between segments. This process continues until the client encounters the #EXT-X-ENDLIST parameter in the index file. This command tells a client video player that the playback is over.

Note that this tag is not present in a live broadcast. This "indicates that new media files will be added to the index file as they become available" [18]. To get the update manifest data, the client loads a new version of the index file periodically. "The client looks for new media and encryption keys in the updated index and adds these URLs to its queue" [13]. This continues until the broadcast is over.

2.3.8 Variant manifest file

A variant manifest file contains a specially tagged list of variant index files if multiple profiles are available. Figure 2.5 illustrates a primary index file referencing three other index files with different profiles.

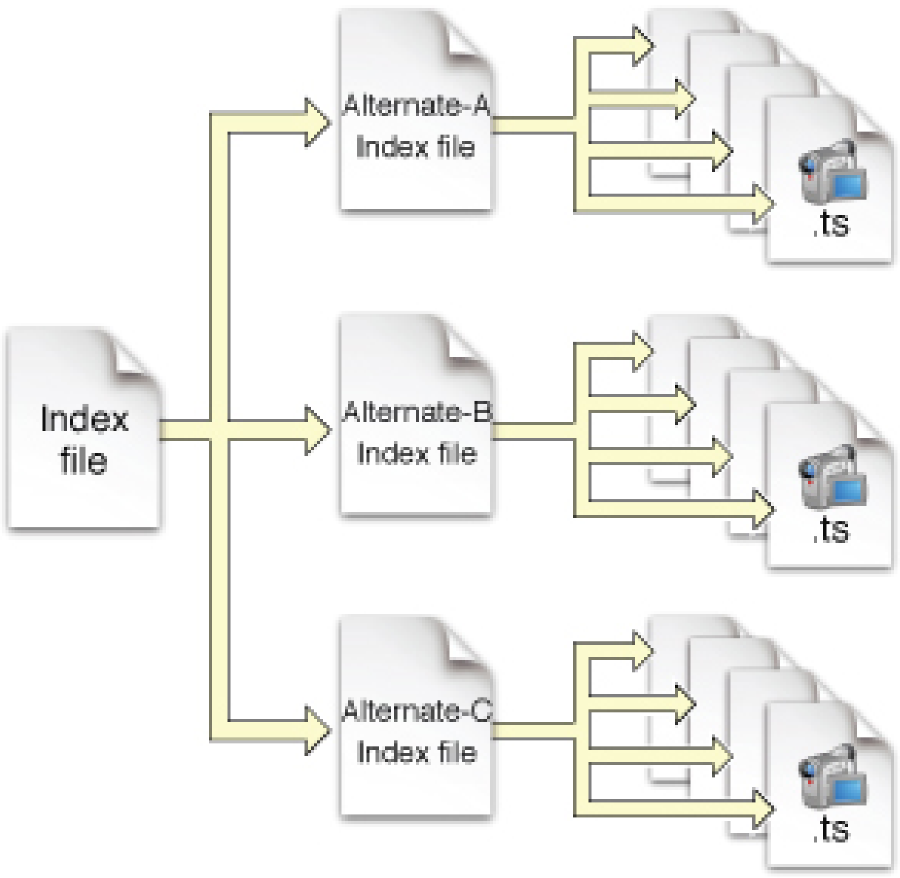


Figure 2.5: HLS indexing [13]

For example, if the media is available in a high and a low quality, it is necessary to use a separate index file for each quality. Meaning, a video stream of two different bit rates requires a total of three different index files. The client first downloads the primary index file. This is done once. "A variant playlist is not re-read. Once the client has read the variant playlist, it assumes the set of variations is not changing" [18]. The bit rates and resolutions of the variant streams are specified in this file. Then, the client downloads the index file for the bit rate it wants to playback. If the client want to change quality during a playback, it has to download the index file for that bit rate as well. For live broadcasts these index files are reloaded periodically.

Typical playback latency for HLS is around thirty seconds. "This is caused by the recommended size of the chunks (10 seconds) and the need for the client to buffer a number of chunks before it starts displaying the video" [95].

The client chooses among the alternate bit rates while considering available bandwidth. "Changing to an alternate stream may happen at any time, such as when a mobile device moves between cellular and WiFi connections, or between 3G and EDGE connections. It is recommended to provide an alternative stream to cellular-capable connections. If you cannot provide video of acceptable quality at 64 Kbps or lower, you should provide an audio-only stream, or audio with a still image" [13]. For smooth transition among streams, it is recommended to use identical audio bit rate for all video bit rates. To achieve this, audio and video has to be encoded separately. Afterwards, they can be multiplexed [166] back together.

2.3.9 Redundancy support

If the media is represented in alternate streams, it is not only for bandwidth and device alternate purposes. It can also be used for failure fallback. "Starting with iOS 3.1, if the client is unable to reload the index file for a stream (due to an 404 error, for example), the client attempts to switch to an alternate stream" [13]. The client chooses the highest bandwidth alternate stream that the network connection supports. Such a feature allows media to reach clients in the event of local failures. For example, if a server crashes or a content distributor node goes down. A .m3u8 index file with redundant stream support might look like this:

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=200000,
RESOLUTION=720x480
http://A.media.com/lo/index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=200000,
RESOLUTION=720x480
http://B.media.com/lo/index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000,
RESOLUTION=1920x1080
http://A.media.com/md/index.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1, BANDWIDTH=500000,
RESOLUTION=1920x1080
http://B.media.com/md/index.m3u8
```

The playlist has a parallel stream for each resolution. The parallel stream is located on a separate server. The backup stream should be listed in the index file after the primary stream. This redundancy technique is not limited to only one backup stream. There can be several. Another type of redundancy technique can be a single low-bandwidth stream on a backup server.

2.3.10 Live broadcasts and VOD sessions

VOD sessions allow clients access to the entire program. The primary index file is static and is downloaded once. It contains a complete list of other index files with available bit rates.

Live sessions have dynamic index files. They are continuously updated. This is because new media files are created on the fly. To avoid an index file becoming to big, it is possible to remove older media files from the index file. If it is necessary to convert a live broadcast to VOD afterwards, then all media files should be present in the index file. "A live broadcast can be instantly converted to a VOD presentation by including an #EXT-X-ENDLIST tag to the index file. This allows clients to join the broadcast late and still see the entire event. It also allows an event to be archived for rebroadcast with no additional time or effort" [13].

2.3.11 Content Protection

Media files may be individually encrypted. References to the corresponding key files are part of the index files, so the client can decrypt the media. Currently HLS supports Advanced Encryption Standard (AES)-128 [124] using 16-octet keys. Key files can be served using either HTTP or HTTPS. They require an initialization vector (IV) [152] to decode encrypted media. The media stream segmenter tool supports three modes of configuring encryption. It is possible to generate a random key file or use an existing key file. If the key is generated randomly, a new key can be made for each interval. Each group of n media segments will then be encrypted using a different key. All media files are encrypted with this key. "It is less likely that changing to a new key periodically will impact system performance than changing keys for each segment. Current recommendations for encrypting media while minimizing overhead is to change the key every 3-4 hours and change the IV after every 50 Mb of data" [13].

2.3.12 Fast Forward and Reverse Playback

A new feature in iOS 5 and HLS protocol version 4 is support for Fast Forward and Reverse Playback. This is possible by jumping between i-frames in the movie. "All you need to do is specify where the I-Frames are. I-Frames, or Intra frames, are encoded video frames whose encoding does not depend on any other frame. To specify where the I-Frames are, iOS 5 introduces a new I-Frame only playlist" [18].

2.3.13 HLS takeaway points

Video segments have a recommended duration of ten seconds. Multiple index files are used to achieve adaptive streaming. The media is transported using MPEG-2 Transport Stream. HLS is optimized for Apple products.

2.4 IIS Smooth Streaming

IIS Smooth Streaming is Microsoft's approach to adaptive streaming. The protocol was introduced in 2008. That year they demonstrated a prototype version of Smooth Streaming by delivering on-demand streaming content of events such as the Beijing Olympics [93]. "Whereas in Beijing we experimented with HTTP adaptive streaming for on-demand SD delivery only, the one thing we all knew for sure as soon as the Beijing closing ceremony was over was that for Vancouver 2010 we wanted to deliver all video in HD, both live and on-demand, using HTTP adaptive streaming. By November 2008 the first glimpses of IIS Smooth Streaming definitely put on-demand HD delivery within reach, and by May 2009 live Smooth Streaming was a reality too" [212].

To begin with, Smooth Streaming relied on physically chopping up video resources of long duration into multiple video chunks of shorter duration [122]. For example, segmenting an olympic event of an hour becomes eighteen hundred video segments of two second duration. After the experiences with the Olympics in 2008, Microsoft decided to improve their file format and server design. Instead of storing a lot of individual video segment files, the protocol supports representing all these video segments as a single file.

2.4.1 Smooth Streaming overview

Smooth Streaming has the typical characteristics of adaptive streaming. The video content is segmented into small video chunks and it is delivered over HTTP. Using metrics based on local CPU processing power and network conditions, the client can choose among multiple video bit rates to receive an optimal viewing experience.

The MPEG-4 file format [164] is used for both storage and transport. If the media is available in multiple bit rates, one MPEG-4 file is necessary for each bit rate. A media file is then virtually split into multiple chunks as fragmented MPEG-4 files. Fragments has less overhead than a regular MPEG-4 file. A reference to fragmented chunks are stored within a contiguous MPEG-4 file for easy random access. The chunk size is typically two seconds. The media file is usually encoded with H.264 video encoding and AAC audio. Also supported is video encoder VC-1 [194] and audio encoder WMA [204].

When a client request chunks from a web server, it dynamically finds the appropriate fragment within the contiguous MPEG-4 file. Then, the server returns the fragment as a standalone file. "In other words, file chunks are created virtually upon client request, but the actual video is stored on disk as a single full-length file per encoded bit rate. This offers tremendous file-management benefits" [211].

2.4.2 MPEG-4 and file management

Microsoft has traditionally used the media file format ASF [125] for streaming. For adapting streaming, they choose the MPEG-4 file specification instead. They did this because of several reasons. MPEG-4 has less overhead than ASF. The specification is based on a widely used standard. MPEG-4 is architected with H.264 video codec support in mind. The MPEG-4 specification allows a file to be internally organized as a series of fragments.

Internally organizing fragments are necessary to make the Smooth Streaming specification work. The goal being to transport multiple small file chunks to a client, but at the same time save the media as a single file on disk. "The videos are no longer split into thousands of file chunks, but are instead "virtually" split into fragments (typically one fragment per video GOP) and stored within a single contiguous MPEG-4 file" [211]. Hence, better file management. Figure 2.6 illustrates an example of the fragment structure.

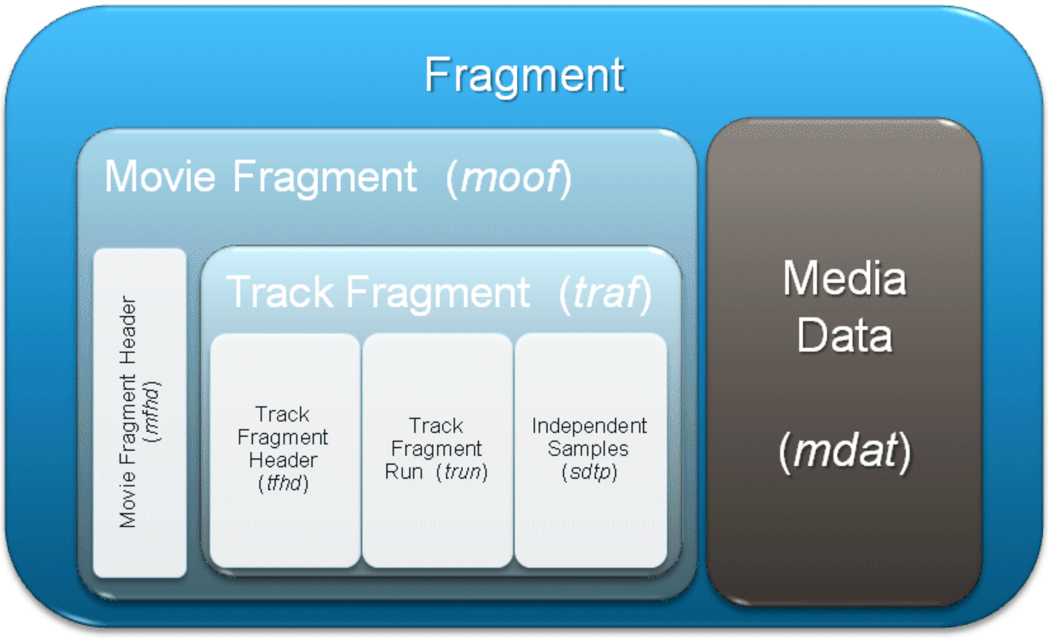

Fragment
Movie Fragment (moof)
Movie Fragment Header (mfhd)
Track Fragment (traf)
Track Fragment Header (tfhd)
Track Fragment Run (trun)
Independent Samples (sdtp)
Media Data (mdat)

Figure 2.6: Smooth Streaming fragment [211]

2.4.3 Disk file format

The basic unit of an MPEG-4 file is called a "box" and contains both video and metadata. These components can be organized in various ways within a file. For example, one might want to organize metadata up-front before the actual data video fragments. This might be useful in a VOD situation when a client video player needs to read information about the video/audio before playing it. On the other hand, metadata up-front causes overhead and slower startup times. In live situations, metadata might not be fully known at the time of broadcast.

The MPEG-4 container can be adapted for different situations. It is possible to write a file "as you go" as a series of short metadata and box pairs rather than one long metadata and data pair. This is important for the Smooth Streaming usage of the file format. Figure 2.7 illustrates the structure of MPEG-4 file in a Smooth Streaming context.

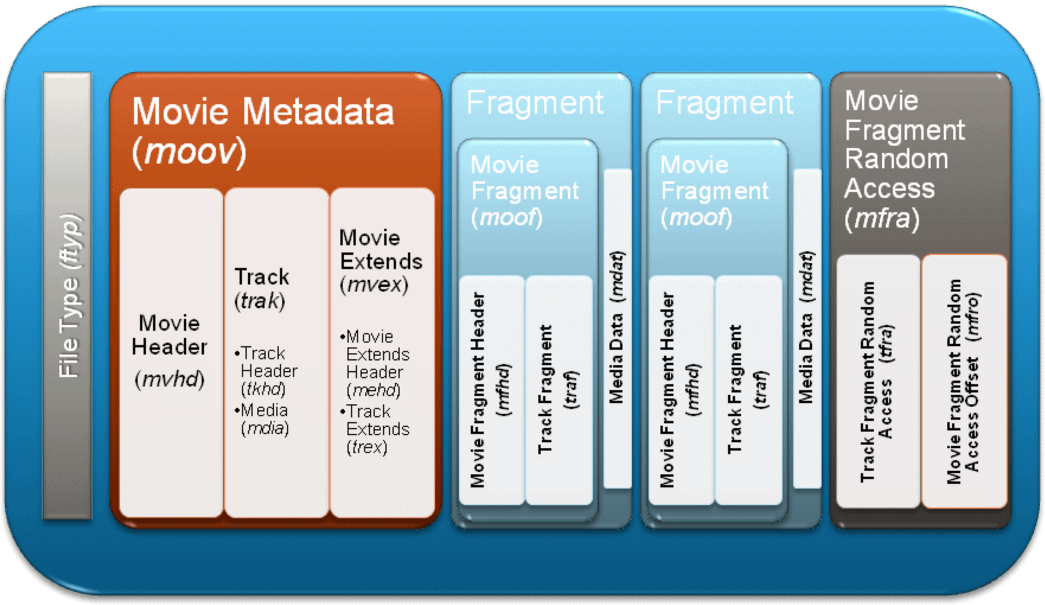


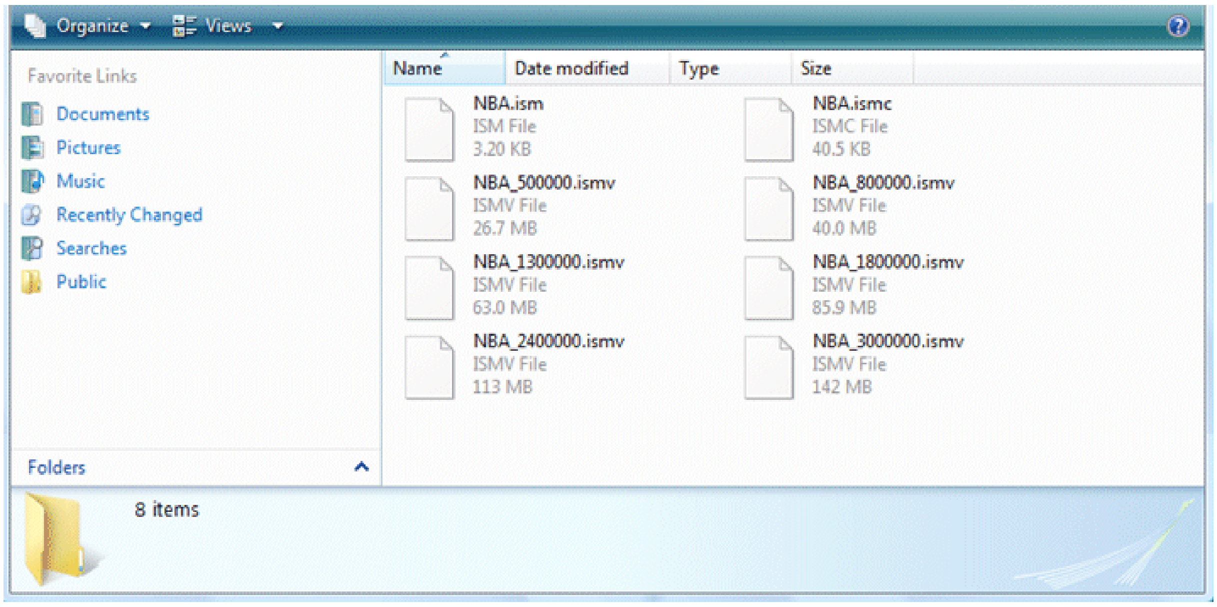
Figure 2.7: Smooth Streaming file format [211]

Movie metadata (moov) generically describes the video resource. Each fragment has it own set of descriptive metadata (moof) and media data (mdat). The MPEG-4 container also includes an index box (mfra) that references where each fragment is located within the MPEG-4 file.

Such a design requires the server to translate URL requests into exact byte range offsets within the MPEG-4 file. Since the server logic does the mapping between requests and locating video segments, clients can request chunks in a more developer-friendly manner like requesting time codes instead of index numbers.

2.4.4 Manifest files

The manifest files are XML-formatted, so the files are extensible [137]. They support multiple hardware profiles like a bit rate targeted for multiple playback devices. The client manifest support Gzip compression [144]. Hence, less bandwidth to download a manifest. Figure 2.8 illustrates the necessary files for presentation.



NBA.ism
ISM File
3.20 KB

NBA.ismc
ISMC File
40.5 KB

NBA_500000.ismv
ISMV File
26.7 MB

NBA_800000.ismv
ISMV File
40.0 MB

NBA_1300000.ismv
ISMV File
63.0 MB

NBA_1800000.ismv
ISMV File
85.9 MB

NBA_2400000.ismv
ISMV File
113 MB

NBA_3000000.ismv
ISMV File
142 MB

8 items

Figure 2.8: Smooth Streaming media assets [211]

.ismv files contains video with or without audio. There is one .ismv file per encoded video bit rate. .isma files contains only audio. .ism is the server manifest file. It describes the relationship between the media tracks, bit rates and files on disk. .ismc is the client manifest file. It describes the available streams to the client, for example the codecs used, bit rates encoded, video resolutions and so on.

2.4.5 Client device

Web browsers with the Silverlight plug-in [74] can play Smooth Streaming media. This plug-in program has support for Smooth Streaming from version 2. This includes parsing of the MPEG-4 file format, the HTTP download, the bit rate switching heuristics and so on.

A client video player initially request the .ismc client manifest file from the server. The manifest includes information about codecs, bit rates, and a list of available chunks. Learning what codecs were used to compress the content, the player can then initialize the correct decoder to playback the video. It also prepares the playback pipeline [211].

A client uses its own analysis of local bandwidth and CPU consumption. For example, the client-side code evaluates chunk download times, buffer fullness and rendered frame rates. The client player then decides whether to request higher or lower bit rates from the server. Clients request fragments in the form of a RESTful [180] URL:

- http://from.somewhere.com/QualityLevels(100000)/Fragments(video=123456)

These example numbers represent the encoded bit rate (100000) and fragment start offset (123456). The client uses these values from the ISMC file.

The server then looks up the bit rate in the corresponding .ism server manifest file. The .ism file has a mapping to a physical .ismv or .isma file on disk. It then reads the appropriate MPEG-4 file and uses the offset value to find the correct video fragment within this file. This is possible using the index reference box (mfra). The fragment is extracted from the video source and transferred to the client as a standalone file. Such files can then be cached at for example CDNs for later requests from other clients.

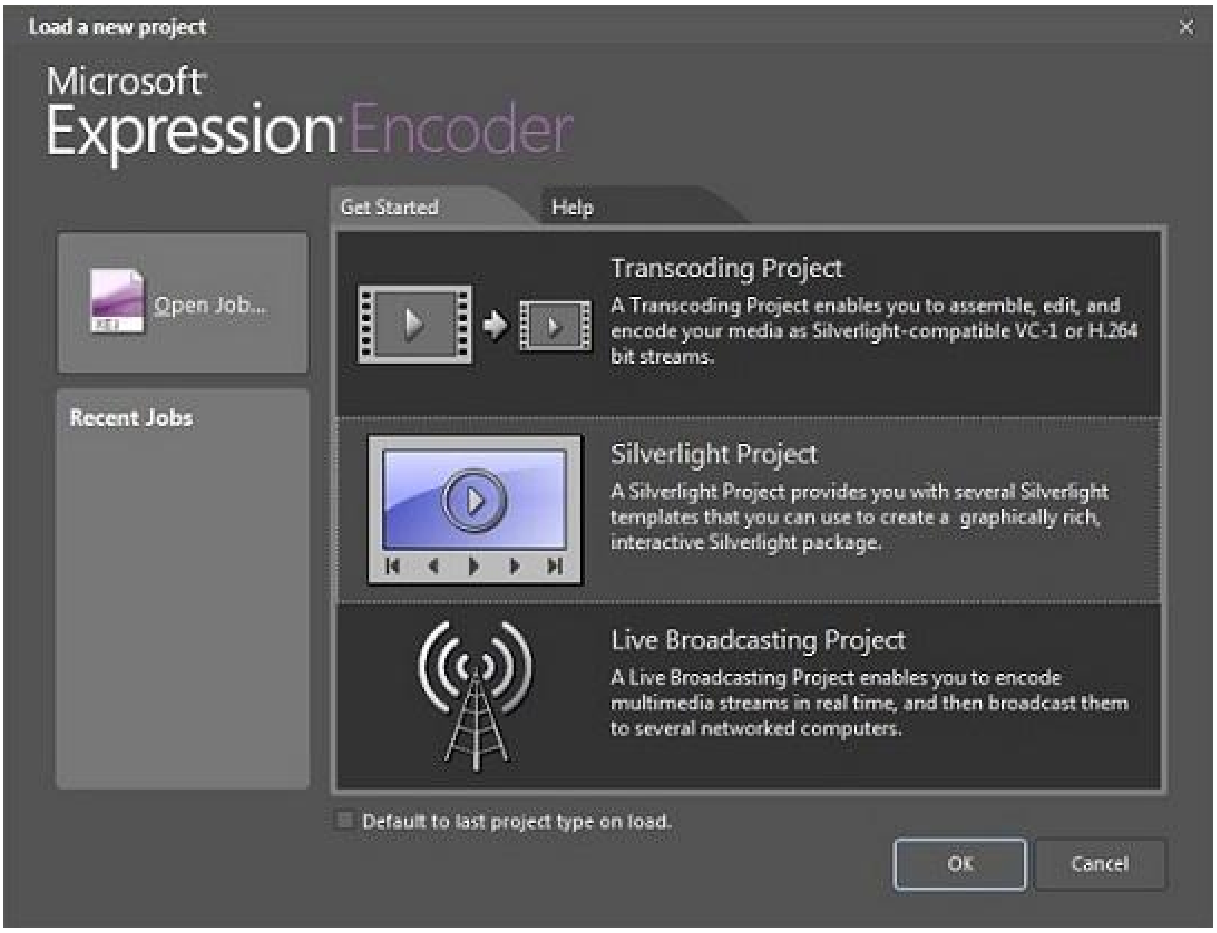
2.4.6 Supported options

Smooth streaming support services similar to what you would expect in for example a DVD or Blu-ray product:

- Text streams
- Multi-language audio tracks
- Alternate video and audio tracks (multiple camera angles, director's commentary and so on)
- Markers/chapters, captions

2.4.7 Developer tools

It is possible to encode Smooth Streaming media files using the Microsoft window based program Expression Encoder 4 Pro [75]. Figure 2.9 illustrates a user interface in the program.



Transcoding Project
A Transcoding Project enables you to assemble, edit, and encode your media as Silverlight-compatible VC-1 or H.264 bit streams.

Silverlight Project
A Silverlight Project provides you with several Silverlight templates that you can use to create a graphically rich, interactive Silverlight package.

Live Broadcasting Project
A Live Broadcasting Project enables you to encode multimedia streams in real time, and then broadcast them to several networked computers.

Figure 2.9: Expression Encoder user interface [75]

With this program you can encode several quality levels of a specific piece of content, with each quality level encoded as its own complete file [75]. Note that this product is no longer available for sale.

For customers who previously purchased this product, it will be supported through its support lifecycle. Expression Encoder 4 remains available for download at no charge [76].

2.4.8 Smooth Streaming takeaway points

Video segments have a typical duration of two seconds. There are one client and one server manifest file. The media is stored and transported using the MPEG-4 container format. Smooth Streaming uses "virtual" segments for better file management. This adaptive streaming technology uses the Silverlight plug-in program for playback.

2.5 MPEG-DASH - Adaptive Streaming standard

The streaming industry needs confidence in a solution that can be adopted for universal deployment. Streaming platforms such as HLS [13, 14, 31, 95], Smooth Streaming [75, 93, 211] and Adobes HTTP Dynamic Streaming (HDS) [5] all uses different manifest and segment formats.Therefore, to receive the content from each server, a device must support its corresponding proprietary client protocol. "A standard for HTTP streaming of multimedia content would allow a standard-based client to stream content from any standard-based server, thereby enabling interoperability between servers and clients of different vendors" [99].

Such needs required an international standard. In april 2009 MPEG issued a Call for Proposal for an HTTP streaming standard. Fifteen full proposals were received by July 2009. MPEG developed the specification with participation of many experts and in collaboration with other standard groups such as 3GPP. "MPEG-DASH became a Draft International Standard in January 2011 and a international standard in November 2011" [117].

2.5.1 MPEG-DASH overview

The technology has the familiar adaptive streaming pattern of breaking content into a sequence of small HTTP-based video segments. The content is made available at a variety of different bit rates. As the content is played back by an MPEG-DASH client, the client automatically selects from the alternative bit rates and downloads the next segment to be played back by the video player. As figure 2.10 illustrates, "the client selects the segment with the highest bit rate possible that can be downloaded in time for play back without causing stalls or rebuffering events in the playback" [117].

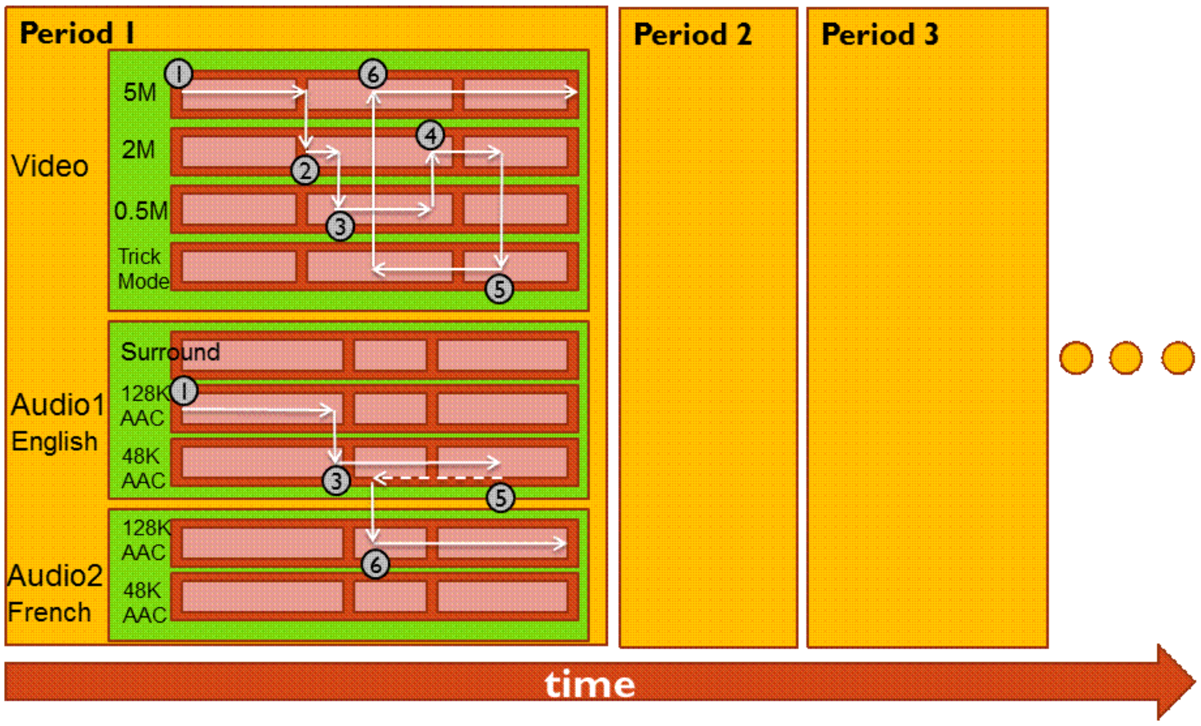


Figure 2.10: MPEG-DASH segments [99]

Note that the figure shows periods of different durations. This is important because of latency issues. If streaming live, you do not want any unnecessary latency delay. MPEG-DASH can be configured to buffer less data before playing than other streaming technologies. "If we want to get closer to live edge, we're able to with one of our clients that has live content. We're generally about three to five seconds behind the live edge" [102].

This is not only a live streaming benefit. A good user experience depends among other things on latency being as little as possible. A video player can be configured to how long the video segments should be. It is possible to start with shorter video segments to get faster start up. Later, the player can be adjust for longer segments by calculating how much buffer time it needs.

Being an adaptive HTTP streaming technology, MPEG-DASH consists of segmented multimedia files and a metadata description file. These are available on a HTTP server.

2.5.2 Manifest file

A Media Presentation Description (MPD) is the MPEG-DASH manifest file. The MPD file describes video segment information like available content, various aspect ratios, different bit rates, indexing of URL addresses and other characteristics. Being a XML [137] document, the MPD file has a hierarchical data model. This model consists of one or multiple periods. A period is an interval of the program along a temporal axis. Each period has a starting time and a duration. Within each period, there is one or multiple adaptation sets. An adaptation set provides information about one or multiple media components and its various encoded alternatives [99]. Figure 2.11 illustrates a MPD with three adaption sets.

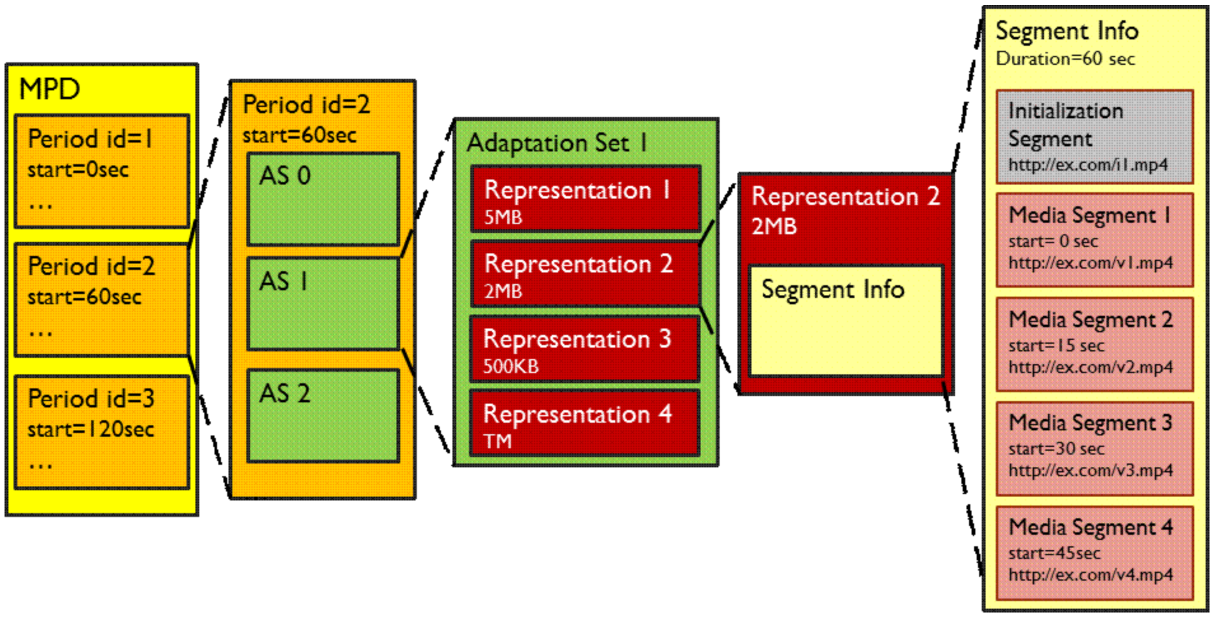


Figure 2.11: MPEG-DASH hierarchical data model [99]

In this figure, adaption set 0 has 4 x 3 content while adaption set 1 has 16 x 9 content. The different aspect ratio [126] representations are within each adaption set. Thus, each representation is a child node of an adaption set. "So all of our 16x9 streams are all in one adaptation set, all of our 4 x 3 streams are in a separate adaptation set. Why? Well, because it's far more jarring for an end user to switch from a wide screen video to a 4x3 than it is to switch from a high quality to low quality within the same size" [102].

Each representation has a segment info child node which contains the playable media. There can be separate segments for video and audio. This structure has several benefits. For example, you might want different languages to choose from. In live sports, you might want to watch a soccer match with the radio announcers instead of the TV announcers.

Another benefit is file space. When the video and audio are multiplexed [166] together, every video file needs to be big enough to also support the audio track. Usually, when switching bandwidth it is because the video quality needs to be changed. "So why do we need to take up that much more file space to have exactly the same audio data in all of our different segments for the video? It makes more sense, it's more efficient to have them separated" [102]. Sync issues [127] may arise when separating audio and video, so this technique should be used carefully.

MPEG-DASH also supports live presentations. For such streaming purposes, the manifest provides segment start and end time availability, approximate media start time and fixed or variable duration of segments.

2.5.3 Segment format

The segmented data contains the actual audio and video bitstreams in single or multiple files. Each media segment is assigned a unique URL (possibly with byte range), an index, explicit or implicit start time and duration. "Each media segment contains at least one Stream Access Point (SAP) [143], which is a random access or "switch to" point in the media stream where decoding can start using only data from that point forward" [99].

It is possible to download a segment in multiple parts. The specification defines a method of signaling sub-segments using a segment index box. The index describes sub-segments and SAP's in the segment by signaling their durations and byte offsets. A MPEG-DASH client can use this information to request sub-segments using partial HTTP GETS. This might be useful under dire network conditions like wireless connections to a mobile device.

The indexing information of a segment can be put in the single box at the beginning of that segment or spread among many indexing boxes in the segment. Different methods of spreading are possible, using combinations of hierarchical or ring-based solutions. "This technique avoids adding a large box at the beginning of the segment and therefore prevents a possible initial download delay" [99].

2.5.4 Switching bit rates

One of the powers and possible pitfalls of MPEG-DASH is user flexibility. A lot need to be considered when building a client player. "As any Web application developer will tell you, there's much more to building a good player than just setting a source URL for the media element" [211].

- The heuristics module that determines when and how to switch bit rates. Elementary stream switching functionality requires the ability to swiftly adapt to changing network conditions while not falling too far behind, but that's often not enough to deliver a great experience.
- What if the user has enough bandwidth, but does not have enough CPU power to consume the high bit rates?
- What happens when the video is paused or hidden in the background (minimized browser window)?
- What if the resolution of the best available video stream is actually larger than the screen resolution, thus wasting bandwidth?
- How large should the download buffer window be?
- How does one ensure seamless rollover to new media assets such as advertisements?

Configuring a client player for switching bit rates is particularly tricky. Available bandwidth has to be calculated. Typically, this begins when downloading the manifest file. Comparing the size of the file with the duration it took to download it, usually gives a good indication of which bit rate is suitable for playback. This calculation needs to be done at intervals during playback for possible changes in network conditions. Such calculations becomes one important metric for determining how to adapt bit rates. Other considerations is also necessary.

"Do we have enough time? How is it performing? Am I dropping frames? How long is my buffer? Is my buffer emptying faster than my buffer is filling? These are things that could be used to determine "Should I switch up or should I switch down? How do I adapt my bit rate?" [102].

A controller logic should decide when the player switches bit rates. However, it is difficult to design the controller to suit all needs. While company A require low latency instead of high quality, company B might want to prioritize high quality instead of fast startup times. The player needs to adapt to different situations.

"So what we did in architecting this was we came up with the idea of a set of rules. And we can add more or less rules as we need and the idea is that we'll run through each set of rules and each rule will make a suggestion that our bit rate is good, our bit rate is too low, our bit rate is too high. And then we have our ABR controller that gets all this feedback and makes a decision, Do I switch?" [102].

Defining too many rules might result in the switching happening too often. Then, it is necessary to have another rule that limits how often the player switches bit rates. Sometimes it best to give a certain bit rate a chance over time to see if it works, instead of switching every time the feedback changes.

2.5.5 Client device

MPEG-DASH uses the HTML5 video tag. Javascript files are included to support the implementation of the specification. To play the multimedia file, the client player uses an URL to request a MPD file. By parsing the MPD, the MPEG-DASH client learns what possible options this particular stream consists of. Such options can be the program length, the various bit rates, location of the media and required DRM. Using this information, the client decides which segments to download. Request are formulated using URL and byte range.

2.5.6 Additional features of MPEG-DASH

- Multiple DRM and common encryption. A segmented multimedia file can use multiple content protection schemes and as long as the client recognizes at least one, it can stream and decode the content. MPEG is developing a common encryption standard. Using this standard, the content can be encrypted once and streamed to clients which support different DRM license management systems.

- Support of different container formats. The specification offers specific guidance to two types of container formats, namely the MPEG-4 file format and the MPEG-2 Transport Stream.

- Interoperability among various systems. MPEG-DASH is media codec agnostic. MPEG-DASH describes the content, tells how the content should be segmented and how it should be described in a manifest, but it doesn't say anything about what codecs you must use. "What this means to us is that when H.265 is ready for the world, we can simply use the same DASH idea and the content that we're delivering happens to be to H.265 now" [102]. Whether your media player is capable of playing for example H.264 or H.265, the MPEG-DASH specification does not care. MPEG-DASH is also agnostic to the underlying application layer protocol. Thus, DASH can be used with any protocol [117].

- MPEG-DASH supports both multiplexed [166] and unmultiplexed encoded content [99]. Separating audio and video is useful in certain situations.

- The duration of segments can be varied. With live streaming, the duration of the next segment can also be signaled with the delivery of the current segment.

- Switching and selectable streams. The MPD provides adequate information to the client for selecting and switching between streams, e.g. selecting one audio stream from different languages, selecting video between different camera angles, selecting the subtitles from provided languages, and dynamically switching between different bit rates of the same video camera.

- A manifest can be fragmented. An MPD can be divided into multiple parts or some of its elements can be externally referenced, enabling downloading a manifest file in multiple steps.

- Other features are ad-insertion, compact manifest, multiple base URLs, clock drift control for live sessions, scalable video coding (SVC) [182], multiview video coding (MVC) [167] support and more.

2.5.7 Scope of the specification

Figure 2.12 illustrates that the specification only defines the MPD and the segment formats marked in red.

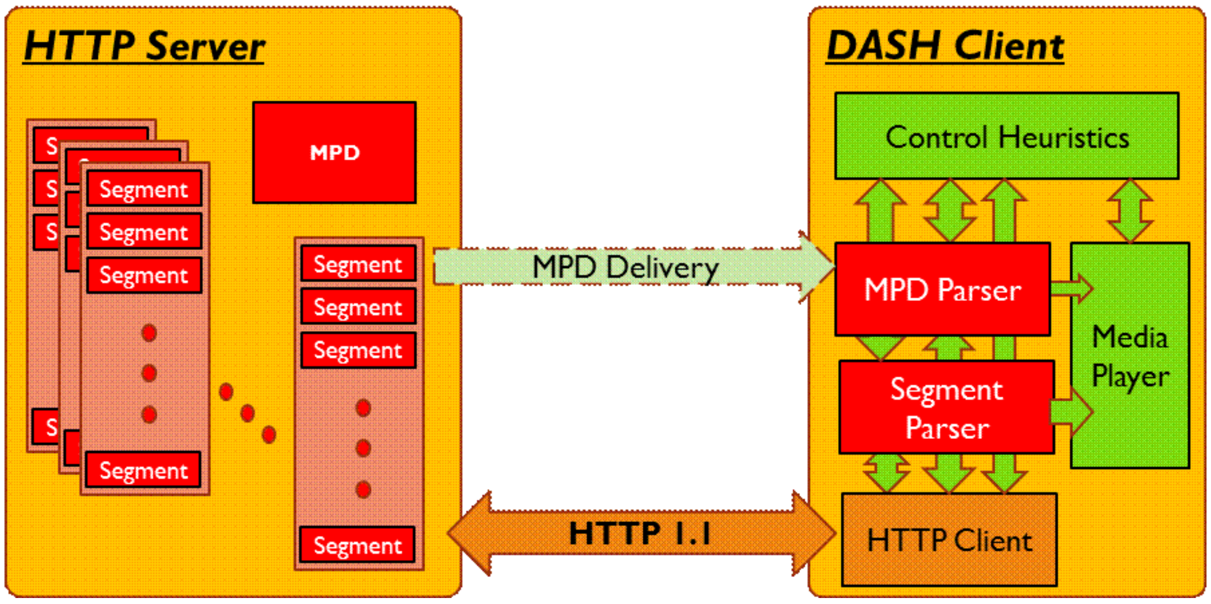


Figure 2.12: Scope of the MPEG-DASH standard [99]

The delivery of the MPD and the media encoding formats containing the segments, are outside of MPEG-DASHs scope. This also applies to the client's behavior for fetching, adaptation heuristics and playing the content.

2.5.8 MPED-DASH in practice

Google, being a competitor to companies like Microsoft and Apple, has shown a particular interest in this standard. They have experimented with supporting MPEG-DASH on the server side. Google can use their products such as YouTube [208] to experiment with implementing the standard, to see if it is good enough performance-wise and scalable. Their web-browser Google Chrome supports MPEG-DASH on the client side. Unfortunately, this is yet a smooth experience. "The implementation of the feature has resulted in video playback being severely degraded by various bugs, such as the video quality options being randomly grayed out and unselectable without multiple refreshes of the page" [117].

2.5.9 Best practice guidelines

The MPEG-DASH specification is a feature-rich standard. However, complexity is always challenged by efficiency. "MPEG-DASH in its original specification tried to be everything to everyone and consequently suffered from excessive ambiguity (a story surely familiar to anyone acquainted with HTML5 Video)" [210]. To solve these issues, the MPEG committee are developing new parts to add to the core specification. Validation tools and best practice guidelines will hopefully improve the implementation of the specification. Encryption and authentication methods for segmented multimedia are also being developed.

The specification defines different profiles addressing different types of applications. Each profile defines a set of constraints limiting the MPD and segment formats to a subset of the entire specification. Therefore, a DASH client conforming to a specific profile is only required to support those required features and not the entire specification [99].

2.5.10 Dash Industry Forum and DASH264

Dash Industry Forum is a non-profit industry association established to catalyze the adoption of MPEG-DASH. Membership includes many large media companies including Microsoft, Qualcomm, Ericsson, Adobe, Sony, Cisco, Intel and Akamai [28]. The forum has published a draft of its DASH264 implementation guidelines. The main focus of the draft is to establish a well-defined set of interoperability constraints. This includes support for the industry standard H.264 video codec. "The vast majority of video on the web is in H.264, so rather than dealing with a spec that can handle anything for anyone at anytime, this specification is about how to handle MPEG-DASH specifically within H.264" [30].

Other essential requirements is support for the High-Efficiency Advanced Audio Coding (HE-AAC) [147] v2 audio codec, ISO base media file format [156], Society of Motion Picture and Television Engineers Timed Text (SMPTE-TT) [191] subtitle format and MPEG Common Encryption for content protection. "The Common Encryption element is particularly interesting because it enables competing DRM technologies such as Microsoft PlayReady [73], Adobe Access [3] and Widevine [38] to be used inclusively without locking customers into a particular digital store" [210]. Such implementation constraint details are essential for the industry to adopt MPEG-DASH.

2.5.11 Media Source Extensions API and dash.js

Dash Industry Forum are also collaborating to build an open source Javascript library named dash.js [28] that plays back MPEG-DASH content. Some of the goals of the initiative being:

- Codec agnostic
- Multiple segment encapsulation types
- Multiple content protection schema
- Simple substitution of switching rules, loading methods and fail-over logic
- H.264 for video and HE-AAC for audio
- Base Media File Format [156] video segments that contains the timing, structure, and media information for timed sequences of media data.
- time-aligned, closed-GOP [143] segments whose duration does not vary more than 25 percent from the mean.
- CENC DRM [133] support

To achieve this, dash.js leverages the Media Source Extensions API set as defined by the W3C [25]. The specification being a collaboration of major streaming companies. Technicians from companies like Google, Microsoft and Netflix are editing the specification. They have also developed a video player where the different companies can test their own media streams. "We're building this in such a way so that people who have content in DASH or people who have DRMs or people who have audio codecs or whatever other - all the members of the consortium can get together and we have one place to test our content and say, "Yes. It works. It's compliant." That's the idea of the reference player" [102].

This draft specification extends HTMLMediaElement to allow JavaScript to generate media streams for playback. Rather than require that each browser support MPEG-DASH directly, the Media Source Extensions expose an interface to the HTML Media element [87]. This allows for ad-hoc addition of media segments. Using such a technique, client-side scripts can parse manifests, request segments and apply switching logic while leaving the video decoding and rendering to the browser. "The combination is a flexible and powerful platform on which to build cross-platform media players" [28].

2.5.12 Web browser support

Currently, it seems only Google's Chrome browser supports the Media Source extension API. Testing the downloadable demo in for example Safari or Microsoft Explorer gives this message: "Your browser does not support the MediaSource API. Please try another browser, such as Chrome". "Throughout the different channels of Chrome, there are different levels of support. The release version has less support than Canary, one still in the Chromium channel, the open-source project that feeds code to Chrome" [102]. Firefox's wrote in their developer webpage that "Firefox 21 includes an implementation of MPEG-DASH for HTML5 WebM video which is turned off by default ... Firefox 23 removed support for MPEG-DASH for HTML5 WebM video. It will be replaced by an implementation of the Media Source Extensions API [25] which will allow support for MPEG-DASH via Javascript libraries such as dash.js" [82]. Eventually multiple browsers might embrace this technology.

2.5.13 MPEG-DASH takeaway points

MPEG-DASH is an international standard. The specification is feature-rich and flexible, but the implementation of MPEG-DASH needs to be improved. A collaboration of companies are working to make MPEG-DASH an adaptive streaming technology that enables interoperability between servers and clients of different vendors.

2.6 Comparing adaptive streaming technologies

We have researched three different approaches to adaptive streaming. Before we start comparing them, we want to emphasize that Adobe's HDS [5] is a good alternative. It is a proprietary technology that uses a plug-in program in a web browser like Microsoft's Smooth Streaming. One of HDS main characteristics is an aggressive adaptive player. Small changes in CPU or network traffic is enough to change bit rate. There are pros and cons to this approach. It might be disturbing to watch a video that often changes quality. On the other hand, end users usually want to watch the best quality available on their receiving device.

Next, we will compare Smooth Streaming with HLS. They are both proprietary technologies. We will use this knowledge to compare proprietary technologies with the international standard MPEG-DASH. On the basis of these comparisons, we will choose an adaptive streaming technology for building our prototype.

2.6.1 Smooth Streaming compared with HLS

There used to be several differences comparing Smooth Streaming and HLS. New features in later versions of HLS [18] has reduced some of these differences.

One of HLS main characteristics is multiple downloads of manifest files. For live streaming, Apple's protocol uses a regularly updated "moving window" index file that tells the client player which chunks are available for download. The index file is updated by removing media URIs from the manifest as new new media files are created and made available [18]. This requires downloading the index file multiple times to get the latest updates. For VOD purposes, HLS needs to download an alternate index files to change to a not previously used bit rate.

Smooth Streaming's approach is to create an index files on both the client side and the server side. Hence, the server index file handles the reference between requests and storage. The client does not have to repeatedly download an updated index file. It is also possible to request segments with timecodes instead of indexes. However, this adds complexity to the server logic.

This leads to a second difference. Since live streaming on HLS require multiple downloads of an update index file, it is desirable to run HLS with longer duration chunks. Thus, minimizing the number of index files downloaded. This is one of the reasons the recommended segment duration using HLS is ten seconds. Smooth Streaming chunk size is by default two seconds.

The "wire format" of the chunks are different. Both formats use H.264 video encoding and AAC audio encoding. However, HLS uses MPEG-2 TS files while Smooth Streaming uses "fragmented" MPEG-4 files. Both formats have some advantages. MPEG-2 TS files have a large installed analysis toolset and have pre-defined signaling mechanisms. Fragmented MPEG-4 files are flexible and can easily accommodate all kinds of data.

There was a difference in infrastructure support for the two formats. Microsoft has included Smooth Streaming server support in their IIS server [154]. This type of server stores the Smooth Streaming media container in an aggregate format, meaning a small number of files "virtually" containing the fragmented chunks. Microsoft positioned their aggregate format as a strength of Smooth Streaming over HLS. One contiguous file can arguably allow files to be managed more effectively than thousands of individual chunks for a given video resource.

Since both Microsoft and MPEG-DASH offer byte-range support, then Apple has included this functionality as well. New in iOS 5, you can now specify a media segment as a byte range (subrange) of a larger URL. This allows you to consolidate your media segments into larger files or a single large file. "The primary benefit of this is when a client is playing your media, rather than downloading each successive segment file from a different location, it is actually walking through a larger file in sequence" [18].

While this benefits the content server side, it is not necessarily the case on the receiving web server or CDN side. Each segmented chunk is requested and delivered individually. The downside of having one large contiguous file is that it requires more complex logic for the server when clients request a media stream.

Both adaptive streaming technologies will probably do the job. Now, we will consider the default durations of the two different adaptive streaming protocols. Smooth Streaming creates chunks of approximately two seconds while HLS recommended usage is ten second chunks.

Our goal is to dig into existing video archives and create new interesting combinations of video. If our system returns media that is very specific, we only need short durations of the media resource. A duration of two seconds more or less does the editing for us. If our system provides a playlist with editing options to fine tune the segment, we need video handles. A durations of ten seconds might be a perfect editing length. There will also be situations where the requested media is divided among several chunks. Meaning, we need a little bit of chunk A, B or a larger offset. Note that the default duration of HLS is ten seconds, but it is possible to playback HLS segments of two seconds as well. If each segment includes an i-frame, the HLS video player accepts it. However, if the HLS video player is optimized for segments of ten seconds duration, then playback of shorter segments might cause a bad user experience. "Here's why: if you've got live content being delivered through a CDN, there will be propogation delays, and for this content to make it all the way out to the edge nodes on the CDN it will be variable. In addition, if the client is fetching the data over the cellular network there will be higher latencies. Both of these factors make it much more likely you'll encounter a stall if you use a small target duration" [17].

2.6.2 Proprietary lifecycle

We also need to consider which technology will be most flexible in the future. Smooth Streaming uses MPEG-4, which is a newer and more flexible encoding standard than HLS's MPEG-2 TS format. They both support H.264 video encoding and AAC audio encoding, so the encoding quality by data size ratio should be the same.

Smooth Streaming uses the Silverlight plug-in program, where support for the latest version, Silverlight 5 is to expire 12.10.2021 [71]. Because Microsoft never publicly has committed to a Silverlight 6, "many believe the platform has met its maker" [69]. Netflix is one of the major streaming companies that uses Silverlight for streaming video. They said in April 2013 they would abandon Microsoft's Silverlight media player plug-in for Windows and OS X in-browser video streaming, and replace it with a trio of HTML5 extensions [69]. The company's director of engineering, Anthony Park, and director of streaming standards, Mark Watson, defended the switch to HTM5 with well-worn arguments, including distrust of plug-ins on security grounds, but they also pointed out that browsers, even on the desktop, are shifting to a no-plug-in-model [69]. There is still some time until 2021, but Smooth Streaming might not be the technology of choice for the future. However, Netflix has announced they will use the next eight years to find a a replacement for Silverlight [69], suggesting it might take some time to develop a satisfying alternative adaptive streaming technology.

HLS is to our knowledge still going strong. Apple has recommended HLS as an Internet Draft to the IETF. Today, HLS works fine with Apple products. For other client platforms it is possible to use a Flash plug-in player as a fallback. Desktop browsers like Chrome and Firefox can play HLS streams using a player like the JWPlayer [207].

2.6.3 Mobile devices

This solution does not work for mobile devices without native support for HLS. Simply because they generally cannot use Flash. While earlier versions of Android supported Flash, it was discontinued in Android 4.1, meaning future versions of the OS cannot play back HLS streams natively [207]. Since HLS not yet is an accepted standard by the IETF, other companies like Google and Microsoft might be reluctant to embrace this technology. "Until either a new non-proprietary standard is adopted, or each of the mobile platforms implement native HLS support, it looks like streaming to devices like Android will remain a challenge" [207].

2.6.4 International standard

MPEG-DASH provides an international standard that combines the best features of Apple's HLS, Adobe's HDS and Microsoft Smooth Streaming into a modern format. This standard is flexible enough to handle multiple addressing schemes, multi-track video and audio, timed metadata and advertising insertion [28]. MPEG-DASH supports both the container format MPEG-4 that Smooth Streaming is using, and the MPEG-2 Transport Stream that HLS is using. It allows for fragmented MPD's, similar to HLS's multiple index files. It supports switching and selectable streams, also possible in Smooth Streaming. The duration of segments can be varied. MPEG-DASH is already an international standard. It sounds like the perfect choice, at least in theory.

2.6.5 Comparing HLS and MPEG-DASH flexibilty

At the 2013 Streaming Media East conference, Jeff Tapper at Digital Primates argues why MPEG-DASH is the next generation adaptive streaming technology. Following is a transcript of him comparing HLS and MPEG-DASH flexibility:

"While HLS is supported natively in Safari, as a developer we have no control over what happens to that stream. We simply tell Safari, "Hey, here's a video. Go play it," and it makes every decision that needs to be made about that video. What quality you're watching? You have no control; they decide. What are the adaptive bit rates? They're built-in logic. If you want to have any control over what's happening with that stream you really can't. Whereas with the media source extensions, we as developers decide what bits we hand to the API when we're ready to hand it to it. So if as we deliver one piece of content, if we realize, "You know what? This one's not playing well. We're dropping a lot of frames. It takes longer to download the segment than it does to play out the segment, "I can switch to a lower bit rate. Or the inverse: Maybe I can decide, "Hey, I've got a lot of extra room: I want to deliver a higher-quality bit rate." And we can determine, as we need to, exactly which segment to hand to the media buffer, which ones we want to download, which ones we want to play [102]".

Tapper is arguing for client flexibility. While HLS has a preset of rules for changing bit rates, MPEG-DASH lets each client player define its own set of rules. Smooth Streaming is similar to MPEG-DASH in that it also supports client side switching logic.

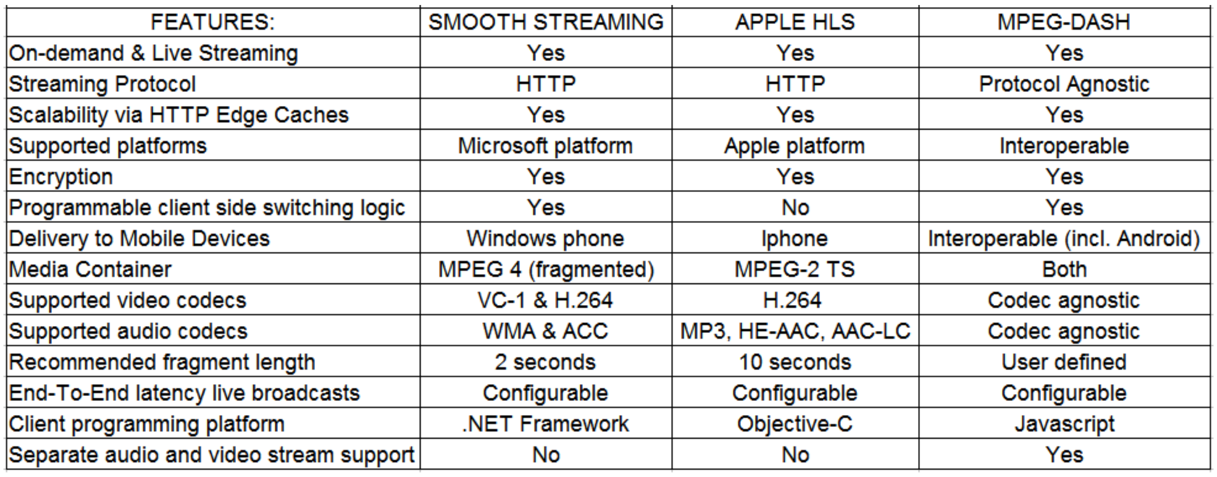
2.6.6 Latency live broadcasts

HLS recommends a fixed length of ten second segments, while Smooth Streaming by default has segments of two seconds duration. MPEG-DASH is flexible in dynamically changing the segment lengths. This is important because of latency issues. If streaming live, you do not want any unnecessary latency delay. MPEG-DASH can be configured to buffer less data before playing than other streaming technologies. For example, streaming segments of short duration first, then segments of longer durations.

Before deciding on segment duration it is important to consider the end user experience. Apple suggest three full segments in the buffer before you start playing. This means a live start up latency of ten seconds x three segments = thirty seconds. This ensures a smooth playback. MPEG-DASH can use the same suggestion as Apple, or be optimized for less latency. Smooth Streaming also supports latency optimization.

2.6.7 Choosing adaptive streaming technology for our system

Table 2.1 illustrates a comparison of the three adaptive streaming technologies.



FEATURES:	SMOOTH STREAMING	APPLE HLS	MPEG-DASH
On-demand & Live Streaming	Yes	Yes	Yes
Streaming Protocol	HTTP	HTTP	Protocol Agnostic
Scalability via HTTP Edge Caches	Yes	Yes	Yes
Supported platforms	Microsoft platform	Apple platform	Interoperable
Encryption	Yes	Yes	Yes
Programmable client side switching logic	Yes	No	Yes
Delivery to Mobile Devices	Windows phone	Iphone	Interoperable (incl. Android)
Media Container	MPEG 4 (fragmented)	MPEG-2 TS	Both
Supported video codecs	VC-1 & H.264	H.264	Codec agnostic
Supported audio codecs	WMA & ACC	MP3, HE-AAC, AAC-LC	Codec agnostic
Recommended fragment length	2 seconds	10 seconds	User defined
End-To-End latency live broadcasts	Configurable	Configurable	Configurable
Client programming platform	.NET Framework	Objective-C	Javascript
Separate audio and video stream support	No	No	Yes

Table 2.1: Comparison of adaptive streaming technologies [70]

We have earlier compared HLS with Smooth Streaming. It seems both technologies offer good streaming capabilities within their proprietary domains. MPEG-DASH offer interoperability between vendors and flexibility to configure how the playback works as well. DASH264 and the media source extension is an important step in defining interoperability guidelines for the standard. However, further research is needed to make this an industry implementation of choice.

We need a well-defined streaming technology for our prototype. HLS are supported on desktops, tablets and smart phones. Even though the Safari player makes a lot of predefined choices regarding video quality, it is optimized for the platforms it supports. Apple also provide good developer guidelines for implementing this technology and offer specialized software to create the adaptive video. If it is necessary to play back the system on a non-Apple platform, it is possible to fall back to a Flash player like JWPlayer or use a wrapper format to convert the video to be playable via for example the Silverlight plug-in.

Using HLS it is possible to either create individually segments or segments as fragments in a contiguous file. A segmented movie are a lot of individual video segments. Especially since the movie will be segmented in multiple bit rates as well. Even though file management can become an issue, it is easier to rearrange individual video segments than virtually segmented files. To build the prototype, we need to pinpoint metadata information to specific video segments. Using individual segments makes this job easier.

We have decided to use HLS for our prototype. Next, we will research different approaches for searching video resources.

2.7 Searching video resources

We have compared different adaptive streaming technologies and decided upon HLS to transfer video segments to client devices. Using such a technology requires a full length movie to be divided into thousands of video segments. These segments can be played back individually or in sequence using a manifest playlist file.

The next challenge is to be able to search for a specific segment range within this large collection of segments. It seems logical to index each segment to separate them from each other. For example, a particular scene in a movie can have an index range of 1456 to 1556. This means 100 segments needs to be streamed in sequence to the client to be able to watch this particular movie scene. We need a search engine that is able to locate these indexes. We can calculate the location of these indexes given a fixed duration of each segment. For example, if each segment has a duration of two seconds, then the following calculation should tell us the location of the scene within the movie:

```
Start index: 1456x2 = 2912 seconds into the movie (in point)
End index: 1556x2 = 3112 seconds into of the movie (out point)
Duration: 3112 - 2912 = 200 seconds
Inpoint converted to minutes: 2912 / 60 = 48 minutes and 32 seconds
Outpoint converted to minutes: 3112 / 60 = 51 minutes and 52 seconds
Duration converted to minutes: 200 / 60 = 3 minutes and 20 seconds
```

Alternatively, we can fast forward a movie to our in point and out point, and check if the movie scene corresponds to our calculation. This approach might be good for testing, but the process is time-consuming.

In addition to indexes and calculations, we need to collect information about specific scenes in a movie to make them searchable. There are two main approaches of achieving this. We can analyze the video content, or we can use metadata describing the video content. In the following sections we will evaluate both approaches. Next, we will research making video summaries using image analysis.

2.8 Video summary using image analysis

"In computer vision this is the task of finding and identifying objects in an image or video sequence. Humans recognize a multitude of objects in images with little effort, despite the fact that the image of the objects may vary somewhat in different view points, in many different sizes / scale or even when they are translated or rotated. Objects can even be recognized when they are partially obstructed from view. This task is still a challenge for computer vision systems" [169].

Here are a few appearance-based image recognition examples. This can be changes in lighting, color, viewing direction, size and shape. Note that these parameters look different under varying conditions [169].

- Edge matching. This technique compares edges in templates and images to find possible matches.
- Divide-and-Conquer search. It consider all positions in an image, evaluates which parts of the image to investigate further, then divides this part of the image to find a possible match.
- Greyscale matching. This technique looks for a specific luma [159] between black and white. Can also be applied for color. For example, "the woman in the red dress in the first Matrix movie [60] is the only character in the program's city street setting wearing a vibrant shade of red. (In contrast, other pedestrians' attire is mostly black.)" [116].
- Gradient matching. Similar to grayscale matching, this technique compares image gradients.
- Histograms of receptive field responses.
- Large model-bases. A collection of geometric models of the object or a specific image is used to recognize an object.

We will present a few approaches using such techniques to create video summaries from a video resource.

2.9 Video Manga

Today, many blockbuster movies are based on comic books. Among 2013's highest-grossing films are the stories of a wisecracking, armour-wearing billionaire (Iron Man 3, #1), a clean-cut alien in tights (Man of Steel, #4), a beefcake Norse god (Thor: The Dark World, #11), and an ex-lumberjack with claws (The Wolverine, #20) [91]. Another example is the movie 300 about a force of 300 Spartan soldiers fighting the Persians at at Thermopylae in 480 B.C. [43]. This movie adaption aimed to look the same as the comic book by Frank Miller. "The film was directed by Zack Snyder, while Miller served as executive producer and consultant. It was filmed mostly with a super-imposition chroma key technique, to help replicate the imagery of the original comic book" [120]. With such adaptions, the comic book looks like a video summary or storyboard [187] of the movie.

2.9.1 Video Manga overview

This is what Video Manga (VM) summaries aims to do with video in general. By extracting certain keyframes [143] throughout a movie, it is possible to present a short summary of the movie. "The result is a compact and visually pleasing summary that captures semantically important events, and is suitable for printing or Web access" [111].

Any collection of video keyframes can be considered a summary. For example, we can extract one i-frame image from every video segment in a movie. This will give us a collection of thousands of still frames. We will later use such a technique to create thumbnails for our system. See subsection 3.8.2 for details. However, thousands of still frames are too many to create a satisfactory video summary. For example, if a video scene portraying a landscape has a long duration, then still frames from many video segments in sequence will be almost identical images.

To reduce the amount of data and to avoid redundancy in picture information, VM summaries uses a ranking system to decide which keyframes should be included in the summary. "Using the importance measure, keyframes are selected and resized to reflect their importance scores, such that the most important are largest." [111].

2.9.2 Importance measure

VM summaries incorporates multiple sources of information in the importance measure. Low-level automatic analysis [34] are used to search for different kinds of video shots:

- Titles
- Slides or other visual aids
- Close-ups of humans
- Long shots of audiences.

These methods can be used equally well to find anchor shots, reporter "talking heads," and graphics in a news broadcast. "This lets us emphasize important images such as human figures and deemphasize less important images such as long shots. We also perform optical character recognition on image text that appears in the source video or in synchronized presentation graphics. This gives us text for automatically captioning our summaries (though equivalent text could be found by other means such as closed captions or speech recognition)" [111].

2.9.3 Selecting keyframes

One important feature of a VM video summary is to avoid similar images in their presentation. Each frame in the video resource is assigned to a unique cluster. Using a tree structure, all frames becomes leaf nodes under their assigned category. "Once clusters are determined, we can segment the video by determining to which cluster the frames of a contiguous segment belong. This avoid all the pitfalls of on-the-fly shot detection, as it does not generate spurious frames due to motion, pans, or fades. In particular, it does not rely on manually-tuned thresholds for good performance, and thus works well across a variety of video genres" [111].

Combining clustering with an importance measure, VM applications calculate an importance score for the different keyframes. Segments with an importance score higher than a threshold are selected to generate a pictorial summary reminiscent of a comic book or Japanese manga" [111]. For example, a VM summary of a concert will look something like the illustration in figure 2.13

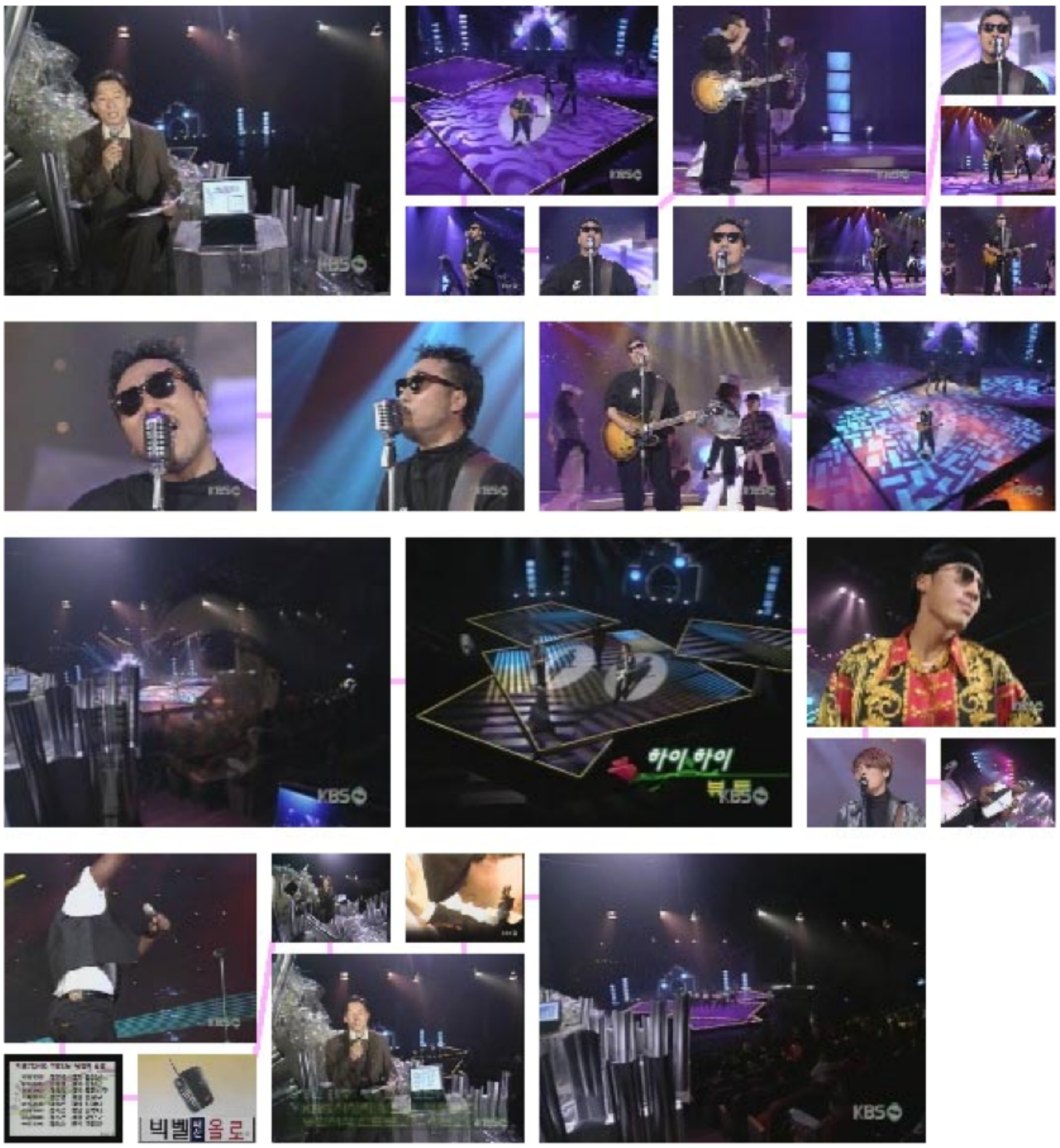


Figure 2.13: Video Manga example [111]

2.9.4 Adapting to our needs

The FX Palo Alto Laboratory has used these summarization techniques on genres such as commercials, movies, and conference videos [111]. They have built a system that automatically creates an interactive summary for each VOD. The system is interactive, and a user can watch different parts of the VM summary. Once an interesting segment has been identified, clicking on its keyframe starts video playback from the beginning of that segment [111].

VM summaries can be enhanced with text captions. For example, conference videos summaries can be presented with meeting minutes displayed as captions. If the meeting minutes were taken on a computer, they can be automatically timestamped and aligned with the video. "Other video summarization method use text, typically from closed-caption subtitles or manual transcriptions. However, most informal videos in our domain are not closed-captioned and transcripts are not likely to be provided" [111].

This is an interesting workflow that we can learn from and use in our system. When searching for specific video segments in our video archive, we can create video summaries of segment ranges. Using thumbnails representing these video segment ranges, an end user can navigate and choose which video segments to playback. However, searching based on image information like close-ups of humans or visual aids is to general to find specific scenes within a movie. On the other hand, textual enhancements like closed-caption subtitles or manual transcripts might be helpful for our system. See section 2.16 for details. Next, we will look into recognizing objects in motion.

2.10 Video Synopsis

Video Synopsis (VS) is an approach to create a short video summary of a long video recording [200]. This technology tracks movement within a recorded video.

"This can be a person, car or an animal passing the camera's field of view. The process begins by detecting and tracking objects of interest. Each object is represented as a "tube" in "space-time" of all video frames. Objects are detected and stored in a database in approximately real time" [21].

Hence, an operator can later playback these activities without watching the entire recording. "Following a request to summarize a time period, all objects from the desired time are extracted from the database, and are shifted in time to create a much shorter video containing maximum activity" [21].

2.10.1 Video tracking

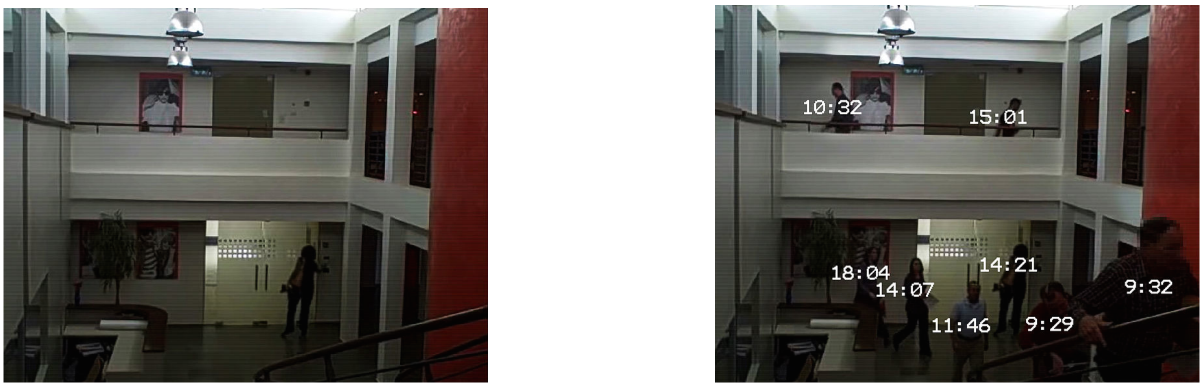
Video tracking is the process of locating a moving object (or multiple objects) over time using a camera. "The objective is to associate target objects in consecutive video frames" [201]. This introduces a lot of challenges. The association can be difficult when the objects are moving fast relative to the frame rate [140]. Another situation is when the tracked object changes orientation over time. Adding further to the complexity is the possible need to use object recognition techniques for tracking [201].

2.10.2 Algorithms

To perform video tracking an algorithm analyzes sequential video frames and outputs the movement of targets between the frames. There are two major components of a visual tracking system, being target representation and localization. These methods give a variety of tools for identifying the moving object. For example, blob tracking identifies human movement because a person's profile changes dynamically. Other techniques include kernel-based tracking, contour tracking and visual feature matching. "Filtering and data association can be used to incorporate prior information about the scene or object, dealing with object dynamics, and evaluation of different hypotheses" [201].

2.10.3 Video surveillance

Such a technology is typically used for video surveillance. A static camera monitors the same area for several hours. "It is reported, for example, that, in London alone, there are millions of surveillance cameras covering the city streets, each camera recording 24 hours / day. Most surveillance video is therefore never watched or examined. Video synopsis aims at taking a step toward sorting through video for summary and indexing, and is especially beneficial for surveillance cameras and webcams" [90]. An example using VS is illustrated in figure 2.14.



(a) Before (b) After

Figure 2.14: VideoSynopsis before and after example [200]

Most of the time there is no activity. The VS technology can be used to filter away such recordings. This is far less costly and time-consuming than sorting through a collection of raw video recordings. "Each movement or object can then be played back or mixed together in a freeze frame. The synopsis is made possible by simultaneously presenting multiple objects and activities that have occurred at different times" [21]. Hence, somebody investigating something that happened in front of the camera's field of view, will get feedback on relevant objects of interest. Later on, the investigator can access all video regarding each object.

2.10.4 Adapting to our needs

It is impressive how this technology allows effective access to recorded surveillance video and enable an end user to find and display an event in only a few minutes [21]. VS is an efficient way to gather surveillance information. The primary usage of this technology is to compute movement from a static camera monitor. However, our test data will be movies that have dynamic content. Tracking movement such as a car chase within a movie can be interesting, but it is hard to differentiate what is interesting movement and what is not. A movie also consists of different scenes that usually appear without the constraints of time and space. This technology seems most practical in situations where such constraints matter.

We have described video tracking using image analysis. Alternatively, it is possible to track objects using network sensor technology. Next, we will present a system that tracks objects using such a technology.

2.11 Bagadus

A team of researchers in Norway has made a prototype of a sports analytics application using soccer as a case study. Bagadus [36, 41, 92, 101, 108]. integrates a sensor system, a soccer analytics annotations system and a video processing system using a video camera array. This prototype is currently installed at Alfheim Stadium in Norway. Using this system, it is possible to follow and zoom in on particular player(s). An overview of the system is illustrated in figure 2.15

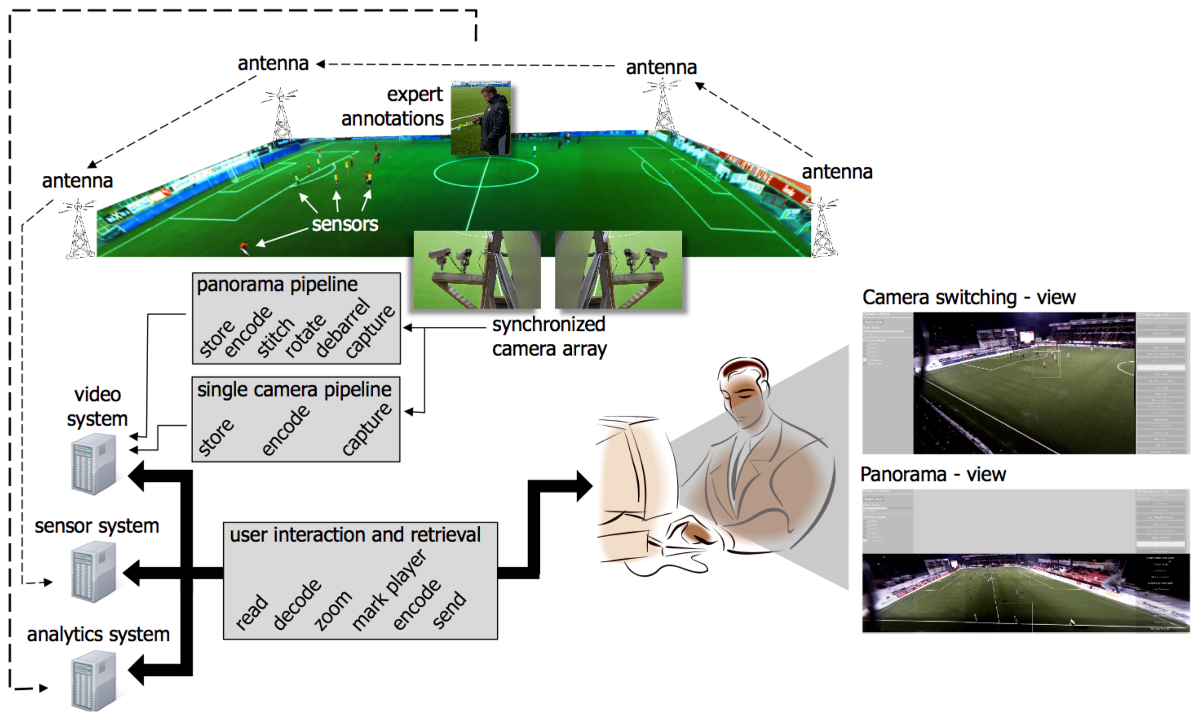


Figure 2.15: Bagadus system overview [41]

2.11.1 Sensor system

This system uses a sensor technology developed by ZXY Sport Tracking [214]. Players wear a data-chip with sensors that sends signals to antennas located around the perimeter of the pitch. "The sensor data is then stored in a relational database system. Based on these sensor data, statistics like total length ran, number of sprints of a given speed, foot frequency, heart rate, etc. can be queried for, in addition, to the exact position of all players at all times" [41]. Bagadus can also use this sensor system position information to extract videos of for example particular players.

2.11.2 Adapting to our needs

The Bagadus [36, 41, 92, 101, 108] example has shown us that is possible to focus and extract a specific topic from a video resource. For example, all events when a striker is in proximity to the opponents goal. This is possible since the sensors provide metadata about the game and the players positions. It seems rich metadata is useful when searching through a video resource.

2.12 Video summary using metadata

We have researched a few approaches to video summaries using image analysis. From the Video Manga technology, we learned that we can access different scenes in a movie by choosing a thumbnail in a storyboard. This technology relies on picture information like close-ups of humans or visual aids to make a selection of scenes. Given that movies consists of different scenes that usually appear without the constraints of time and space, this might give unexpected results. From the Video Synopsis technology, we learned that it is possible to search a large video resource, and display any event in only a few minutes. However, this technology is mainly intended for surveillance using a static camera. We also learned about different object recognition techniques that can be applied to movies. From the Bagadus example, we learned that it is possible to gather metadata from moving objects. Querying this metadata, makes it possible to extract a specific topic from a video resource.

We have researched some video summary technologies that analyses video internally. They use complex mechanisms to evaluate which frames in a video resource to include in their summaries. These metrics might not be the same as what an end-user is looking for in a system about movies. The main purpose of our system is to deliver a specific scene that the user want to playback, or include in a playlist. For example, image analyzing moving objects like a moving car or a gun firing a bullet, can only be applied for specific genres and situations in movies.

Another approach that can be applied to movies in general, is to use text based keywords to retrieve segment ranges within a movie. This requires a search engine and a mapping between descriptive metadata and video segments. Next, we will research a system that uses metadata about a video resource to create video summaries. This system can be used for several purposes.

2.13 Davvi

A collaboration of scientists from the universities in Oslo and Tromsø in Norway has developed a system for mapping metadata to a video resource. This end-to-end prototype is called Davvi [63, 64]. It can be used to create highlights from soccer matches on the fly.

2.13.1 Davvi overview

Davvi is a complex system delivering an end-to-end solution for querying video content and presenting the search result for an end user. An overview of the system is illustrated in figure 2.16

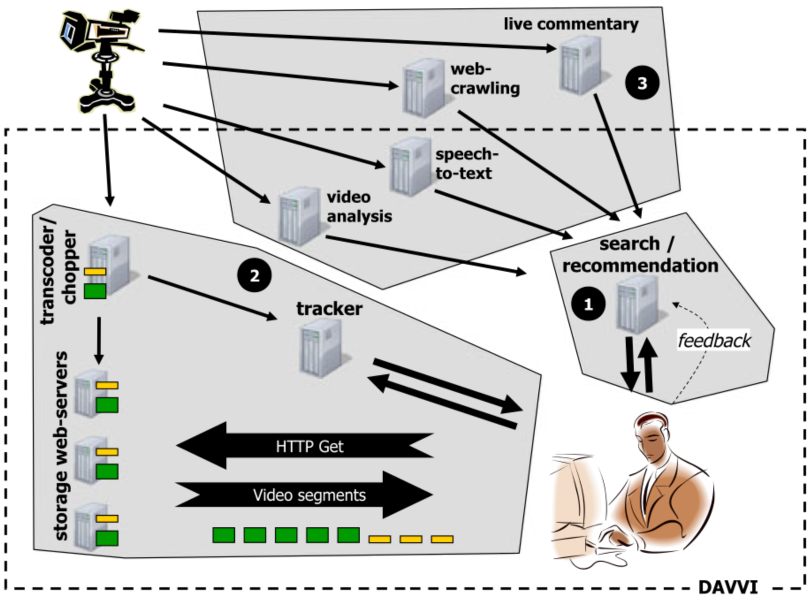


Figure 2.16: Davvi system overview [63]

This system uses existing technologies like the Solr search engine [11], Apache web-servers [10] and FFmpeg coding tools [33]. "To follow good engineering practice, we do not reinvent every single aspect of a new architecture. Instead, we integrate and enhance existing solutions if possible, and then demonstrate novelty through the combination" [63].

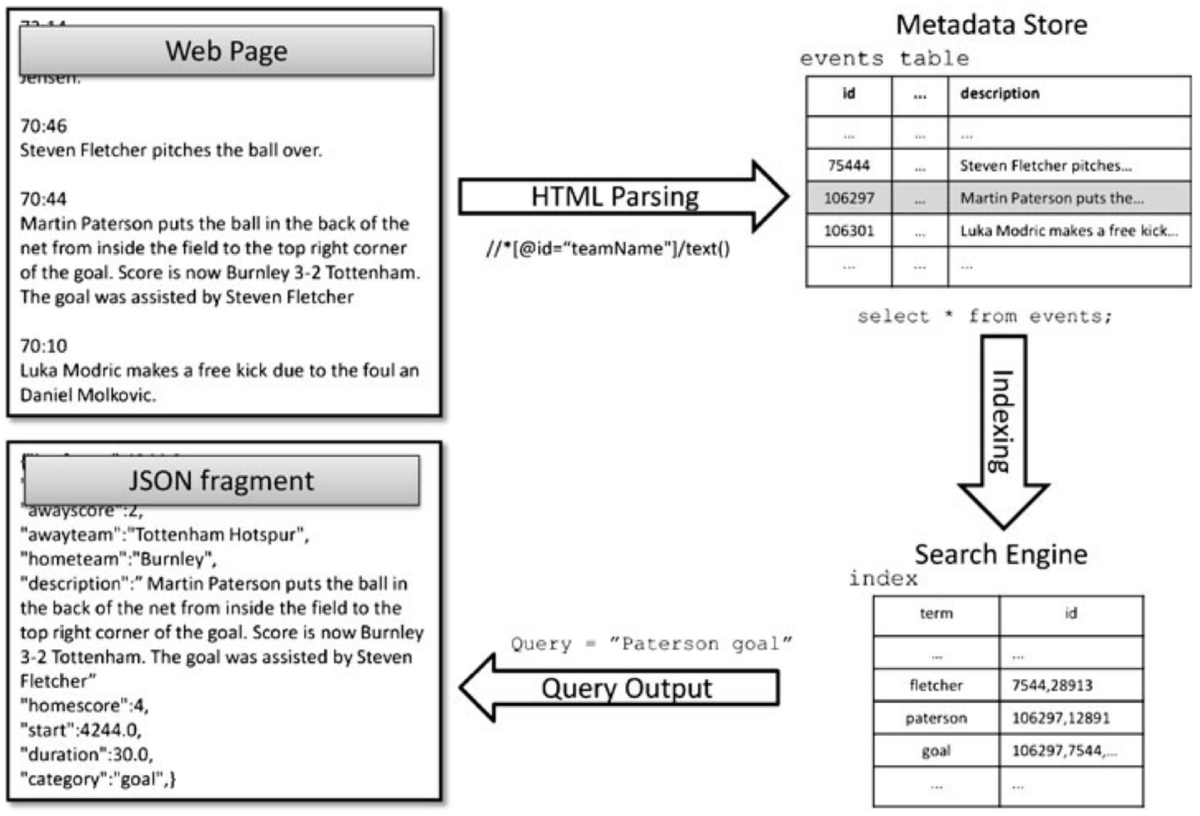
2.13.2 Segmentation and streaming

Davvi uses adaptive streaming for data delivery over HTTP, and therefore splits the video stream into many independent video segments. Each segment is encoded with multiple bit rate quality levels [63]. A challenge was to identify the ideal length of the video segments.

"For Davvi, we varied the size and experimented with lengths in this range to identify the optimal combination of granularity and quality. In this video search scenario, the lower range proved optimal, so Davvi uses two second segments. With shorter segment lengths, for instance one second segment, one can observe visible noise in the video playback due to flicker effects and decreased compression efficiency. With larger segments, we achieved negligible better compression, but since the search and quality adaption granularity is at segment boundaries, the two second length proved to be an efficient trade-off, i.e., a size also frequently uses in commercial systems" [63].

2.13.3 Metadata

To be able to deliver specific events with this system, a video resource needs to be mapped with descriptive metadata. Multimedia files cannot be parsed [173] the same way you would parse a text document. You parse the metadata instead, and map this data with timecodes representing the correct video segments in the video resource. Davvi does this with sporting events [63, 64]. Davvi bases its metadata on online commentators referencing the match clock. It is possible to scrape [202] web pages containing such metadata and store this data in a database table. This workflow is illustrated in figure 2.17



70:46
Steven Fletcher pitches the ball over.

70:44
Martin Paterson puts the ball in the back of the net from inside the field to the top right corner of the goal. Score is now Burnley 3-2 Tottenham. The goal was assisted by Steven Fletcher

70:10
Luka Modric makes a free kick due to the foul an Daniel Molkovic.

id	...	description
...	...	...
75444	...	Steven Fletcher pitches...
106297	...	Martin Paterson puts the...
106301	...	Luka Modric makes a free kick...
...	...	...

term	id
...	...
fletcher	7544,28913
paterson	106297,12891
goal	106297,7544,...
...	...

```
"awayscore":2,
"awayteam":"Tottenham Hotspur",
"hometeam":"Burnley",
"description":" Martin Paterson puts the ball in
the back of the net from inside the field to the
top right corner of the goal. Score is now Burnley
3-2 Tottenham. The goal was assisted by Steven
Fletcher"
"homescore":4,
"start":4244.0,
"duration":30.0,
"category":"goal",}
```

Figure 2.17: Davvi metadata search [63]

Metadata from a match is indexed and available for future playback. An end user can do a text based query and requested metadata about an event will be returned to the client. From the Video Manga video summary researched the in the previous section 2.9, we learned about the importance measure and selecting relevant keyframes from a video resource. Clicking on a keyframe in a storyboard begins video playback from that point. "Davvi also employs storyboards, but where the keyframes represent selected logical units within the video content, such as goals, penalties, and other significant events. These keyframes map directly to the specific keyword query issued by the user. The benefit of such an approach is that Davvi can support different interfaces to the video content, such as a listing of the important event (e.g., goals and corners) in the content, or the presentation of a ranked list of events or scenes" [63, 64].

2.13.4 Search engine

The developers of Davvi [63,64] emphasise the need to utilize the large amount of video content available today in an improved manner. They also emphasize the limitations of contemporary text-based queries only returning entire documents/links and not the selections we are really looking for. Even if it is not desirable to quick read an entire document, an end user accepts this because of the low latency of search queries. To do the same with video, you have to consider size and metadata. Traditionally, an end user downloads an entire movie or buys a DVD [135] to watch a favorite scene within a movie. This is time consuming and not very efficient.

The prototype Davvi aims to deliver only that favorite scene. It also delivers a small amount of video handles [19] before and after the query result, so the user can trim the clip exactly as desired. This is not only limited to a single video resource. Davvi is able to find and present video material from multiple sources in the same playlist. The system can present bits and pieces from different soccer matches. For example, a popular query might be all the goals from a specific soccer player. A Davvi user interface is illustrated in figure 2.18

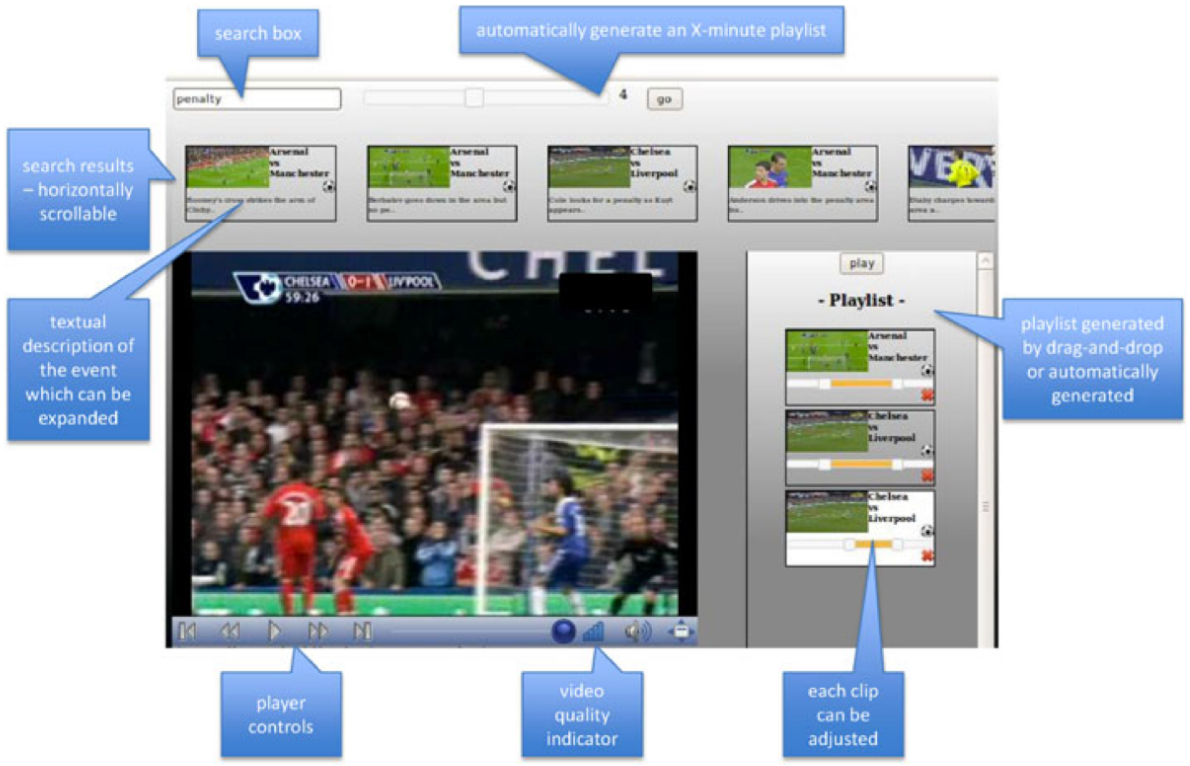

search box
automatically generate an X-minute playlist
penalty
4
go
search results – horizontally scrollable
textual description of the event which can be expanded
play
- Playlist -
playlist generated by drag-and-drop or automatically generated
player controls
video quality indicator
each clip can be adjusted

Figure 2.18: Davvi user interface [63]

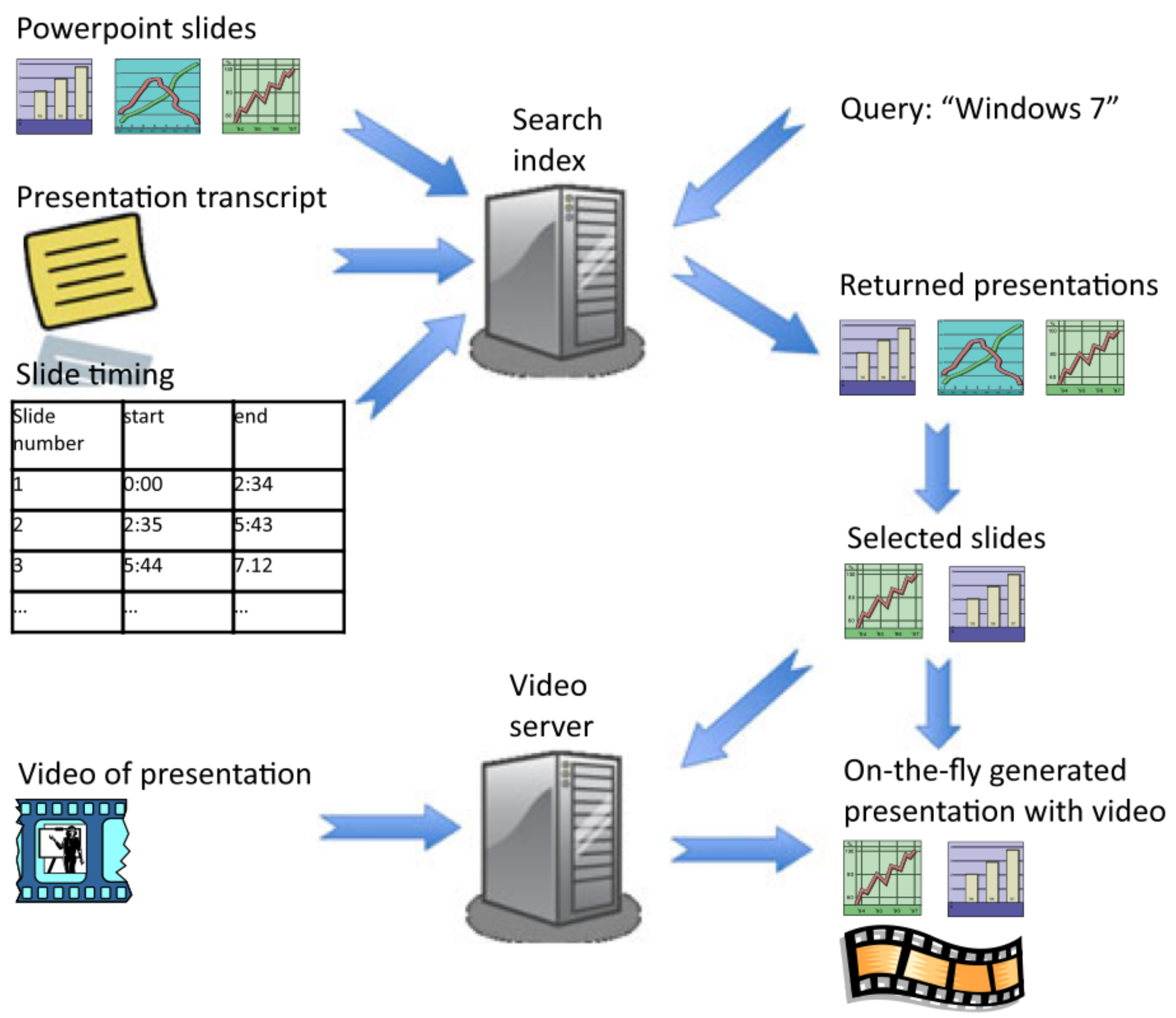
The system includes a playlist interface and a media player. This enables an end user to play through the clips and decide which goals to include in a playlist. Optionally, the system can auto-generate a playlist. Next, we will research how Davvi successfully has been integrated into a commercial system.

2.14 vESP

One way to make new technology interesting for large enterprises is to integrate this technology with their own products. The Davvi system has been integrated into the FAST enterprise search platform (ESP) developed by Microsoft. "This is a scalable, high-end enterprise search engine commercially used world-wide" [72].

2.14.1 vESP overview

vESP main functionality is to improve text-based searches. By adding Davvi functionality, ESP can now be used to automatically discover and extract relevant video scenes from for example existing PowerPoint presentations and transcripts. This functionality makes it possible to combine selected parts or slides from different presentations. "The idea of our video-enabled enterprise search platform (vESP) is thus to automate this process such that the user can generate personalized, customized presentations on-the-fly with the corresponding generated video" [39, 40, 63]. An overview of the system is illustrated in figure 2.19



Slide number	start	end
1	0:00	2:34
2	2:35	5:43
3	5:44	7.12
...	...	...

Figure 2.19: VESP system overview [63]

2.14.2 Selecting specific slides

ESP already returns slides based on textual queries, but the improved version vESP uses a playlist that can put together one or more slides from each presentation, and then create a new presentation from different archived resources. The video of the presentations is indexed corresponding to the selected slides, and streams the video segments necessary to present the wanted video. This saves the user for time-consuming searches after relevant video slides.

2.15 Evaluating text based video summary technologies

Davvi is an end-to-end prototype while vESP is extending a proprietary technology [72]. It is beneficial when a prototype is developed in such a way that it can easily be integrated with different systems. Since the Davvi prototype uses open source technologies [10, 11, 33], it is easier for a company to integrate Davvi with their own system than if the prototype was using another proprietary technology. For example, Google might not want a system that uses Microsoft's ESP [72] search engine technology.

2.15.1 End-to-end prototype or proprietary extension

Descriptive movie metadata might be text based documents. For this purpose, the vESP approach seems like a good option. Microsoft's search engine [72] returns results based on a queried phrase or category search. These search results are presented in a slide playlist. However, one specific search result might not be what an end user is looking for. Maybe an end user wants more of the video resource than what is presented? Hence, what happened before or after that specific search result. It seems that the vESP extension does not tailor for such needs. Davvi on the other hand, has a playlist with more options to edit each search result, providing video handles [19] before and after a certain position in the movie. This is a more flexible way to dynamically create new video material with our system in mind.

2.15.2 Adapting to our needs

Is it possible to use a Davvi like system to create new videos from existing movies? Both the Davvi prototype and the vESP extension are based on utilizing a large video archive. This is exemplified with sports events and powerpoint slides, but the same approach can be possible with movies. While a company pays for exclusive access to soccer matches from a specific series or tournament, so does streaming companies pay for exclusive access to certain movies. An integration of such functionality with their system, might give them an advantage over their competitors. Hence, such a company will gain profits by reusing media resources in a video archive.

The Davvi prototype will be used as an inspiration for our video summary system. However, our system cannot collect metadata in the same way as the Davvi prototype does by extracting online sports commentaries. Our system needs metadata about movie related material. What kind of metadata that is available will influence how our system will be built. Next, we will present possible movie metadata resources.

2.16 Film-series-metadata

In the last section, we concluded that available movie metadata will influence how our prototype will be built. So, what does descriptive metadata mean in an audio and video context?

Metadata is usually used for identifying, describing and classifying media resources [118].

- To identify the media might be useful information like clip or playlist names, duration, timecode and so on.
- To describe the content might be the quality of the video, rating, or description. For example, a movie might have keywords like car chase or romance associated to some scenes.
- To classify the media includes sorting the media, or to easily locate video content.

These types of metadata categories can be used on a macro or a micro scale. Macro scale is metadata of for example an entire movie. Micro scale is metadata about parts of a movie. Regardless of scale, the same techniques apply. To be able to create new video content from existing video fragments, we need metadata on a micro scale. For searching for example movie genres, we need metadata on a macro scale.

A challenge is the availability of metadata for movies. In a perfect scenario, every movie ever made has been logged in detail with time codes [190] and descriptive metadata like genre and dialog. This metadata should be available on a free website with an easy to use api to deliver the desired result. On the other hand, in a worst case scenario we have to log all metadata manually. This scenario is not all bad considering the flexibility in amount and structure of metadata. However, this is a time-consuming process. Next, we will research possible movie metadata resources.

2.16.1 IMDb

The IMDb [52] website contains a lot of information about movies and series. This is promising searchable metadata material. For example, IMDb references goof scenes within a movie and transcripts of dialog highlights. However, these references are without time codes.

Considering the size of the IMDb database it is plausible to assume they want to utilize their knowledge base. The company Amazon [7] owns products like IMDb and a streaming services. September 2012 they launched the feature X-Ray for movies. Later they also launched the same feature for television. It is available on tables [8] like Kindle Fire, Kindle Fire HD, and the Amazon Instant Video app on Wii U. "What does X-Ray do? With a single tap viewers can discover the names of the actors appearing on the screen and what they have been in" [6]. When you pause a movie, a window pops up listing every actor appearing in that given scene. The window shows links to their IMDb resume and it is possible to select other movies that the actor has appeared in for later viewing.

This require the X-Ray feature to know the specific timecode whenever an actor appear in the movie or television series. We do not have access to how Amazon has solved this. It is plausible to think that the current timecode when the movie is paused with a tap, is sent as a request to a server connected to the movie database. Available metadata for that scene is then returned to the device. In announcing the product, Amazon said they were working on making more movies available with the X-Ray feature. This might indicate they log the movies manually, or use some kind of image analysis technology to track actors throughout a movie.

2.16.2 Filmgrail

Filmgrail is a Norwegian based company aiming to provide unique collections of metadata for movies. The company is in development. Their primary goal is create a search engine for movies. They argue contemporary movie search engines does not have good enough metadata. The company has hired a team of 15 professional loggers in Russia. The team has logged over 3000 movies and tagged each movie where action scenes and romance scenes are quantified and measured. All information is saved in their database for later searches [105]. Upon completion, this product will offer an api for metadata based searches. They argue streaming companies needs to offer its users relevant hits on search queries. For example, Netflix has invested a lot of money in advanced algorithms in trying to suggest what movie you want to see [105]. Their goal is to become "big data" for movies hoping both professional streaming companies and end users want to use their products. Their metadata API might be helpful for our needs when ready.

2.16.3 Movie transcripts

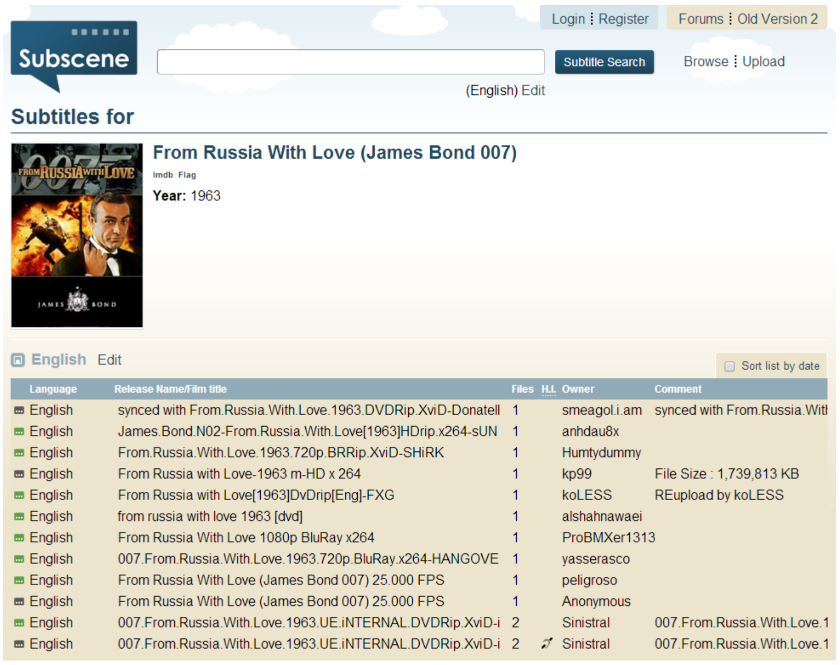
Big budget movies are usually logged prior to post-editing. Logging means going through the source video material and labeling it accordingly to its contents with metadata. This is typically done with a timecode and a description what happened in that part of the video material. Logged metadata might not be relevant to the end product, but it might be plausible that similar metadata exists for the finished movie.

Unfortunately, we do not have access to such metadata, but it is possible to collect bits and pieces from websites. A place to start is movie transcripts. Websites like Internet Movie Script Database claims to have "the biggest collection of movie scripts available anywhere on the web" [62]. The website lets you read or download movie scripts for free. It also allows text based searches and lists movies based on genre. There is a large collection of movie transcripts and a few television series. Transcripts are presented in HTML. Such transcripts are useful for manually building metadata about movies. Even if the transcripts may not entirely be as the final movie, it is a good reference to what is happening in the movie. Transcripts are text-based, and describes in detail scenes and dialog throughout the movie. Unfortunately, they do not include time codes. Transcripts can be used in combination with manual logging. However, it is time-consuming to create metadata manually. A movie duration is approximately between ninety minutes and three hours. The amount of metadata can be endless.

2.16.4 Subtitles

Another option is to use subtitles for search queries. Subtitles need to be mapped to the correct part of the movie with accuracy. Hence, the necessary timecode for that part of the movie is available. If lists of subtitles to specific movies are publicly available, then this might be a good start for using metadata to locate movie fragments.

These snippets of dialogue may not be entirely what the actors are saying in the movie, but provides enough information to locate a specific scene. Some subtitles supports description for the hearing impaired. Meaning, text describing sounds and atmosphere in the movie. A useful website for finding such subtitles is Subscene [103]. Figure 2.20 illustrates a search result for this movie using Subscene.



Subtitles for

From Russia With Love (James Bond 007)

Year: 1963

English Edit

Language	Release Name/Film title	Files	H.I.	Owner	Comment
English	synced with From.Russia.With.Love.1963.DVDRip.XviD-Donatell	1		smeagol.i.am	synced with From.Russia.Wit
English	James.Bond.N02-From.Russia.With.Love[1963]HDrip.x264-sUN	1		anhdau8x	
English	From.Russia.With.Love.1963.720p.BRRip.XviD-SHiRK	1		Humtydummy	
English	From Russia with Love-1963 m-HD x 264	1		kp99	File Size : 1,739,813 KB
English	From Russia with Love[1963]DvDrip[Eng]-FXG	1		koLESS	REupload by koLESS
English	from russia with love 1963 [dvd]	1		alshahnawaei	
English	From Russia With Love 1080p BluRay x264	1		ProBMXer1313	
English	007.From.Russia.With.Love.1963.720p.BluRay.x264-HANGOVE	1		yasserasco	
English	From Russia With Love (James Bond 007) 25.000 FPS	1		peligroso	
English	From Russia With Love (James Bond 007) 25.000 FPS	1		Anonymous	
English	007.From.Russia.With.Love.1963.UE.iNTERNAL.DVDRip.XviD-i	2		Sinistral	007.From.Russia.With.Love.1
English	007.From.Russia.With.Love.1963.UE.iNTERNAL.DVDRip.XviD-i	2	👂	Sinistral	007.From.Russia.With.Love.1

Figure 2.20: Subscene example [104]

This website has filtering where you can choose language and if a subtitles support descriptions for the hearing impaired. The Subscene database has a large amount of movies and television series subtitles. One of our test movies will be the James Bond movie "From Russia with Love (1963)" [48]. A possible extension of our system is support for television series. TVsubtitles.net has a large amount of television series subtitles. According to their site statistics from mid june 2013, their database include almost a quarter of a million subtitles from approximately thirteen hundred different television shows [110]. A similar website for movies is the Moviesubtitles.org. This database include approximately eight six thousand subtitles from ten thousand movies [81]. Both TVsubtitles.net and Moviesubtitles.org include subtitles in several languages. However, they lack a filter for hearing impaired subtitles.

The .srt subtitle file format

The file format .srt is often used in subtitling. It is a text file that repeat the same structure [191]. Consider this action scene from the movie "Die Hard (1988)" [47] with hearing impaired support:

```
271
00:22:54,562 --> 00:22:56,562
...[speaking German]...

272
00:23:10,851 --> 00:23:12,543
...[crowd screaming]...

273
00:23:12,643 --> 00:23:14,819
...[machine gun fire]
```

Every subtitle or sound description in the movie has an incremental unique id. This can easily fit into a table in a database. The timecode is millisecond specific. Between segment 272 and 273 in the example, there seems to be a minimum of a hundred milliseconds gap between continuous subtitling. Hearing impaired information is included in brackets. Searchable keywords words can for example be "German", "screaming" or "machine gun". Alternatively, plain dialogue can be used as searching. Consider this example when the hero confronts the villain:

```
1520
01:56:19,049 --> 01:56:20,630
Americans... all alike.

1521
01:56:20,730 --> 01:56:24,518
This time, John Wayne does not walk off
into the sunset with Grace Kelly.

1522
01:56:24,618 --> 01:56:26,358
That was Gary Cooper, asshole.

1523
01:56:26,458 --> 01:56:28,167
Enough jokes.

1524
01:56:28,267 --> 01:56:30,999
You’d have made a pretty
good cowboy yourself, Hans.
```

```
1525
01:56:31,099 --> 01:56:35,560
Oh, yes. What was it
you said to me before?

1526
01:56:35,660 --> 01:56:39,242
Yippee-ki-yay, motherfucker.
```

Meta references to other movies in dialog might result in surprising search results. You might not expect a search for "John Wayne", "Grace Kelly" or "Gary Cooper" to result in a scene from "Die Hard" [47]. The actor Bruce Willis [45] famous phrase "Yippee-ki-yay..." used on a database containing all "Die Hard" movies might give you the desired result. Such one-liners might be a popular search query.

Using dialogue and hearing impaired descriptions might provide a good starting point for automated metadata searches in movies. One could consider combining key words with certain categories or labels. For example is "Yippee-ki-yay..." a famous one-liner, while "machine gun fire" can imply an action scene. Hopefully, the use of subtitle .srt files will provide new opportunities and ideas for effective metadata searches.

One possible usage of a dialogue based montage is actually used within a "Die Hard" movie. In the movie "Live Free or Die Hard (2007)" [53] a terrorist threat is conveyed via a cutup montage of presidential speeches. This is an example of remixed video using dozens of small segments from speeches from different US presidents like John F. Kennedy, Ronald Reagan and George W. Bush. Such editing techniques might have other purposes than creating a home video. "This technique of transforming the meaning of presidential speeches via cutup became a staple strategy among politically motivated remixers during the 2004 presidential election" [9].

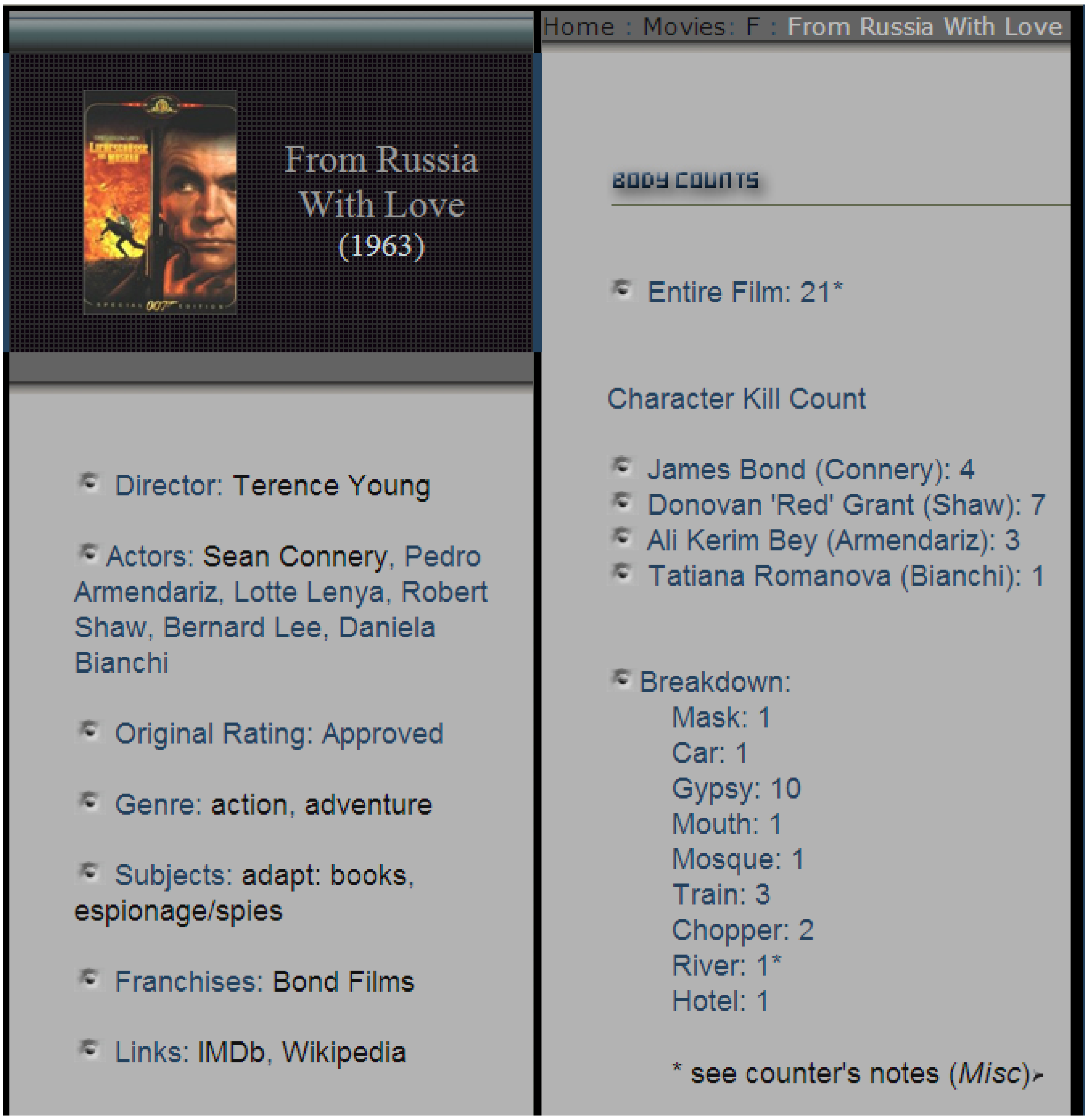
Creating a prototype with easy access to a large video archive might be used for different purposes depending on the users of the system. The movie industry might also have a motive for including such a montage within this "Die Hard" movie. "With this appropriation of the strategy and placing it in the hands of Internet-based hacker-terrorists, the filmmakers reinforce a narrative mendaciously associating media remixing with piracy and terror networks" [9].

2.16.5 Movie scenes

Subtitle metadata can be used for dialogue based searches. When people talk about a movie they often refer to a specific scene within the movie. This might be a scene of a certain type like a car chase or romance scene. People have different preferences with only their imagination setting the limit. There are many websites dedicated to such preferences. We will present one extreme example. The Norwegian newspaper "Aftenposten" wrote an article about the actors that have killed most people in movies [35]. According to this article, the actor Arnold Schwarzenegger [44] and the actress Uma Thurman [61] have killed most people on screen throughout movie history. Ph.D. student Randal S. Olson at Michigan State University is interviewed in the article. He has researched how lethal each actor and actress are. His ranking is based on data from the website community moviebodycounts.com [79]. "They are very accurate about humans, animals, zombies or other fantasy figures that meet their maker in movies. But rules are strict: People that get killed by for instance a plane falling down or buildings crushed, is not part of the count" [35]. Using a web scraping [202] tool, he extracted the data from the website into his own database. Maybe we can use the same website as searchable metadata for our prototype?

2.16.6 Movie tribute website

"Movie Body Counts tallies the actual, visible on screen kills/deaths/bodies of your favorite action, sci/fi, and war films" [79]. In this website it is possible to search for movies, actors and directors. A search for our test movie "From Russia with love" is illustrated in figure 2.21.



Home : Movies: F : From Russia With Love

From Russia With Love (1963)

- Director: Terence Young
- Actors: Sean Connery, Pedro Armendariz, Lotte Lenya, Robert Shaw, Bernard Lee, Daniela Bianchi
- Original Rating: Approved
- Genre: action, adventure
- Subjects: adapt: books, espionage/spies
- Franchises: Bond Films
- Links: IMDb, Wikipedia

BODY COUNTS

- Entire Film: 21*

Character Kill Count

- James Bond (Connery): 4
- Donovan 'Red' Grant (Shaw): 7
- Ali Kerim Bey (Armendariz): 3
- Tatiana Romanova (Bianchi): 1

- Breakdown:
  Mask: 1
  Car: 1
  Gypsy: 10
  Mouth: 1
  Mosque: 1
  Train: 3
  Chopper: 2
  River: 1*
  Hotel: 1

* see counter's notes (*Misc*)

Figure 2.21: Movie Body Counts example [80]

The column on the right shows statistics about body counts in the movie. It lists the entire movie, kill count by actor and the breakdown section seems to be scene based. Unfortunately, these statistics does not include time codes. There is a link to "counter's notes" [78], a forum with more information about each movie. A search for the same movie in this forum returned more rich textual information about the body counts. Unfortunately, there are no time codes included in the count. However, it seems that counting is very accurate. "On another boat there are 2 men and 1 on fire and the on-fire-guy falls into the water. Just after he does, the boat explodes with the 2 others still onboard. The guy who fell off didn't fall far enough away from the boat when it blew up, allowing me to strongly assume he was killed, adding up to at least 6 kills total for the scene and for Bond" [78].

This is one extreme example of a movie tribute webpage. There are probably hundreds of such webpages with different themes. They contain lots of usable metadata, but not all include time codes.

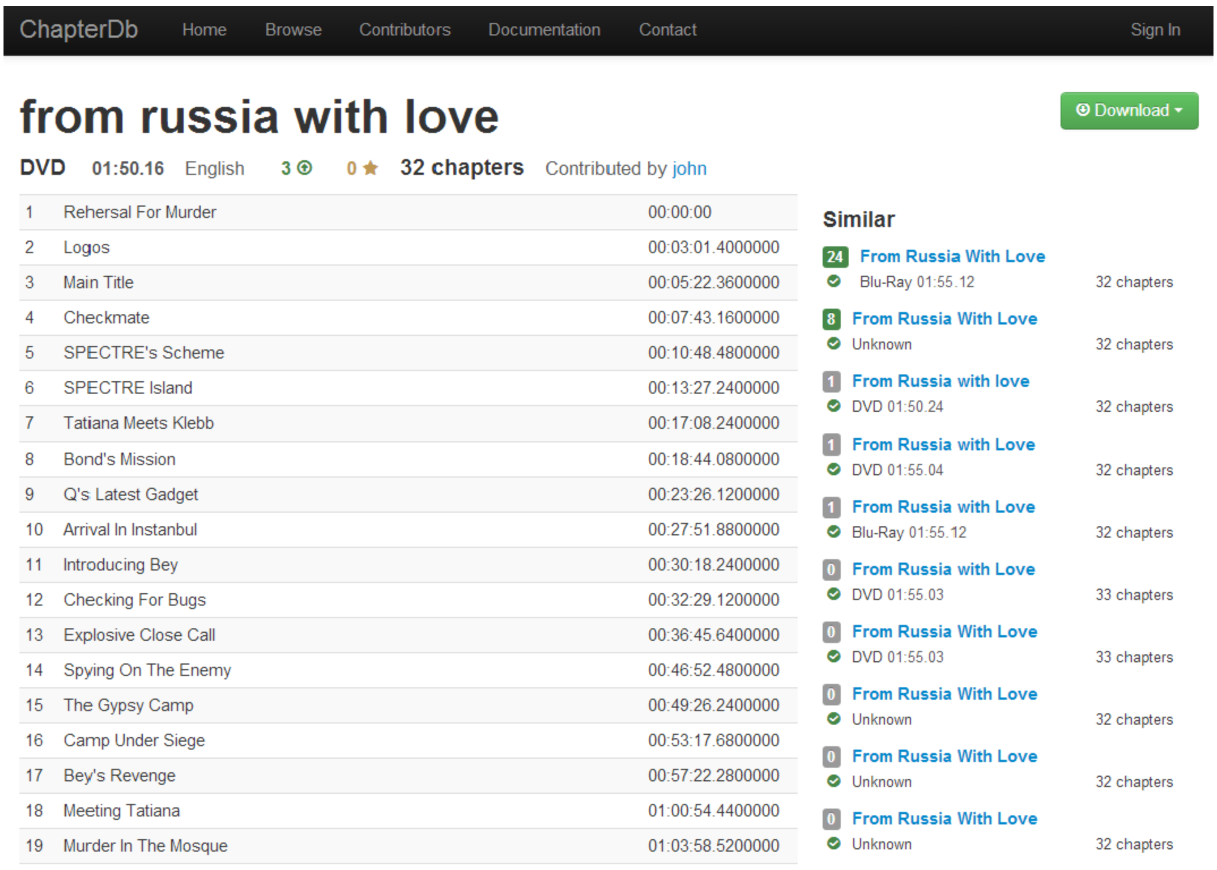
2.16.7 Combining metadata and time codes

Our challenge so far has been to find websites that contains both metadata and time codes. Another approach is to use available information within the movie container. A DVD or BlueRay movie release usually includes chapters. They reference scenes within a movie. Chapters are presented with a thumbnail of the scene, an index and a textual description. It is possible to extract chapter information by using video program tools such as FFmpeg [33]. However, to our knowledge extracting chapter information from the video material only returns the index and time codes. We still need to retrieve textual description and thumbnails.

There are websites providing chapter information. A search for "From Russia with love" on the website tagChimp [106] presents an overview similar to the movie body count webpage illustrated in figure 2.21. This includes information about the release date, genre, director, casting and so on. Instead of body counts, this webpage shows chapter information. They are organized with an index and a textual description. However, no timecodes.

2.16.8 Online chapter database

The next step is to figure out if it is possible to combine chapter time codes extracted from a movie container with chapter information from a webpage like tag chimp. Fortunately, this has already been done. The website chapterDB [22] include both chapter information and time codes. A search for our test movie "From Russia with love" is illustrated in figure 2.22.



from russia with love

DVD 01:50.16 English 3 0 ★ 32 chapters Contributed by john

1	Rehersal For Murder	00:00:00
2	Logos	00:03:01.4000000
3	Main Title	00:05:22.3600000
4	Checkmate	00:07:43.1600000
5	SPECTRE's Scheme	00:10:48.4800000
6	SPECTRE Island	00:13:27.2400000
7	Tatiana Meets Klebb	00:17:08.2400000
8	Bond's Mission	00:18:44.0800000
9	Q's Latest Gadget	00:23:26.1200000
10	Arrival In Instanbul	00:27:51.8800000
11	Introducing Bey	00:30:18.2400000
12	Checking For Bugs	00:32:29.1200000
13	Explosive Close Call	00:36:45.6400000
14	Spying On The Enemy	00:46:52.4800000
15	The Gypsy Camp	00:49:26.2400000
16	Camp Under Siege	00:53:17.6800000
17	Bey's Revenge	00:57:22.2800000
18	Meeting Tatiana	01:00:54.4400000
19	Murder In The Mosque	01:03:58.5200000

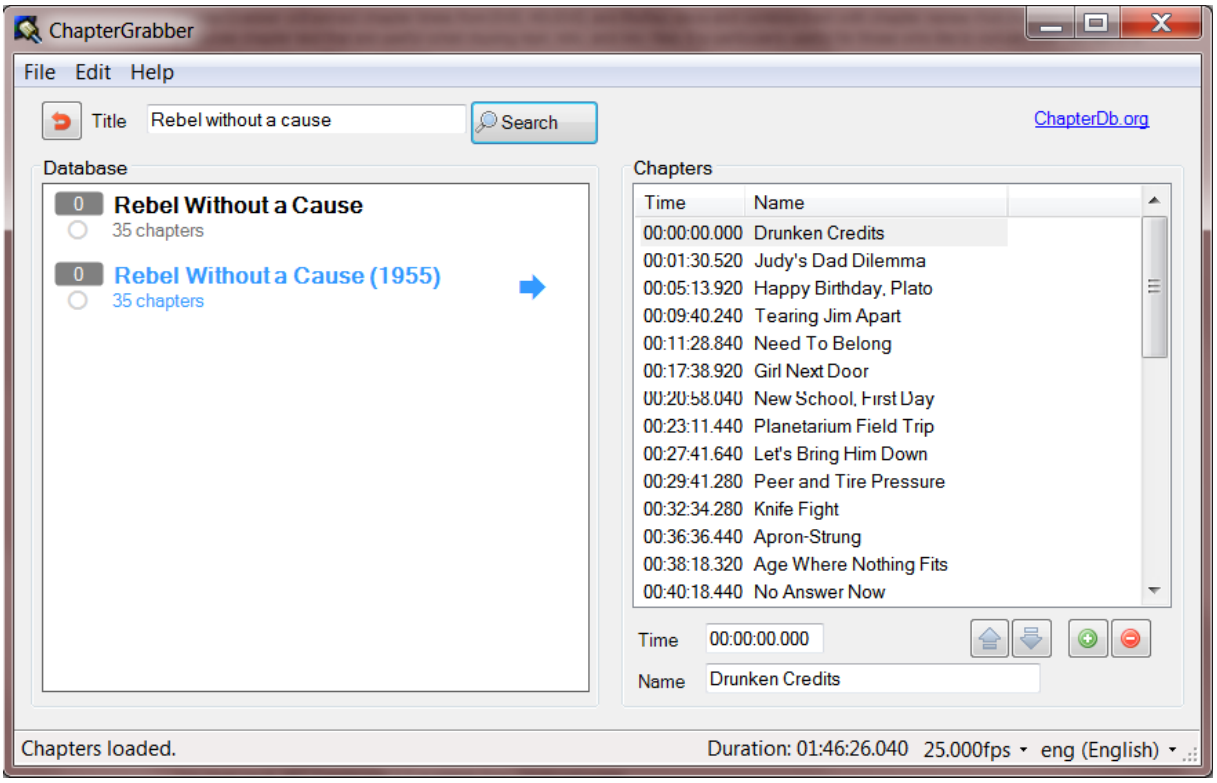
Similar

24	From Russia With Love	Blu-Ray 01:55.12	32 chapters
8	From Russia With Love	Unknown	32 chapters
1	From Russia with love	DVD 01:50.24	32 chapters
1	From Russia with Love	DVD 01:55.04	32 chapters
1	From Russia with Love	Blu-Ray 01:55.12	32 chapters
0	From Russia with Love	DVD 01:55.03	33 chapters
0	From Russia With Love	DVD 01:55.03	33 chapters
0	From Russia With Love	Unknown	32 chapters
0	From Russia With Love	Unknown	32 chapters
0	From Russia With Love	Unknown	32 chapters

Figure 2.22: ChapterDb example [23]

The search result includes a lot of useful information. In addition to the chapter information, we are also informed about the duration and frame rate [140] of the movie. This will be useful when mapping chapter information with video segments. Most importantly, it is also possible to download the search result as text or XML. The downloaded file can then be parsed and relevant metadata included in the prototype's database.

The websites database is synced with a software extracting time codes from media sources such as DVD's. This software is a Windows based open source program ChapterGrabber. It is written by Jarrett Vance [112] and extracts chapter times from DVD, HD-DVD, and BluRay discs and combine them with chapter names from the Internet. We downloaded the program and tested it with the James Dean movie "Rebel Without a Cause (1955)" [58]. The result is illustrated in figure 2.23.



Title: Rebel without a cause

Database:

0 **Rebel Without a Cause** — 35 chapters

0 **Rebel Without a Cause (1955)** — 35 chapters

Time	Name
00:00:00.000	Drunken Credits
00:01:30.520	Judy's Dad Dilemma
00:05:13.920	Happy Birthday, Plato
00:09:40.240	Tearing Jim Apart
00:11:28.840	Need To Belong
00:17:38.920	Girl Next Door
00:20:58.040	New School, First Day
00:23:11.440	Planetarium Field Trip
00:27:41.640	Let's Bring Him Down
00:29:41.280	Peer and Tire Pressure
00:32:34.280	Knife Fight
00:36:36.440	Apron-Strung
00:38:18.320	Age Where Nothing Fits
00:40:18.440	No Answer Now

Time: 00:00:00.000

Name: Drunken Credits

Chapters loaded. Duration: 01:46:26.040 25.000fps eng (English)

Figure 2.23: Rebel without a cause chapters

By using the DVD as the movie source, the program automatically detects frame rate, duration of the movie, and all chapter time codes. They appear in the right hand window of the program. By typing the name of the movie in the search bar, the program gives you the option of syncing the chapters with metadata located at either the website "tagChimp" [106] or "chapterDB" [22]. The search result appears in the left hand window of the program. The number of chapters should match between the windows. By pressing one of the search results, a blue arrow appears and the "Chapter" text in the right hand window is replaced with the correct chapter description. Afterwards, it is possible to export chapters with time codes to multiple text formats.

The edit pane in the program has an option "Sync Online Database". If this option is checked, our experiment with the movie "Rebel without a cause" might be added to the "chapterDB" website. This means every user of the program can contribute to building the chapter database.

An implementation of our prototype require searchable chapters. They are not as specific as categories such as car chases or romance scenes, but descriptive enough to locate many scenes within a movie. If an end user of the prototype already have seen the movie, then chapter searches will provide easy access to a favorite scene. With searchable access to scenes from different movies, it should be possible to create montages containing chapters from different movies.

2.17 Summary

We have described adaptive streaming as an improvement of earlier streaming technologies. Then, we compared different adaptive streaming technologies. Apple and Microsoft protocols are examples of proprietary technologies. MPEG-DASH is an attempt to standardize adaptive streaming, and the specification is a collaboration of different companies. We concluded that even though the Safari [16] web browser makes a lot of predefined choices regarding video quality, it is optimized for the platforms it supports. Apple also provide good developer guidelines for implementing this technology, and offer specialized software to create the adaptive video.

Next, we introduced existing technologies for searching video resources. Image analysis demonstrated that it is possible to present parts of a large video resource in a matter of minutes. Davvi demonstrated that text based metadata can be used for searching and presenting sporting events. We decided to use this end-to-end prototype as an inspiration, but we had to develop our system in a way that handles movie metadata.

Finally, we researched possible movie metadata resources. We were particularly interested in online databases containing both time codes and textual descriptions. We found that subtitles and chapters are suitable metadata resources that can be extracted into our system. In the next chapter we will design and implement our prototype.

Chapter 3

Design and Implementation

We learned that subtitle and chapter metadata can be used to extract events from movies in the previous chapter. In this chapter we will implement a system using such metadata resources. An overview of the necessary components in our system is illustrated in figure 3.1.

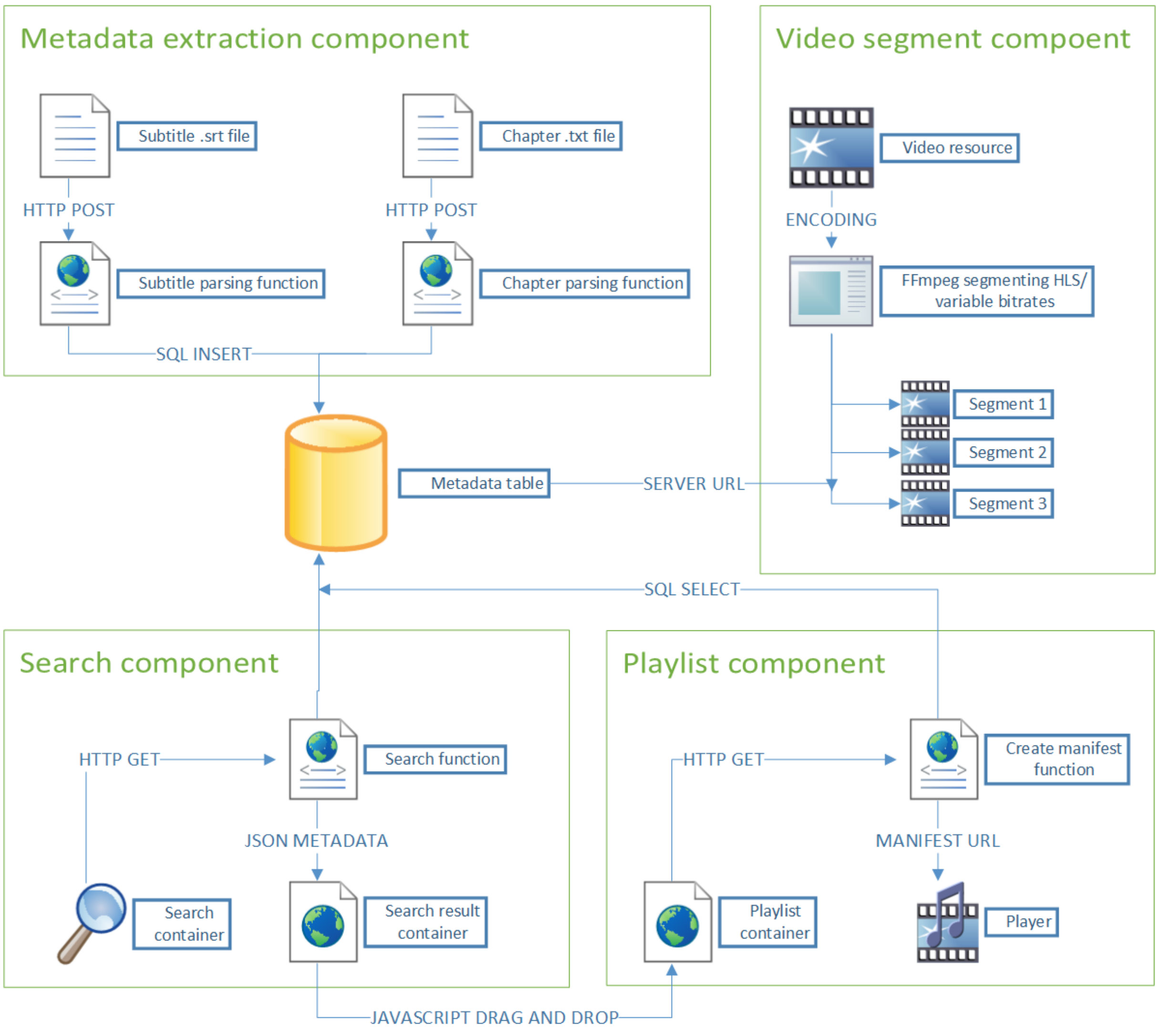


Figure 3.1: System components

A search component will consist of a text based search engine and a category list of available movies and scenes within them. Using a metadata extraction component, it will be possible to search for dialogue and scenes in a relational database. This database includes file paths to a video archive component containing a file structure of video segments. A playlist component allows end users to select which video segments to be played back. We do use some existing components like the Apache web-server [10] and the FFmpeg [33] encoding tools.

Our system will be developed in multiple steps. First we will focus on getting HLS [13, 14, 31, 95] up and running in a web interface. We will experiment with this adaptive streaming technology and try to dynamically manipulate a manifest file. This enables us to stream new movies based upon video segments ranges in the video archive. We will evaluate different segmentation alternatives. Optimizing this process, we will continue with creating the metadata extraction component and the system's searching capabilities. Then, we will describe the system's user interface. Finally, we will ready our system for adaptive streaming. We will explain functionality and system design as the work progresses.

This development process will be illustrated with pictures of the user interface, code samples and so on. The user interface shows video and thumbnails from our test movies. This information has been replaced with a black box because the movies are copyright protected. Samples of code are also illustrated. Not all code had been fully developed at the time these pictures were taken. Hence, some error handling and optimization happened at a later stage in the process. The main point of these samples is to show the basic functionality of the system.

3.1 Technical choices

We have chosen to use an Apache web server for hosting the prototype. This is convenient since our research lab uses such servers to host web sites. Apache web server and MySql [84] database is part of the open source XAMPP [206] web server solution stack package. We will first test the system on a computer acting as both the client and the server. XAMPP can be configured for such purposes. The system also offers a backend interface "phpMyAdmin" which can be used to update the prototype's database. While adding metadata to the prototype, we need data management. When the prototype has been developed to a certain point, the project will be moved to our research lab server. Hosting the prototype on an external server will test the prototype's adaptive capabilities under different network conditions.

The XAMPP 1.8.3-2 version will be used for this project. The prototype will be developed on a Macbook Pro late 2013 with OS X version 10.9.1 Mavericks. This computes has a processor with 2 GHz Intel Core i7, 8 GB 1600 MHz DDR3 RAM and a Intel Iris Pro 1024 MB graphic card. The streaming protocol require a client that support HLS. The web browser Safari [16] does this natively. Current version of the browser is 7.0.1. We will be working with video resources of different formats. Hence, we need a tool to encode this material to HLS supported H.264 [145] video and AAC [123] audio. The encoded video needs to be wrapped in the MPEG-2 TS [165] format. Then, the media will be chopped into multiple small segments referenced by a .m3u8 index file. One alternative is to use Apple's Developer tools. The "Mediafilesegmenter" tool can be used to segment the encoded video. Another alternative is to use the open source tool FFmpeg [33]. First we will try FFmpeg. Later, we will also try the "Mediafilesegmenter" and compare the two alternatives. FFmpeg offers free tools for recording, converting, streaming and playing multimedia content. Basically, we are interested in a command line tool that inputs a video file format, encodes it to the right format and outputs segmented .ts files with a matching .m3u8 index file. This is possible with FFmpeg.

3.2 Experimenting with manifest files and HLS

For starters, we will implement a simple web client supporting HLS. The goal being to be able to stream a movie. We have chosen the classic James Bond movie "Goldfinger (1964)" [51] played by the actor Sean Connery [59]. The movie has many typical elements from the franchise like the theme song, one-liners, the Aston Martin car, evil villains and so on. Such elements can be used for metadata searches on a later stage of the project.

To be able to run HLS via an Apache web server, we need to add two new MIME [161] types. In XAMPP these are included in the file "mime.types". This file is located in the "xampp/xamppfiles/etc" catalog. The new MIME types adds support for the manifest file format .m3u8 and the transport stream format .ts.

```
application/x-mpegURL .m3u8
video/MP2T .ts
```

FFmpeg is a complex command line tool. It allows for a lot of input concerning video and audio quality. These are the input parameters we used for segmenting the "Goldfinger" movie:

```
./ffmpeg -i ./goldfinger.mov (input file)
-c:a libmp3lame (audio codec)
-ar 48000
-ab 64k (audio size)
-c:v libx264 (video codec)
-b:v 96k (video size)
-flags
-global_header
-map 0
-f segment (output format)
-segment-time 10 (segment length)
-segment_list movie.m3u8 (manifest path and name)
-segment_formats mpegts gfsegments/gf%05d.ts
(segment type, path and name)
```

Note the last three parameter inputs. They tell FFmpeg the desired segment duration, the name and location of the .m3u8 index file and the video .ts files.

In XAMPP a web server project is placed in a subdirectory under the "htdocs" folder. In a directory named "jb" we have placed an "index.html" file and the .m3u8 manifest file. In the subfolder "goldfinger-segments", we have placed the video segments. The "index.html" file contains this basic HTML script:

```
<html>
<body>
<video controls autoplay src="http://localhost/jb/index.m3u8">
</video>
</body>
</html>
```

The "autoplay" attribute within the video tag, auto trigger the manifest file. The "src" attribute indicate the location of the manifest file. The name "localhost" defines that the server is run on the same machine. This markup is enough to pause and play the movie, or go to another part of the movie. Figure 3.2 illustrates the HTML5 video player.

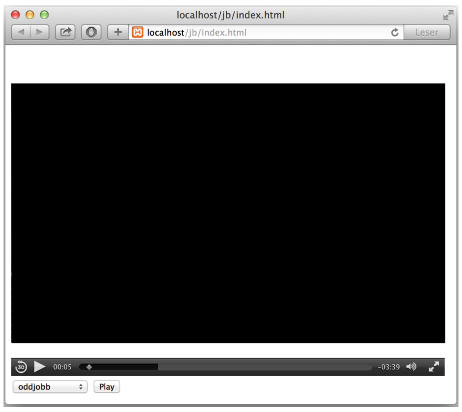


Figure 3.2: Video player in the Safari browser

3.2.1 Challenges with transcoding a video resource

FFmpeg is a powerful tool and it will take some time to research how all the necessary parameters work. A second attempt to make a segmented HLS movie from a .mp4 movie resulted in an error. We tried several combinations of input parameters before we realized the video file could be converted to another file format before using FFmpeg. We converted the video file to the Quicktime [176] .mov format.

The Quicktime player has some simple and convenient editing functions. It is possible to remove parts of a video resource, or add new video to the end of a video resource. This edited video is then rendered and saved as a new video file in the .mov format. This is particularly useful when we wanted to test only parts of a movie for segmenting results. Using FFmpeg on a feature film is time-consuming. It is preferable to use a video resource of a shorter duration to see if it is successfully segmented the way we want.

Converting the video file from .mp4 to .mov made FFmpeg output a successful result with the same input parameters as the last attempt. MPEG-4 [164] is a container format, and maybe our second test movie contained some data FFmpeg did not accept. By converting to another file format like .mov, maybe this data was removed or ignored. According to the FFMPEG synopsis, "each input or output file can, in principle, contain any number of streams of different types (video/ audio/ subtitle/ attachment/ data). The allowed number and/or types of streams may be limited by the container format" [33].

3.2.2 First attempt to stream separate parts of a movie

The manifest file references all the segmented video files in order. The first segment is named "gf00000.ts". The last segmented movie is "gf00631.ts". The letters in the filename is optional as the manifest only need the numbers to play them in sequence. We have already proven that we can play an entire movie using the manifest file. Now we want to edit the manifest file and remove unwanted parts of the movie. To test this, we made five copies of the manifest file. We renamed them:

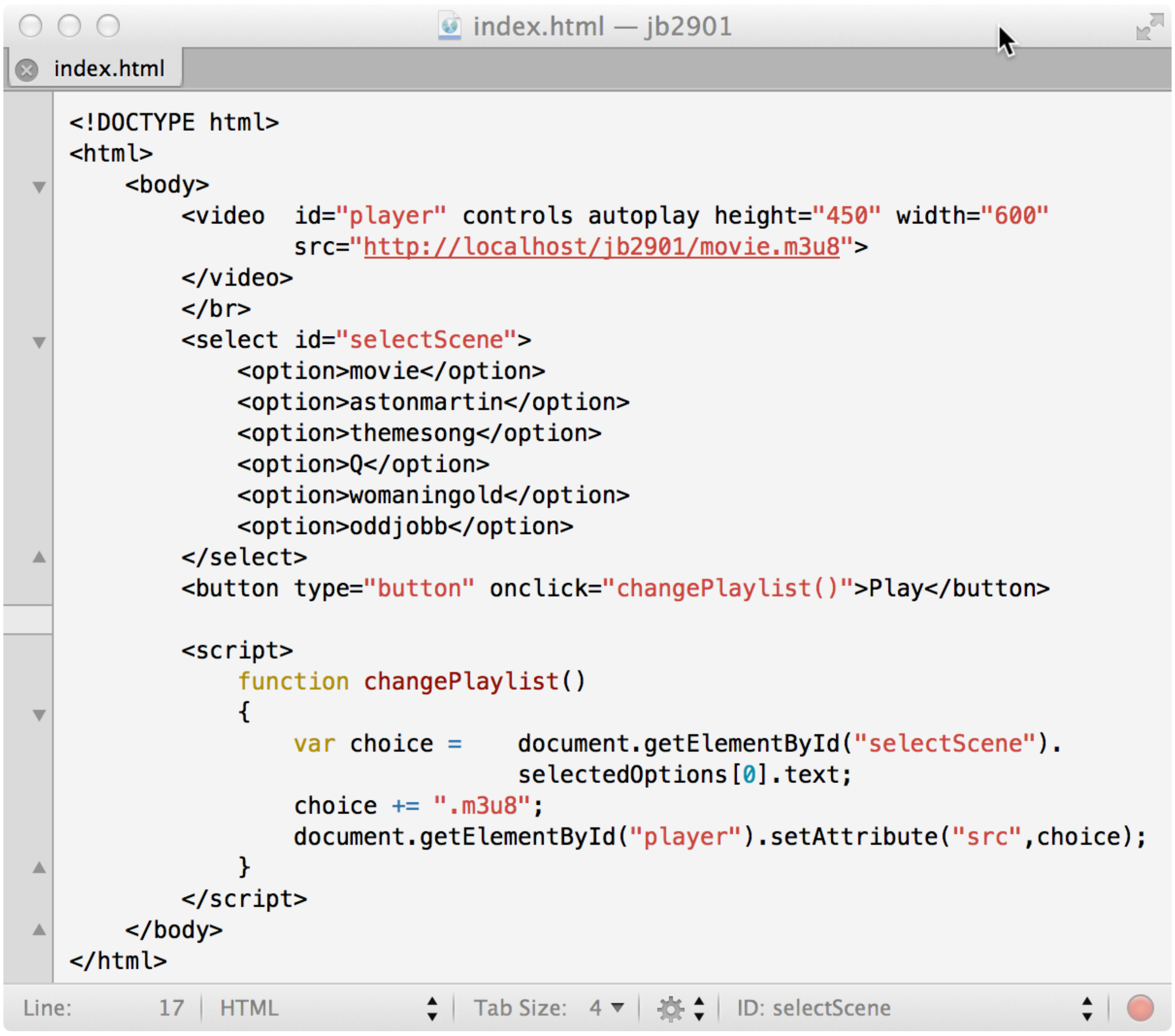
- Movie
- Astonmartin
- Themesong
- Q
- Womaningold
- Oddjobb

These key words references scenes within the movie "Goldfinger". First, we manually forwarded the movie to find the start time for each scene. For example, if each segment is 10 seconds long and the .ts files is incremented from 0, it is easy to calculate approximately which segment contains the start time. One minute into the movie will probably be segment 7 since 10 seconds x 6 segments = 1 minute.

We wanted to create video segments of 10 seconds duration. Given this input parameter, FFmpeg did not output segments of 10 seconds duration as expected. For example, the first 10 segments varied from 3 seconds to 17 seconds. It seems 10 seconds is the average value. Being an average, it is still fairly easy to calculate which segment gives an approximately starting time. Each video segment is an individual fragment of the movie and can be played back in for example a Quicktime player. This is an easy way to check if the video segment matches our calculation. We used the same approach to find which video segment ending a movie sequence.

Start and end segments does not match entirely with the movie resource. Some scenes change within a segment. This approach only works as rough edits and some fine trimming is necessary to playback a perfect cut. However, for this attempt it is more than enough. In the renamed manifest files, we only included the necessary segments to playback one scene. For example, the "Oddjobb" sequence only contain reference to segment 569 - 590, being a total duration of 3 minutes and 44 seconds. Note that the folder containing all the segmented video files has not changed. This means we can use different manifest files to reference the same segmented video resources. This is necessary for further development of the system.

For this attempt we needed to change the HTML index file. The changes are illustrated in figure 3.3.



```
<!DOCTYPE html>
<html>
    <body>
        <video  id="player" controls autoplay height="450" width="600"
                src="http://localhost/jb2901/movie.m3u8">
        </video>
        </br>
        <select id="selectScene">
            <option>movie</option>
            <option>astonmartin</option>
            <option>themesong</option>
            <option>Q</option>
            <option>womaningold</option>
            <option>oddjobb</option>
        </select>
        <button type="button" onclick="changePlaylist()">Play</button>

        <script>
            function changePlaylist()
            {
                var choice =    document.getElementById("selectScene").
                                selectedOptions[0].text;
                choice += ".m3u8";
                document.getElementById("player").setAttribute("src",choice);
            }
        </script>
    </body>
</html>
```

Figure 3.3: Index file

We have added a select tag with options containing the base name of the different manifest files. The default behavior of the video player is to stream the entire movie manifest. To change to a selected manifest, an end user needs to press the button "play". This triggers a javascript function that receive the selected option base name from the select tag, and adds the .m3u8 file extension. Afterwards, the "src" attribute of the video tag is changed to the chosen manifest file. Since the video tag uses the "auto play" attribute, new manifest file starts to play instantly.

The system works as intended. We are able to switch manifests to access desirable scenes in the movie. Next, we will try to make a montage of multiple movie scenes.

3.2.3 Improving the user interface

Our workflow so far has required manually creating manifests with specific knowledge of when a movie scene begins and ends. Our goal is to be able to search for possible scenes using metadata and subtitles resources. We will develop such functionality in several steps.

We will begin by developing an user interface. This interface will allow an end user to search for relevant scenes within a movie. Then, the user can drag and drop selected clips from the search result to a playlist. It will also be possible to rearrange the order of the clips before finally creating a manifest and playback this new movie montage. The user interface will look something like this figure 3.4.

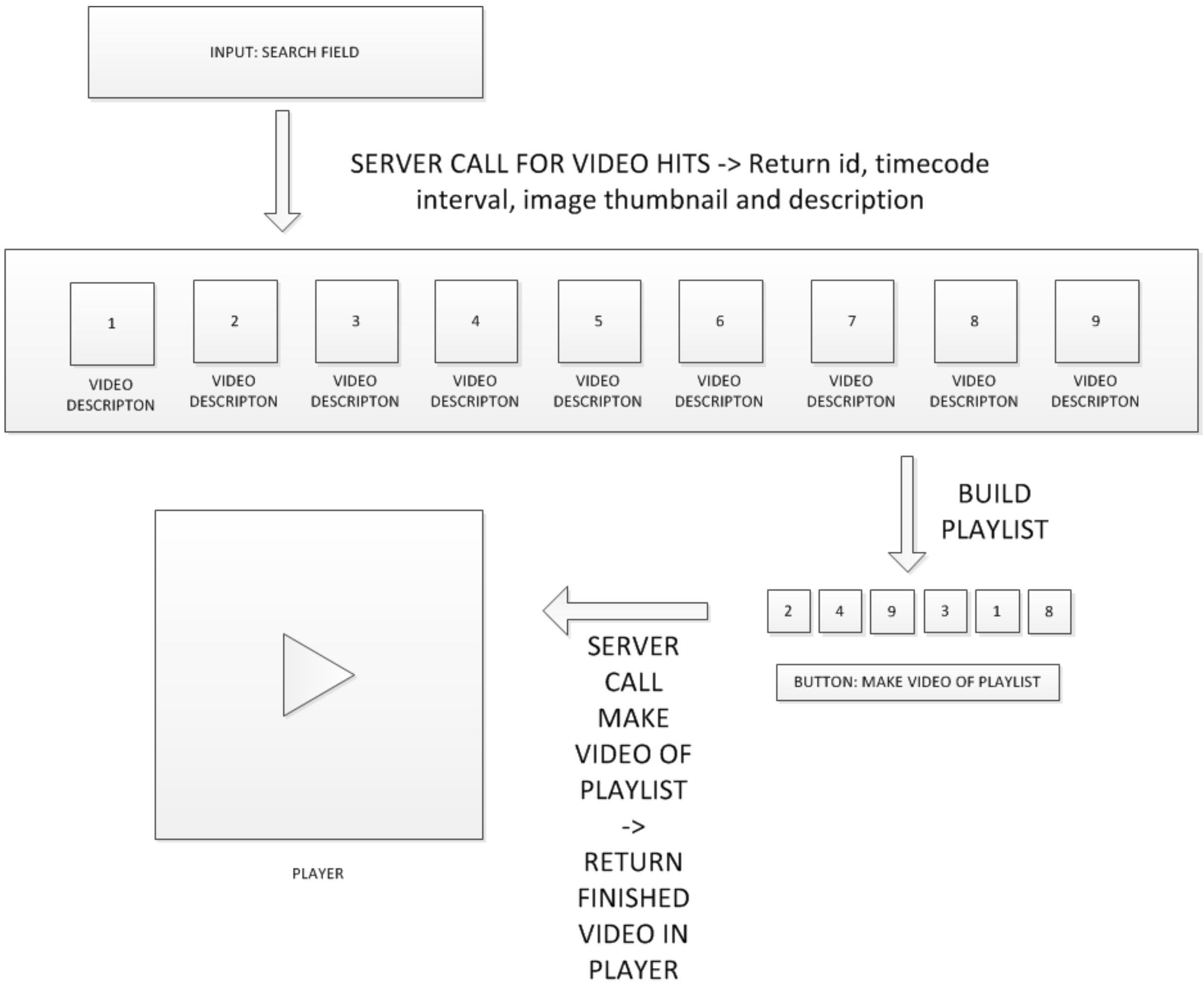

INPUT: SEARCH FIELD
SERVER CALL FOR VIDEO HITS -> Return id, timecode interval, image thumbnail and description
1 2 3 4 5 6 7 8 9
VIDEO DESCRIPTON
BUILD PLAYLIST
2 4 9 3 1 8
BUTTON: MAKE VIDEO OF PLAYLIST
SERVER CALL MAKE VIDEO OF PLAYLIST -> RETURN FINISHED VIDEO IN PLAYER
PLAYER

Figure 3.4: Sketch of user interface

To achieve this, we have decided to use the Jquery UI library [66]. This user interface library supports a lot of useful user interaction tools, including functionality such as drag and sortable drops. We use the jquery-UI-1.10.4 version. Jquery UI is on top of the Jquery library [65] and we use the compatible jquery-1.10.2 version.

Our last test used different manifest files to play specific parts of the "Goldfinger" movie. We will first try to create a list of draggable items that can be dropped in the video player. When this happens, the player will start to play the scene in the movie that the draggable item represents. In Jquery it is possible to assign most HTML tags draggable or droppable functionality. The video tag will be assigned the droppable functionality. The draggable item needs to reappear in its original position after dropping it into the video player. An end user can then preview a movie scene multiple times. Here is an example of a draggable clip:

```
$( "#astonmartin" ).draggable(
{
connectToSortable: "#playlistDroppable",
helper: "clone",
revert: "invalid"
});
```

The "connectToSortable" parameter enables an end user to rearrange clips within the droppable area. The combination of the helper and revert parameter enable a clip to be dropped from its original position multiple times.

Using these interface tools, we are now able to do approximately what the figure 3.4 illustrates. For starters, we will use the predefined playlist sequences we created in our last attempt. Developing searchable topics will happen later in the process. Next, we need to find out if a manifest file can be dynamically created.

3.3 Dynamic manifest

We have already learned that altering the video tags "src" attribute with a valid manifest URL is enough to playback video segments. We now need to consider what kind of information is necessary to be sent between a client and the server. As a minimum, a client needs to tell the server the beginning and end of a video segment range. The server also need to know in which folder the segments are located. We have created a javascript object containing such information.

```
var movie =
{
"id"   :  "movie",
"start" :  "0",
"end" :  "631",
};
```

The key value "id" indicates a folder where the video segments are stored. This object information is hard coded in the javascript file. This is only for testing purposes. Later, our workflow will use HTTP GET requests containing a query. The server will return data found in a relational database using a JSON [67] object. The client will parse this data as a javascript object, and use it as a reference when a user wants to playback a movie sequence. Necessary sequence data will then be sent as a new GET request to the server, and a PHP script will generate a manifest file. The URL of this manifest file is afterwards returned to the client, and used by the video player to playback the manifest.

A manifest file has a certain structure and require some necessary parameters. Consider this example:

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-ALLOW-CACHE:YES
#EXT-X-TARGETDURATION:20

#EXTINF:12.360000,
gfsegments/gf00000.ts
#EXTINF:10.000000,
gfsegments/gf00001.ts

#EXT-X-ENDLIST
```

Lines in a manifest including the command #EXTINF references video segments. They include information about the exact duration of the segmented video and its file path. This information has to be dynamically added. For testing purposes, we rewrite an existing manifest file. This file will be overwritten every time an end user sends a new manifest request.

A manifest referencing video segments from the same video resource uses the same codecs and quality. If the manifest contains references from multiple video resources, this might not be the case. For such purposes, the manifest file needs some extra information.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-ALLOW-CACHE:YES
#EXT-X-TARGETDURATION:20

#EXTINF:12.360000,
gfsegments/gf00000.ts
#EXT-X-DISCONTINUITY
#EXTINF:10.000000,
rwlsegments/rwl00010.ts

#EXT-X-ENDLIST
```

Notice the line containing the command #EXT-X-DISCONTINUITY. It is located between references to two different video segments. This command notifies the video player to expect that the next video segment can be of a different resolution or have a different bit rate than the last segment. "If the videos are all encoded with the same resolution, codecs, and profiles, then this tag can be left out" [213].

3.3.1 Relational database

There are several approaches to dynamically create manifests. We can parse the movie manifest when we search for a matching segment to get the necessary video segment reference. Alternatively, we can create tables in a relational database containing the required data. A relational database has a set of rules and can improve performance in the system. The MySql [84] relational database is included in the XAMPP [206] package. XAMPP also include a "phpMyAdmin" web user interface with access to configure a database and run MySql queries.

We have written a MySql script containing the tables we need for the prototype. This script will be rewritten with additional columns or tables when necessary. We can configure small changes using PHP scripts or via the "phpMyAdmin" web user interface. For large changes we can rebuild the database using the MySql script.

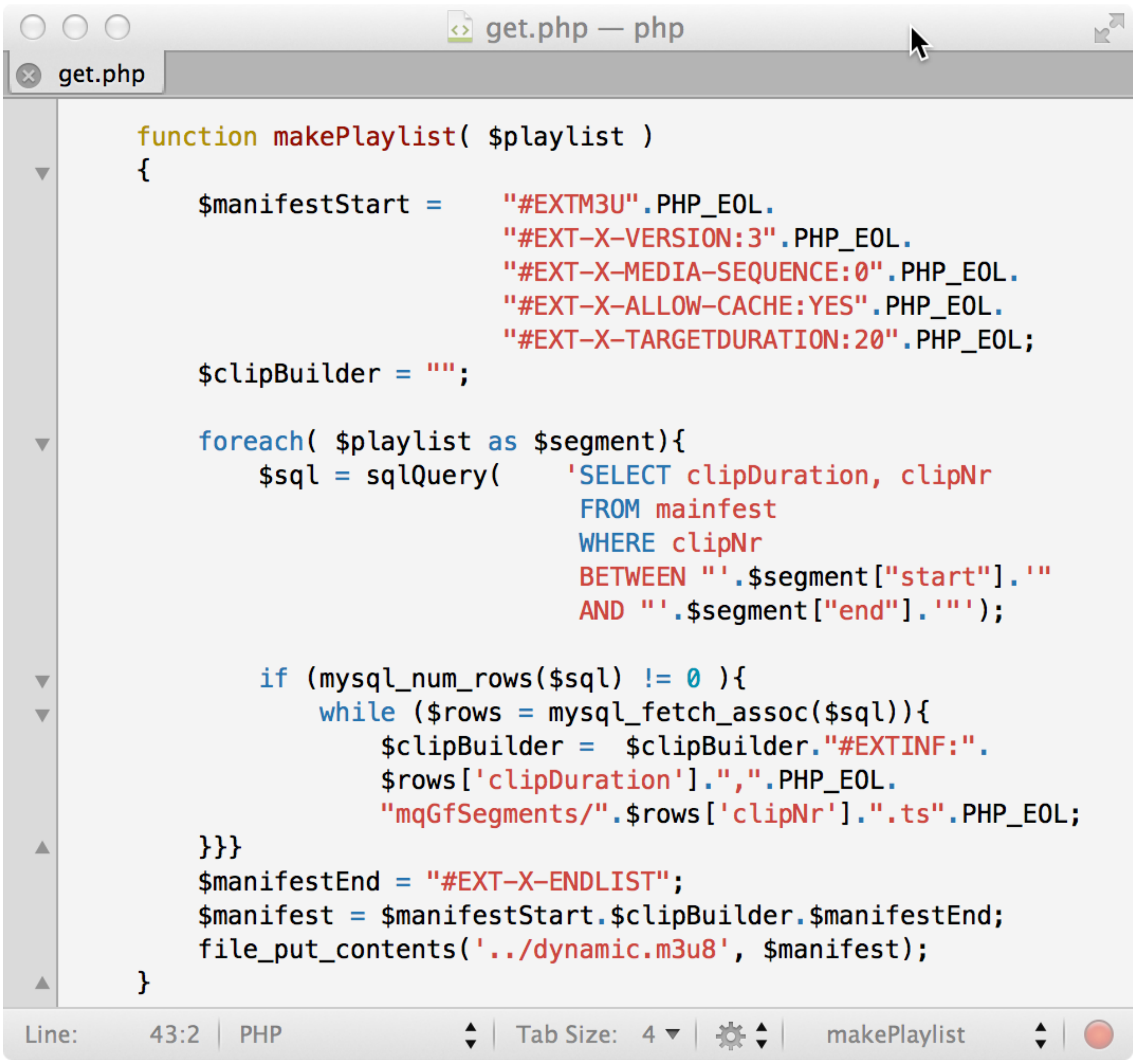
A manifest file consists of a beginning of necessary commands, a series of referenced segments and an end. For now, we only need to include the duration and file name of each video segment in a MySql table. For later versions, we will add more columns to this table. The first version of the manifest table looks like this:

```
CREATE TABLE `mainfest` (
  `clipId` int(10) unsigned NOT NULL auto_increment,
  `clipDuration` DECIMAL(8,6) UNSIGNED NOT NULL,
  `clipNr` varchar(5) NOT NULL,
  PRIMARY KEY  (`clipId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

The column "clipId" is an unique number for each row in the table. Hence, it is also the primary key of the table. The column "clipDuration" extracts for example the substring "12.360000" from the manifest reference line "#EXTINF:12.360000,gfsegments/00000.ts". The column "clipNr" extracts "00000.ts" from the same reference line. We use regular expressions when parsing a manifest file to extract the correct text string.

3.3.2 The make playlist function

We have already created a user interface, client-server connectivity and a database table containing the extracted manifest file references. We need to write a function that retrieves the data from the database table, creates a manifest based on these data, and returns a playable URL to the client. The illustration in figure 3.5 shows a function that rewrites a manifest file named "dynamic.m3u8":



```
function makePlaylist( $playlist )
{
    $manifestStart =    "#EXTM3U".PHP_EOL.
                        "#EXT-X-VERSION:3".PHP_EOL.
                        "#EXT-X-MEDIA-SEQUENCE:0".PHP_EOL.
                        "#EXT-X-ALLOW-CACHE:YES".PHP_EOL.
                        "#EXT-X-TARGETDURATION:20".PHP_EOL;
    $clipBuilder = "";

    foreach( $playlist as $segment){
        $sql = sqlQuery(    'SELECT clipDuration, clipNr
                             FROM mainfest
                             WHERE clipNr
                             BETWEEN "'.$segment["start"].'"
                             AND "'.$segment["end"].'"');

        if (mysql_num_rows($sql) != 0 ){
            while ($rows = mysql_fetch_assoc($sql)){
                $clipBuilder =  $clipBuilder."#EXTINF:".
                $rows['clipDuration'].",".PHP_EOL.
                "mqGfSegments/".$rows['clipNr'].".ts".PHP_EOL;
    }}}
    $manifestEnd = "#EXT-X-ENDLIST";
    $manifest = $manifestStart.$clipBuilder.$manifestEnd;
    file_put_contents('../dynamic.m3u8', $manifest);
}
```

Figure 3.5: Function for creating a manifest playlist

The function receives a video segment object from the client as input. This object contains the hardcoded start and end values. We can use these values to query a selection from the database table. The result is concatenated into a large string. This is the part referencing video segments in the dynamic manifest. We have also hardcoded the necessary start and end commands to make this manifest playable. All three strings are finally concatenated into one large string. This string is then used to overwrite the content of the "dynamic.m3u8" manifest file. Afterwards, we return the URL of the manifest file to the client. For this version of the prototype, we reuse the same URL. The important part is to set the video tag "src" attribute with the updated manifest URL. Since the client player uses the "autoplay" attribute, the player will instantly begin playback of the new version of the manifest.

It is now possible to drag and drop video segments from the search result area directly into the video player or into the playlist area. Figure 3.6 illustrates this user interface.

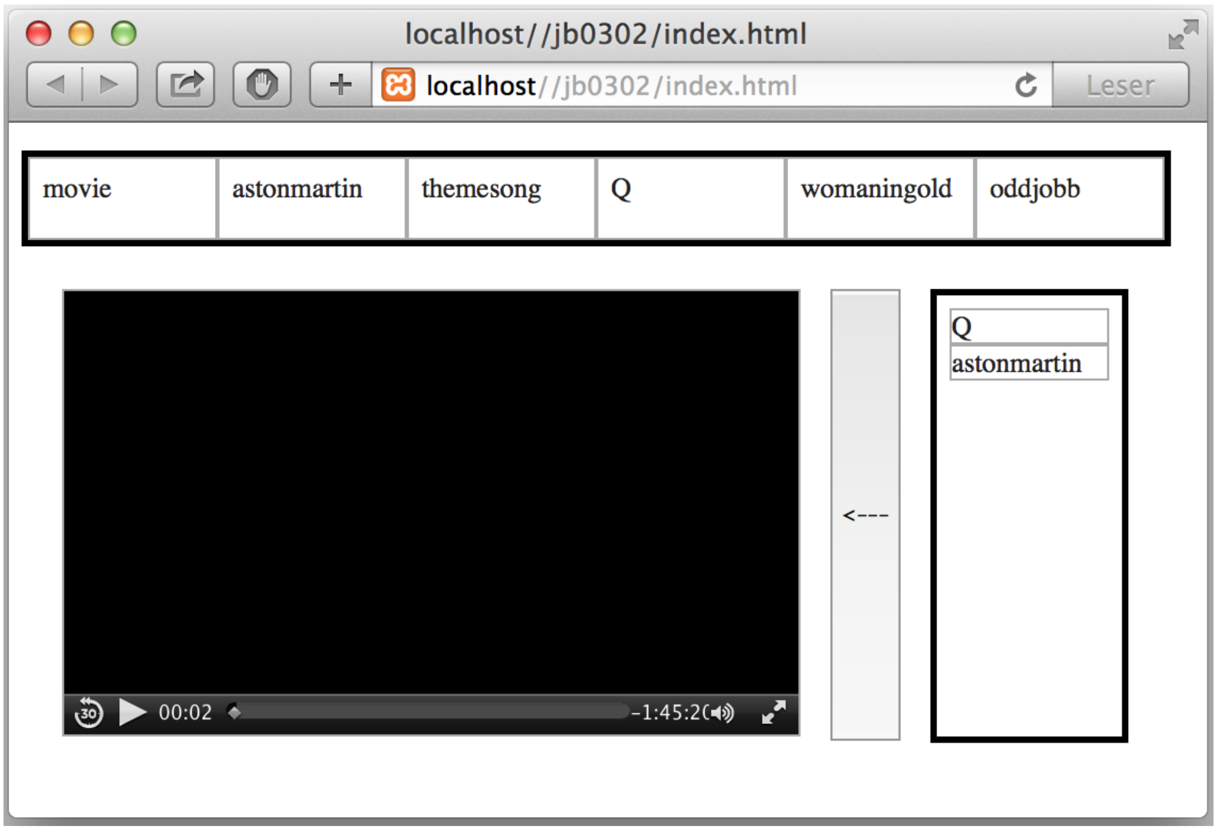


Figure 3.6: User interface with drag and drop functionality

Video segments within the playlist area can also be dragged into the video player. Dragging the video segment somewhere else outside the playlist area, will remove it from the playlist area.

When pressing the "<—" button, a javascript function sends a HTTP GET request to the server with the segment range information. Then the "makePlaylist" function overwrites the manifest file with the new video segment references. When dragging a video segment directly into the player, the same function is used to create a new manifest.

We have proven that it is possible to create a dynamic playlist. Next, we will optimize the segmentation process.

3.4 Encoding and segmentation

We used FFmpeg for transcoding and segmentation of the video resource. FFmpeg requires a lot of input parameters to deliver a satisfactory result. We created a workable manifest after some trial and error. However, the input parameter "segment-time 10" did not deliver the expected duration outcome. The video segments had variable duration. Here is a sample from the manifest file:

```
#EXT-X-TARGETDURATION: 20
#EXTINF:17.440000,
gfsegments/00002.ts
#EXTINF:3.040000,
gfsegments/40415.ts
```

#EXT-X-TARGETDURATION has a duration of 20. This specifies the maximum video segment duration in this manifest. The command tag #EXTINF describes the duration of a video segment. They span from approximately 3 to 17 seconds in this sample. It seems the FFmpeg parameter "segment-time 10" represents an average rather than a fixed segment duration. We want to calculate exact locations of video segments within a movie to extract accurate events. Hence, we need video segments of a fixed duration.

3.4.1 Using the mediafilesegmenter tool

Apple's "mediafilesegmenter" is a command-line tool much like FFmpeg. It receives input parameters and can transcode certain formats into video segments. It supports parameters for encryption and creating a variant playlist. The tool also supports byte-ranges, allowing "you to consolidate you media segments into larger files or a single large file [18]. It also possible to create segments of a chosen duration, even though Apple recommend segment durations of 10 seconds. "Here's why: if you've got live content being delivered through a CDN, there will be propagation delays, and for this content to make it all the way out to the edge nodes on the CDN it will be variable. In addition, if the client is fetching the data over the cellular network there will be higher latencies. Both of these factors make it much more likely you'll encounter a stall if you use a small target duration" [17].

We are particularly interested in video segments of shorter durations. A subtitle text has typically a duration of 2-6 seconds. Hence, it is practical to have video segments of shorter duration than 10 seconds. Otherwise, there will be unnecessary video before and after that specific dialogue in the movie. It is also easier to implement video handles [19] using shorter durations. We can simply add or remove a video segment from the default video segment range.

Issues like propagation delay is important in a web application. However, we want a fixed video segment duration of 2 seconds in order to locate accurate video segment ranges. Other adaptive streaming protocols like Smooth Streaming [75, 93, 211] and prototypes like Davvi [63, 64] also uses a default segment duration of 2 seconds.

Our main concern is creating video segments of a fixed duration. According to Apple's technical note about the "mediafilesegmenter" tool, "the "mediafilesegmenter" takes media from the specified file, multiplexes it into MPEG-2 Transport streams or elementary streams, and divides it into a series of small media files of approximately equal duration" [17]. Hopefully, this means we can create a fixed video segment duration from this tool. The tool's manual describes a series of input parameters we can choose from. The following example is a list of parameters that we have experimented with.

```
-t | -target-duration duration (segment length)
-f | -file-base path (where to save segments and manifest)
-i | -index-file fileName (name of manifest)
-B | -base-media-file-name name (segment name)
-z | -iframe-index-file name (option none means no iframe file)
-start-segments-with-iframe (every segment begin with iframe)
```

We used the same movie resource as with FFmpeg to compare the different tools. Note that we are only interested in the workflow of creating video segments and a manifest file from for example an .mp4 file. The "mediafilesegmenter" tool and FFmpeg have other functionalities that are not comparable. Apple's tool require source media files containing H.264 [145] video or AAC/HE-AAC [123], AC-3 [107] or MP3 [162] audio. Hence, not as flexible as FFmpeg, but efficient in its purpose. Remember the main purpose of this tool is to segment the files to the right format, not to alter aspect ratio or picture quality. We tested the tool with two input file formats and used these input parameters:

```
sudo mediafilesegmenter
-t 2
-i goldfinger.m3u8
-B gf
-f segmentedGoldfinger
sourcMovies/goldfinger.mov (.mp4)
```

This workflow seems promising at first glance. All video segments and the mainfest file are created in the target folder "segmentedGoldfinger". Most importantly, the tool seems to create video segments of the same duration. However, not all video segments can be played individually. During the segmenting process, the tool outputted several times that the "segment does not contain sync frame". We looked at the first 50 segmented .ts files and discovered that only a few contained both video and audio, being approximately 25 percent of the media segments. The rest of the media segments played only audio. So how does this work? The "mediafilesegmenter" also created an I-frame manifest file. The beginning of the I-frame manifest file looks like this.

```
#EXTM3U
#EXT-X-TARGETDURATION: 2
#EXT-X-MEDIA-SEQUENCE: 0
#EXT-X-PLAYLIST-TYPE: VOD
#EXT-X-I-FRAMES-ONLY
#EXTINF: 8.60000,
#EXT-X-BYTERANGE:
1128@376
gf0.ts
#EXTINF: 17.32000,
#EXT-X-BYTERANGE:
3196@4888
gf4.ts
```

Notice the duration of the I-frame [197] video segments. They vary from approximately 8 to 17 seconds. The name of the video segments corresponds with the media segments being able to play video and audio. This is because these segments includes reference I-frames. MPEG-2 [163] is a compression format and reduces picture information in many frames to make the media smaller in size [143]. Hence, this technique reduces bandwidth used and storage space.

I-frame byte range offsets are impractical when creating dynamic manifests. It is easier to concatenate video segments that play individually. We changed the input parameters for our next test using the "mediafilesegmenter" tool:

```
sudo mediafilesegmenter
-t 2
-i goldfinger.m3u8
-B gf
-f segmentedGoldfinger
-z none
-start-segments-with-iframe
sourcMovies/goldfinger.mov (.mp4)
```

We have added the parameter "-z none". This tells the tool to not create an I-frame manifest file, and that all segments will start with an I-frame. During the segmenting process, the tool outputted several times "Key frame distance exceeds target duration; target duration will be adjusted". The output directory now only contain one mainfest file.

```
#EXTM3U
#EXT-T-TARGETDURATION: 20
#EXT-X-VERSION: 3
#EXT-X-MEDIA-SEQUENCE: 0
#EXT-X-PLAYLIST-TYPE:VOD
#EXTINF:8.60000,
gf0.ts
#EXTINF:17.32000,
gf1.ts
```

This sample from the manifest file looks like the FFmpeg manifest file. The maximum duration is now set to 20. However, the #EXTINF command includes video segment durations identical to the byte range referenced in the I-frame manifest file. Each segment includes both video and audio, but some of the audio is muted after a few seconds.

3.4.2 How to create video segments with a fixed duration

We have successfully created individual video segments from the FFmpeg tool. However, these video segments does not have a fixed duration. Since our workflow does not output segments of equal duration, we need to research other input parameters available in FFmpeg. When encoding video, it is possible to define how many I-frames a movie should consist of. Many I-frames means higher quality, but also larger media files. Since we want video segments of 2 seconds duration, we will now try to create an I-frame for each video segment. FFmpeg supports encoding I-frames at different intervals. We tried to segment a movie resource using three extra input parameters.

```
./ffmpeg -i ./goldfinger.mov
-c:a libmp3lame
-ar 48000
-ab 64k
-c:v libx264
-b:v 96k
-flags
-global_header
-map 0
-f segment
-segment-time 2
-segment_list movie.m3u8
-segment_formats mpegts gfsegments/%05d.ts
-sc_threshold 0 (detect scene changes, forces I-frames)
-keyint_min 50 ( minimum frames between I-frames )
-g 50 (maximum frames between I-frames)
```

We have set the "scene change threshold" parameter to tell FFmpeg to ignore scene changes within the movie when defining I-frames. Our implementation of scene-based searches will be based on metadata documents, not I-frames within a movie. Since an encoded movie probably uses more I-frames than there are chapters, it would anyway be difficult to differentiate a scene-based I-frame from other I-frames within the movie.

The "keyint" and "g" parameter sets the minimum and maximum limits for each interval of I-frames within the movie. They could have been set to 48 considering a movie traditionally is shot with 24 frames per second [140]. We need to consider the video resource when deciding this metric. For this test movie, we found that the video resource was set to the European Phase-alternating line (PAL) [172] standard which is shot with 25 frames per second. However, setting the I-frame interval to 48 or 50 will not have a profound impact on our search accuracy. Tools such as Shotdetect [98] or FFprobe [32] can be used to determine exactly how the video and audio is encoded.

A sample of the created manifest file using these new parameters looked like this:

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-ALLOW-CACHE:YES
#EXT-X-TARGETDURATION:3
#EXTINF:2.080000,
goldfinger/00000.ts
#EXTINF:2.000000,
goldfinger/00001.ts
#EXTINF:2.000000,
goldfinger/00002.ts
#EXTINF:0.640000,
goldfinger/03161.ts
#EXT-X-ENDLIST
```

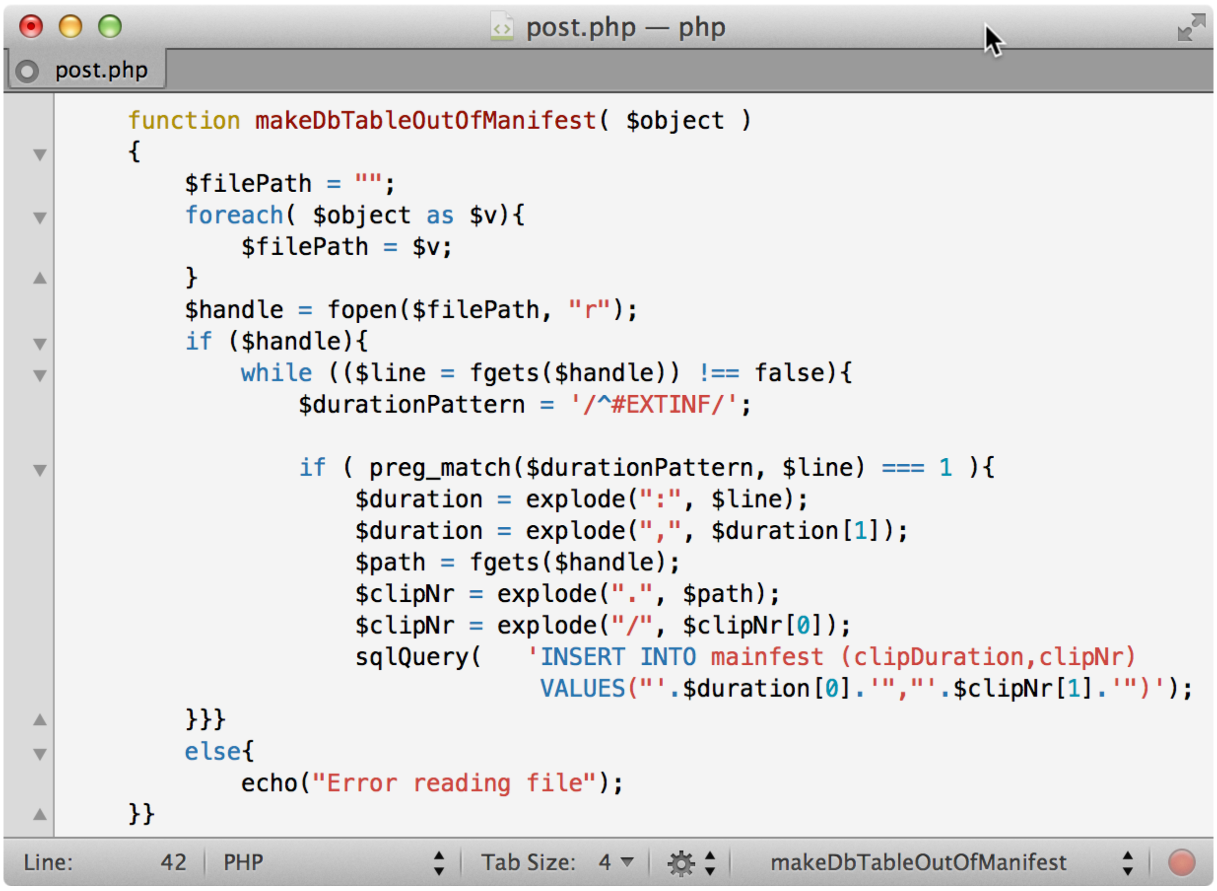
Note that the command #EXT-X-TARGETDURATION is now reduced from 20 to 3. By changing the maximum and minimum frequency of I-frames in the movie, FFmpeg will create video segments of approximately the same duration. We set the "segment-time" parameter to 2 seconds. It is only the first and last #EXTINF command duration that alternates from exactly 2 seconds duration. Most importantly, each media segment plays individually with both video and audio. Achieving a fixed segment duration, it is now possible to calculate when a required video segment begins within a movie. Next, we will add a search engine to the system.

3.5 Metadata and search engine

So far we have used hardcoded data in a javascript file representing different scenes in a movie. The next step is to make the prototype searchable. This requires extraction of searchable data from metadata resources, and search functionality in the user interface. In the following sections we will present how this is done with subtitle and chapter metadata.

3.5.1 Extracting data from metadata resources

We have developed another web user interface for parsing and extracting metadata to the database. This backend interface is relatively simple in structure. Further development of the system might require more functionality and error handling. The interface has basically an input field and a button. When pressing the button, a javascript file reads an input text with a relative path to a metadata resource file on the server. This file path is sent via a HTTP POST request to a PHP script that locates the file and parses it. Regular expressions are used to extract the correct text strings. This illustration in figure 3.7 shows a function that writes extracted data to a database table.



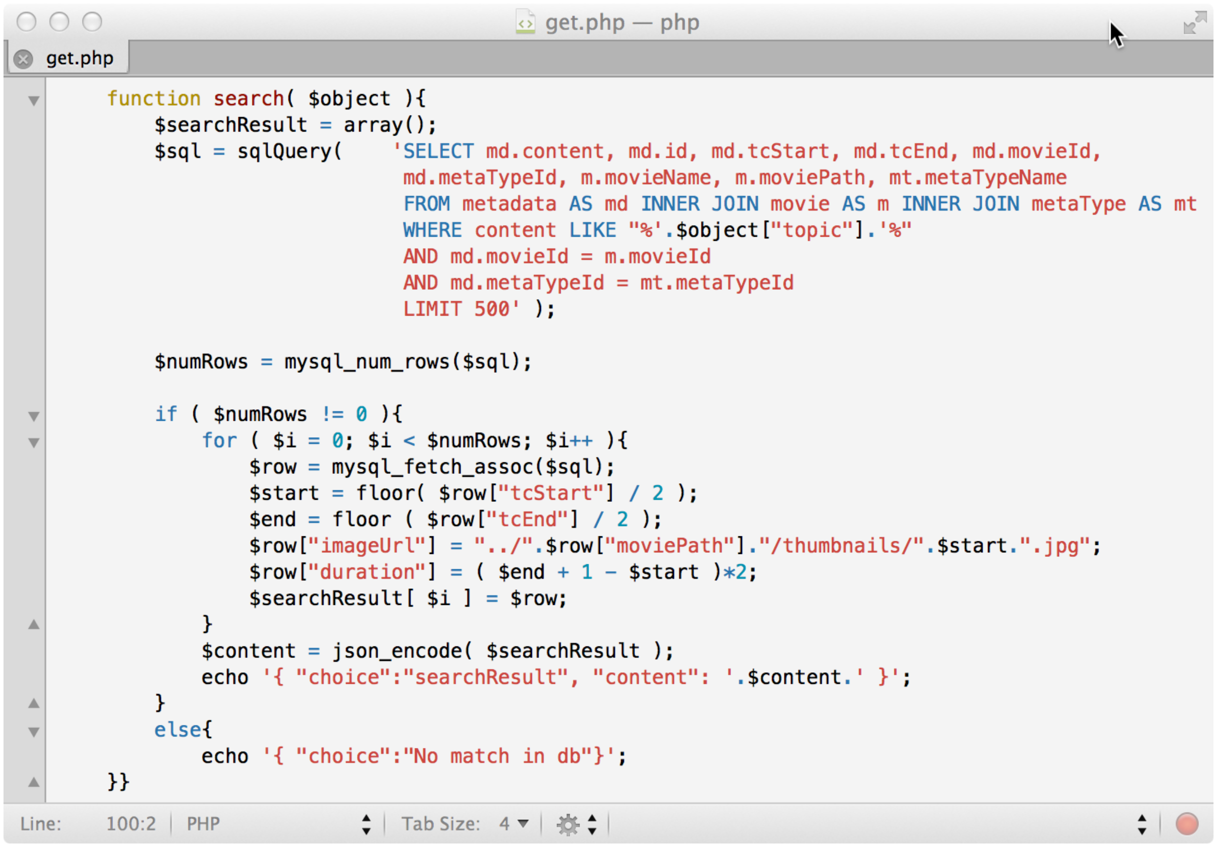
```
function makeDbTableOutOfManifest( $object )
{
    $filePath = "";
    foreach( $object as $v){
        $filePath = $v;
    }
    $handle = fopen($filePath, "r");
    if ($handle){
        while (($line = fgets($handle)) !== false){
            $durationPattern = '/^#EXTINF/';

            if ( preg_match($durationPattern, $line) === 1 ){
                $duration = explode(":", $line);
                $duration = explode(",", $duration[1]);
                $path = fgets($handle);
                $clipNr = explode(".", $path);
                $clipNr = explode("/", $clipNr[0]);
                sqlQuery(   'INSERT INTO mainfest (clipDuration,clipNr)
                             VALUES("'.$duration[0].'","'.$clipNr[1].'")');
    }}}
    else{
        echo("Error reading file");
}}
```

Figure 3.7: Function for inserting manifest data into table

This function receives a file path as input and tries to open the file. If successful, the function parses each line of the file and compares it with a regular expression [179]. If a line begins with #EXTINF, we extract the video segment duration. Since we have researched the structure of a manifest file, we know that the next line in the file contains the video segment file name. Therefore, we also read the next line in the file for each regular expression match. The segment file name and duration are inserted as a new row in the manifest table.

3.5.2 Search functionality

We will add a search bar to the user interface. This search functionality is based on wildcards [203]. If a user types something in the search bar and presses a button, all rows matching that word or sentence will be returned from a table in the database. Figure 3.8 illustrates the search function used for text-based queries.



```php
function search( $object ){
    $searchResult = array();
    $sql = sqlQuery(    'SELECT md.content, md.id, md.tcStart, md.tcEnd, md.movieId,
                         md.metaTypeId, m.movieName, m.moviePath, mt.metaTypeName
                         FROM metadata AS md INNER JOIN movie AS m INNER JOIN metaType AS mt
                         WHERE content LIKE "%'.$object["topic"].'%"
                         AND md.movieId = m.movieId
                         AND md.metaTypeId = mt.metaTypeId
                         LIMIT 500' );

    $numRows = mysql_num_rows($sql);

    if ( $numRows != 0 ){
        for ( $i = 0; $i < $numRows; $i++ ){
            $row = mysql_fetch_assoc($sql);
            $start = floor( $row["tcStart"] / 2 );
            $end = floor ( $row["tcEnd"] / 2 );
            $row["imageUrl"] = "../".$row["moviePath"]."/thumbnails/".$start.".jpg";
            $row["duration"] = ( $end + 1 - $start )*2;
            $searchResult[ $i ] = $row;
        }
        $content = json_encode( $searchResult );
        echo '{ "choice":"searchResult", "content": '.$content.' }';
    }
    else{
        echo '{ "choice":"No match in db"}';
}}
```

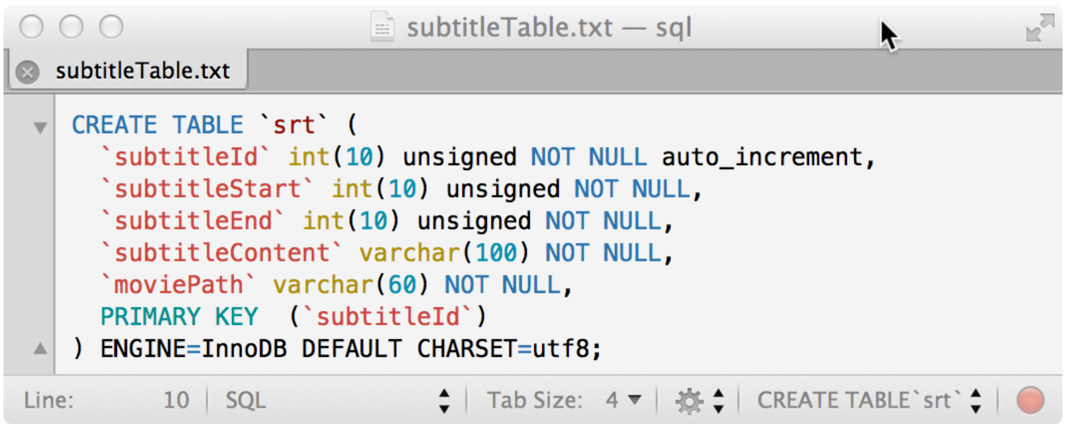
Figure 3.8: Search function

The matching rows include dialogue and scene-based descriptions in movies. For now, this functionality is not ranked [183], and might result in some irrelevant search results.

We will also add a category based search alternative. This will be a list of different categories representing movies in the database. When a user chooses a category, then relevant movies will appear in another list. Choosing a movie will present a search result of the different scenes within a movie. Such a functionality gives a user an overview over the different movies available in the database, and a quick way to find a specific scene within a movie.

3.6 Extracting metadata from subtitle files

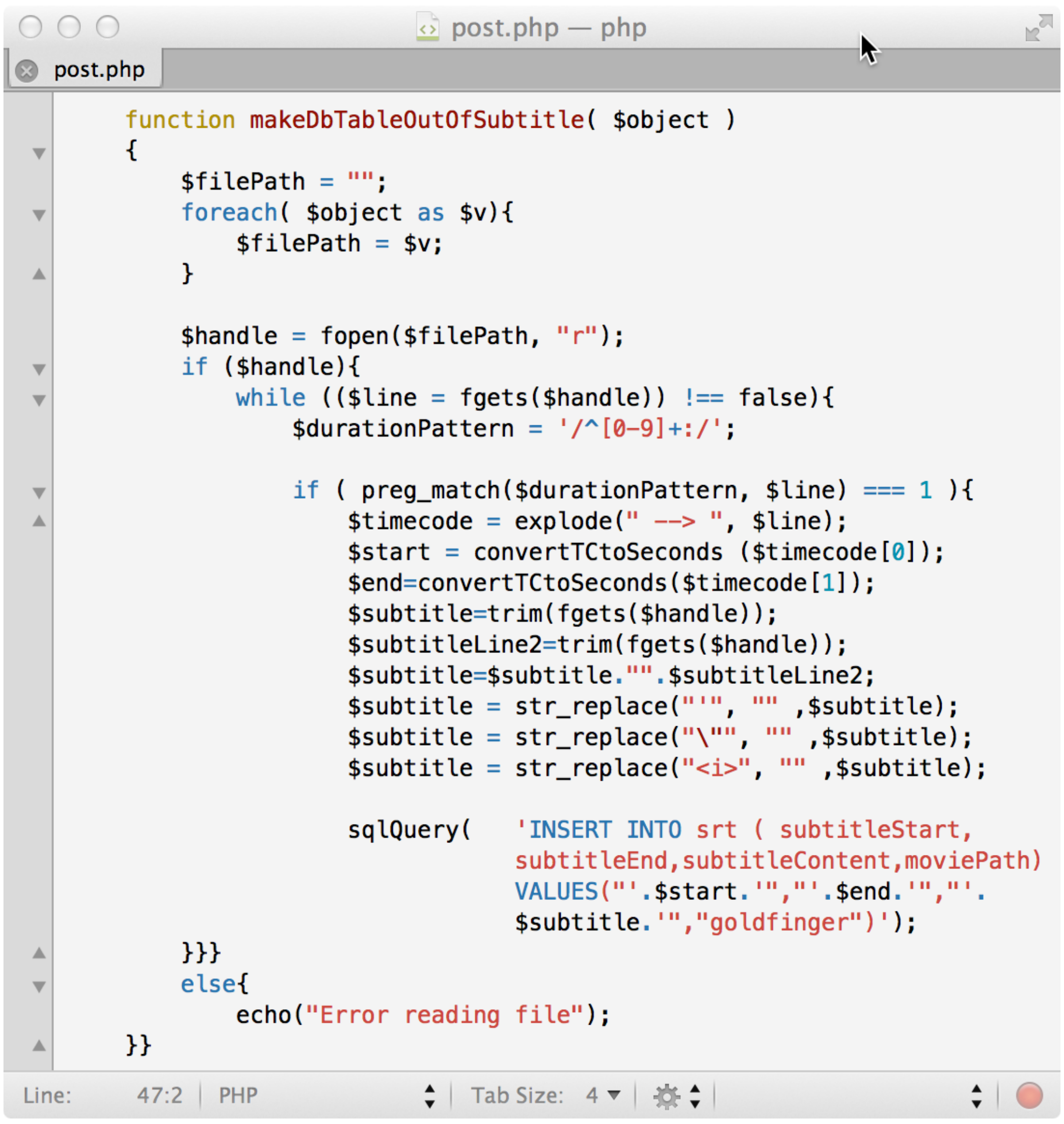
We will start with extracting metadata from a subtitle .srt file to a database table. It is important that the searchable event containing subtitle text and timecode, match the segmented video range in the system. First, we need a database table to insert extracted metadata from a .srt file. This table should have columns for subtitle content and a beginning and end index. The indexes are necessary to calculate the video segment range representing the subtitle event. We also need to know which movie the subtitle event belongs to. Figure 3.9 illustrates this database table.



```
CREATE TABLE `srt` (
  `subtitleId` int(10) unsigned NOT NULL auto_increment,
  `subtitleStart` int(10) unsigned NOT NULL,
  `subtitleEnd` int(10) unsigned NOT NULL,
  `subtitleContent` varchar(100) NOT NULL,
  `moviePath` varchar(60) NOT NULL,
  PRIMARY KEY  (`subtitleId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Figure 3.9: Subtitle table

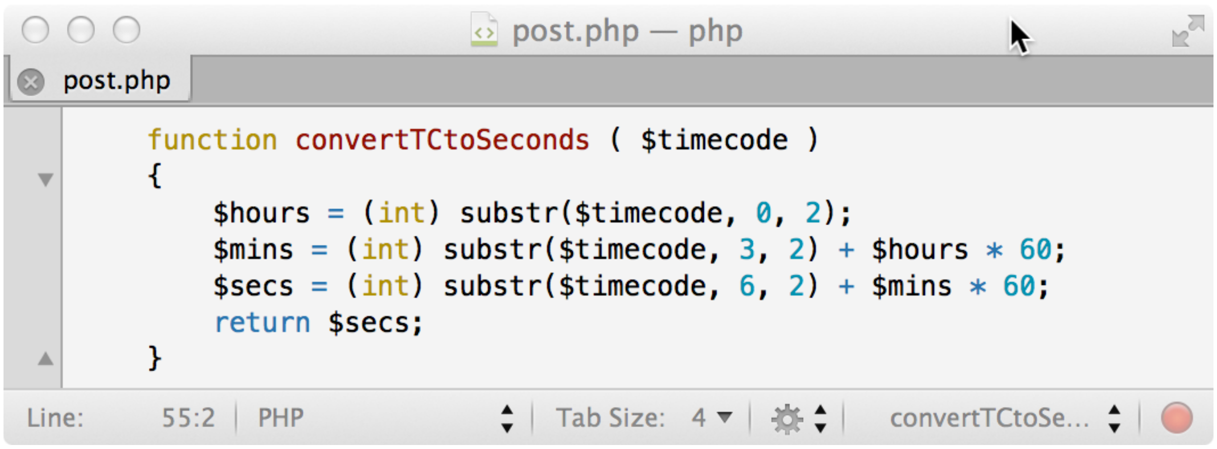
Next, we need to parse an .srt file and extract metadata to the this table. To achieve this, we use the same technique as when parsing mainfest files that was illustrated in figure 3.7. We type the file path to the .srt file in the backend user interface and a PHP script locates the .srt file on the server and parses it. Figure 3.10 illustrates this function.



```
    function makeDbTableOutOfSubtitle( $object )
    {
        $filePath = "";
        foreach( $object as $v){
            $filePath = $v;
        }

        $handle = fopen($filePath, "r");
        if ($handle){
            while (($line = fgets($handle)) !== false){
                $durationPattern = '/^[0-9]+:/';

                if ( preg_match($durationPattern, $line) === 1 ){
                    $timecode = explode(" --> ", $line);
                    $start = convertTCtoSeconds ($timecode[0]);
                    $end=convertTCtoSeconds($timecode[1]);
                    $subtitle=trim(fgets($handle));
                    $subtitleLine2=trim(fgets($handle));
                    $subtitle=$subtitle."".$subtitleLine2;
                    $subtitle = str_replace("'", "" ,$subtitle);
                    $subtitle = str_replace("\"", "" ,$subtitle);
                    $subtitle = str_replace("<i>", "" ,$subtitle);

                    sqlQuery(   'INSERT INTO srt ( subtitleStart,
                                subtitleEnd,subtitleContent,moviePath)
                                VALUES("'.$start.'","'.$end.'","'.
                                $subtitle.'","goldfinger")');
        }}}
        else{
            echo("Error reading file");
    }}
```

Figure 3.10: Function for extracting subtitle metadata

Each subtitle event within the .srt file starts with a number as described in subsection 2.16.4. Using a regular expression matching such lines, we are able to loop through each event. We create a variable for the start and end timecode. These two variables are then converted to seconds in a function illustrated in figure 3.11.



```
function convertTCtoSeconds ( $timecode )
{
    $hours = (int) substr($timecode, 0, 2);
    $mins = (int) substr($timecode, 3, 2) + $hours * 60;
    $secs = (int) substr($timecode, 6, 2) + $mins * 60;
    return $secs;
}
```

```
Line:    55:2 | PHP | Tab Size: 4 | convertTCtoSe...
```

Figure 3.11: Function for converting time codes to seconds

We also extract one or two lines of subtitle text for each loop. The text is concatenated and inserted into the "subtitleContent" column in the database table. Since subtitles may include special characters that can cause SQL injection [185], we remove such characters before insertion. This has a minor effect on text based searches. We also removed characters that signifies audio description for the hearing impaired. Support for such characters might be implemented on a later stage.

3.6.1 Mapping subtitles with video segments

Next, we need to map the "start" and "end" index in the database table with the correct video segments in the manifest. Time can be represented in many ways. We have made a comparison of how time is represented in the client player, the .srt file, the subtitle table and in the manifest file.

- Client player: 6:07 (minutes)
- .srt file: 00:06:07,295 (minutes)
- Subtitle table: 367 (seconds. 6*60 + 7 = 267)
- Manifest: 183 (video segment duration: 2 seconds. Hence, 367 / 2   183)

Matching subtitle event with video segments can be a challenge. If the conversion from time code to segment duration misses with a few seconds, then the system will not output the correct range of video segments. We need to be sure that our source material is correct. Next, we will investigate if there is any difference between .srt files.

3.6.2 Finding a matching .srt file

The file format .srt is originally the output of the subtitle extraction program "SubRip". "Using optical character recognition [171], SubRip can extract from live video, video files and DVDs [135], then record the extracted subtitles and timings as a Subrip format text file. [189]". .srt is a widely accepted container for caption text and timing information [191]. Commonly packaged with ripped DivX [134] movies, .srt files can be read and created by a variety of programs or found online [97]. Such programs pull subtitle text and timing information from a digital movie file.

"In practise, SubRip is configured with the correct codec for the video source, then trained by the user on the specific text area, fonts, style, colors and video processing requirements to recognize subtitles. After trial and fine tuning, SubRip can automatically extract subtitles from the whole video source file during its playback. SubRip records the beginning and end times and text for each subtitle in the output text .srt file" [189].

Meaning, the time codes within the file are only accurate to a movie of the same duration. We have downloaded a selection of .srt files based on the same movie from the websites Subscene [103] and Moviesubtitles.org [81], and compared the timecode offset between the subtitle files. Table 3.1 illustrates the result.

VLC Player TC	Donatello	HANGOVER	FLAWL3SS	FXG	iNCiTE	KISS	sUN	TEXT
6:08	6:23	6:15	6:13	6:14	5:58	6:14	6:15	Check.
6:12	6:26	6:19	6:17	6:24	6:02	6:23	6:19	Knight takes bishop.
6:17	6:32	6:25	6.23	-	6:07	-	6:25	Knight... takes bishop.
7:04	7:22	7:14	7:12	7:13	6:55	7:12	7:14	King to rook two.
7:06	7:24	7:16	7:14	-	6:57	-	7:16	King... to rook two.
7:19	7:36	7:29	7:28	7:29	7:10	7:28	7:29	Queen to king four.
7:22	7:40	7:32	7:30	-	7:12	-	7:32	Queen... to king four.
7:36	7:55	7:47	7:45	7:46	7:26	7:45	7:47	My congratulations, sir. A brilliant coup.
AVERAGE:	+17	+7	+8	+10	-10	+9	+10	

Table 3.1: Comparison of subtitle metadata resource accuracy

This comparison is based on subtitles from the James Bond movie "From Russia with love" [48]. We have compared a chess scene in the beginning of the movie. The first column shows a manual playback of the movie. We wrote down at approximately what time the actor said the same words as was presented in the subtitle file. The next columns are different versions of .srt files. Notice the last row listing the average difference between the manual playback and the .srt files. They vary from +17 to -10, almost half a minute. There are also small differences between each row of subtitles.

Why? These .srt files are probably ripped from different versions of the "From Russia with love" movie. For example, removing a few seconds in the beginning of the movie causes an offset for the reminder of the movie. It is also possible the subtitle editor wanted the subtitle text to be shown between cuts in the movie. Most movies are created in multiple frame rates [140] for different audiences. For example, frame rates are usually 24, 25 or approximately frames 30 per second. Recent movies like "The Hobbit: An unexpected Journey" was shot in 48 frames per second [96]. If a movie is shot with 24 frames and encoded with 25 frames, this will have an effect on the duration of the movie. Hence, a subtitle .srt file based on a 24 frame encoded movie will not entirely match a 25 frame encoded movie. We tried the same comparison with the James Bond movie "Goldfinger" [51]. The last subtitle time codes referenced 3 minutes after the entire duration of the movie. Such a large difference is probably because of different creative versions of a movie. A directors cut is usually longer than a producer cut movie. To map subtitle events with video segments accurately, we need to analyze the source material before extracting an .srt file to the database table. Optionally, we can rip our own .srt file from the source material. During the comparison of the "From Russia with love" .srt files, we also found a perfect match. We will use this .srt file to demonstrate that it is possible to map subtitle events accurately with video segment ranges.

3.6.3 Searching subtitle content

The next step is to include a search bar in the user interface of the system. The hard coded segments in the previous version of the system has been replaced with search results based on queries in the subtitle database table. A database query looks like this example:

```
sqlQuery('SELECT subtitleStart, subtitleEnd, moviePath, subtitleId,
subtitleContent FROM srt WHERE subtitleContent LIKE "%'.$topic.'%"');
```

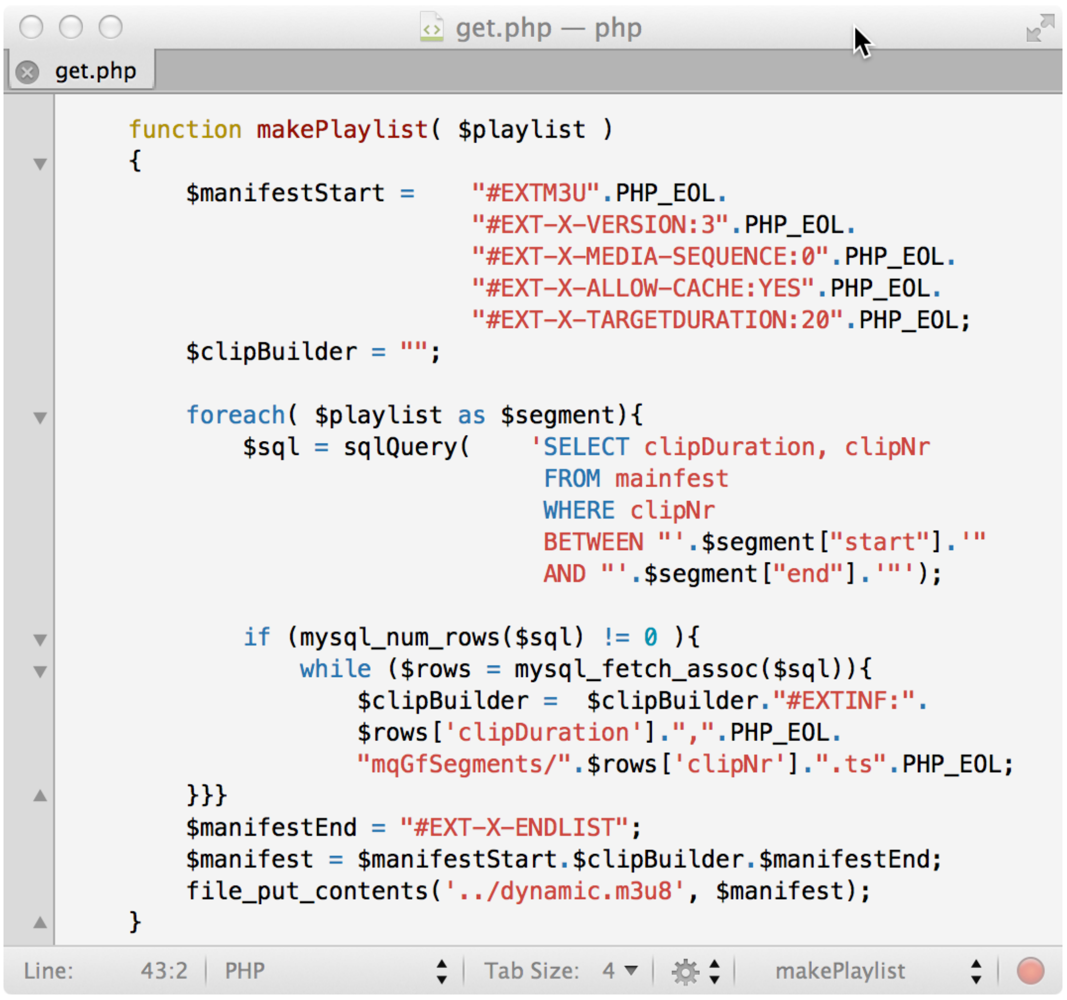
We only need the "subtitleStart", "subtitileEnd" and "moviePath" entities to create a dynamic manifest. The other entities are useful debugging content. All matches are returned to the client using JSON [67]. A sample JSON object can look like this example:

```
{"1": {
"subtitleId" : "56",
"subtitleStart" : "641",
"subtitleEnd" : "643",
"moviePath" : "russiawithlove",
"subtitleContent" : "...would be their agent James Bond."}}
```

Searching the word "Bond" in the "From Russia with love" movie, resulted in 20 hits. Another search word like "Russia" resulted in 25 hits. The word "be" resulted in "132" hits. Another common word "to" resulted in "186". Some searches provoked a syntax error. We solved this by converting the "subtitleContent" variable to UTF-8 [193] character encoding.

3.6.4 Calculating a dynamic manifest

We no longer need the manifest database table for creating a dynamic manifest file. Since we now can create video segments of a fixed duration, we only need the start and end index of an event, and the file path to the video segments. All video segments in between can be calculated, and necessary text information will be concatenated into the manifest file. Less queries to the database improves the performance of the system. Figure 3.12 illustrates this function.



```php
function makePlaylist( $playlist )
{
    $manifestStart =    "#EXTM3U".PHP_EOL.
                        "#EXT-X-VERSION:3".PHP_EOL.
                        "#EXT-X-MEDIA-SEQUENCE:0".PHP_EOL.
                        "#EXT-X-ALLOW-CACHE:YES".PHP_EOL.
                        "#EXT-X-TARGETDURATION:20".PHP_EOL;
    $clipBuilder = "";

    foreach( $playlist as $segment){
        $sql = sqlQuery(    'SELECT clipDuration, clipNr
                             FROM mainfest
                             WHERE clipNr
                             BETWEEN "'.$segment["start"].'"
                             AND "'.$segment["end"].'"');

        if (mysql_num_rows($sql) != 0 ){
            while ($rows = mysql_fetch_assoc($sql)){
                $clipBuilder =  $clipBuilder."#EXTINF:".
                $rows['clipDuration'].",".PHP_EOL.
                "mqGfSegments/".$rows['clipNr'].".ts".PHP_EOL;
    }}}
    $manifestEnd = "#EXT-X-ENDLIST";
    $manifest = $manifestStart.$clipBuilder.$manifestEnd;
    file_put_contents('../dynamic.m3u8', $manifest);
}
```

Figure 3.12: Function for calculating a manifest

We simply divide the start and end durations by two. These durations are in seconds, and we know that each video segment has a duration of two seconds. If a start or end value is an odd number, then we floor start values and round up end values. We ignored the millisecond values in the timecode when converting a .srt file into seconds. By rounding the start and end values, we compensate for some of this loss in accuracy. However, we do not know if a text query matches the beginning or the end of a subtitle event. Small increases to start and end values reduces some accuracy, but an end user experience might improve by increasing the chances they find what they are searching for.

We also experimented with querying two different movies in the database. A text query like "Bond" in both the "From Russia with love" and the "Goldfinger" movies, resulted in more subtile events. Unfortunately, we were unable to find a matching .srt file for our version of the "Goldfinger" movie. Meaning, some of our hits were off target.

We created a manifest containing references to both movies. The client player stopped playback when switching video segments from one movie to the other. We thought this was because the video segments had different file paths, but then we figured out that the two movies had different aspect ratios [126]. We added a #EXT-X-DISCONTINUITY command tag in the manifest file to compensate for this. This tag is inserted every time the manifest switches between segments of different movies. Our system can now play video from multiple sources and with different encodings. Figure 3.13 illustrates the current version of our system's user interface:

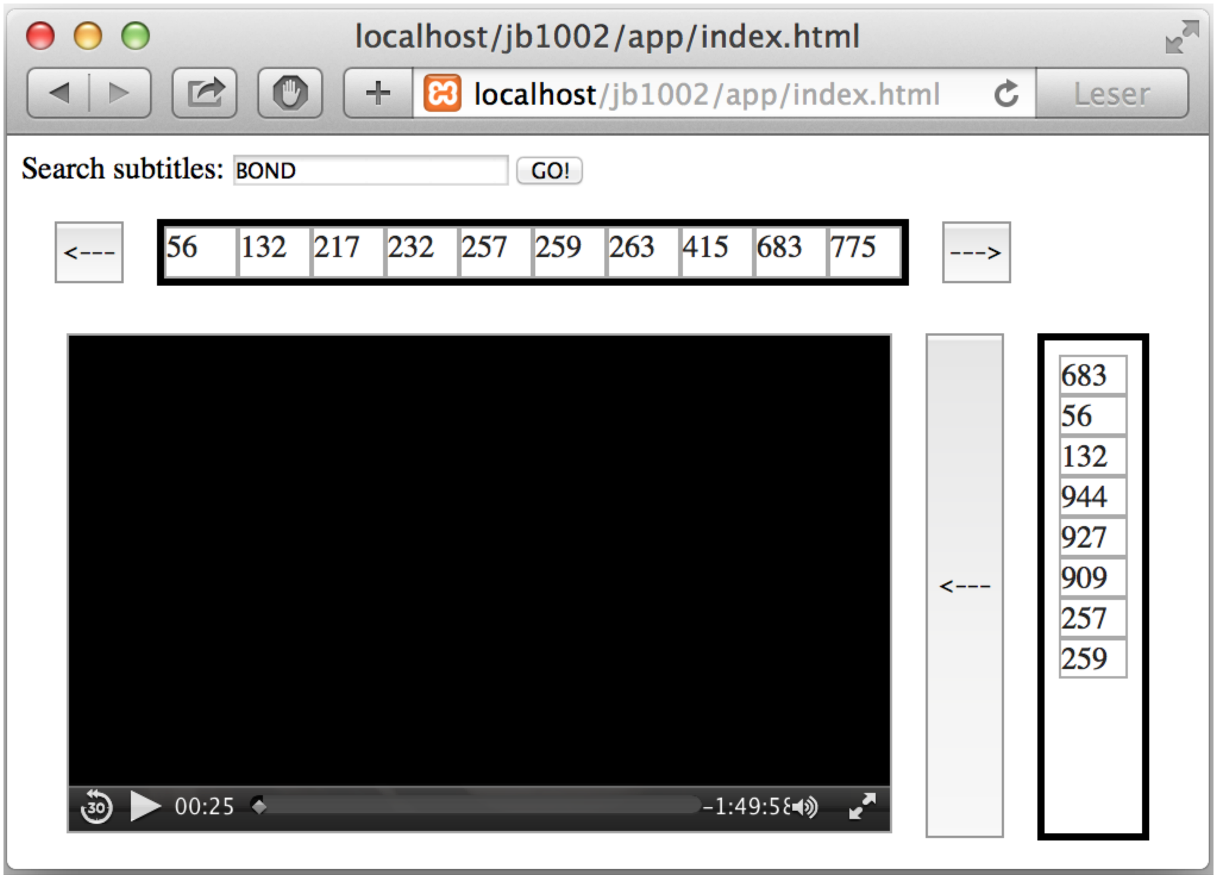


Figure 3.13: User interface with search functionality

3.7 Extracting metadata from chapter files

A system with only subtitle events has limited usefulness. We also want to insert chapter metadata events in the system. By using the chapterDb [22] website, we can download text files containing chapter information and time codes.

3.7.1 Frame rate and duration in a movie resource

It is important to first analyze the movie resource. We need the duration and the frame rate of the movie. FFmpeg can output a lot of information about a movie. We can limit this output by for example using a regular expression search with the command line program "grep" in a Linux shell.

```
./ffmpeg -i fromrussiawithlove.mp4 2>&1 | grep Duration
./ffmpeg -i fromrussiawithlove.mp4 2>&1 | grep fps
(frame rate per second)
```

If this information is included in the movie resource container, then these commands will output the information we need. For other video resources, these commends might need to be altered. Facing the same challenge as with subtitle metadata, we need to map chapter metadata events with a video segment range. In figure 2.22, we illustrated the search result of the movie "From Russia With Love". At the bottom of this web page these two values were listed:

- 25.0
- 01:50:16:280

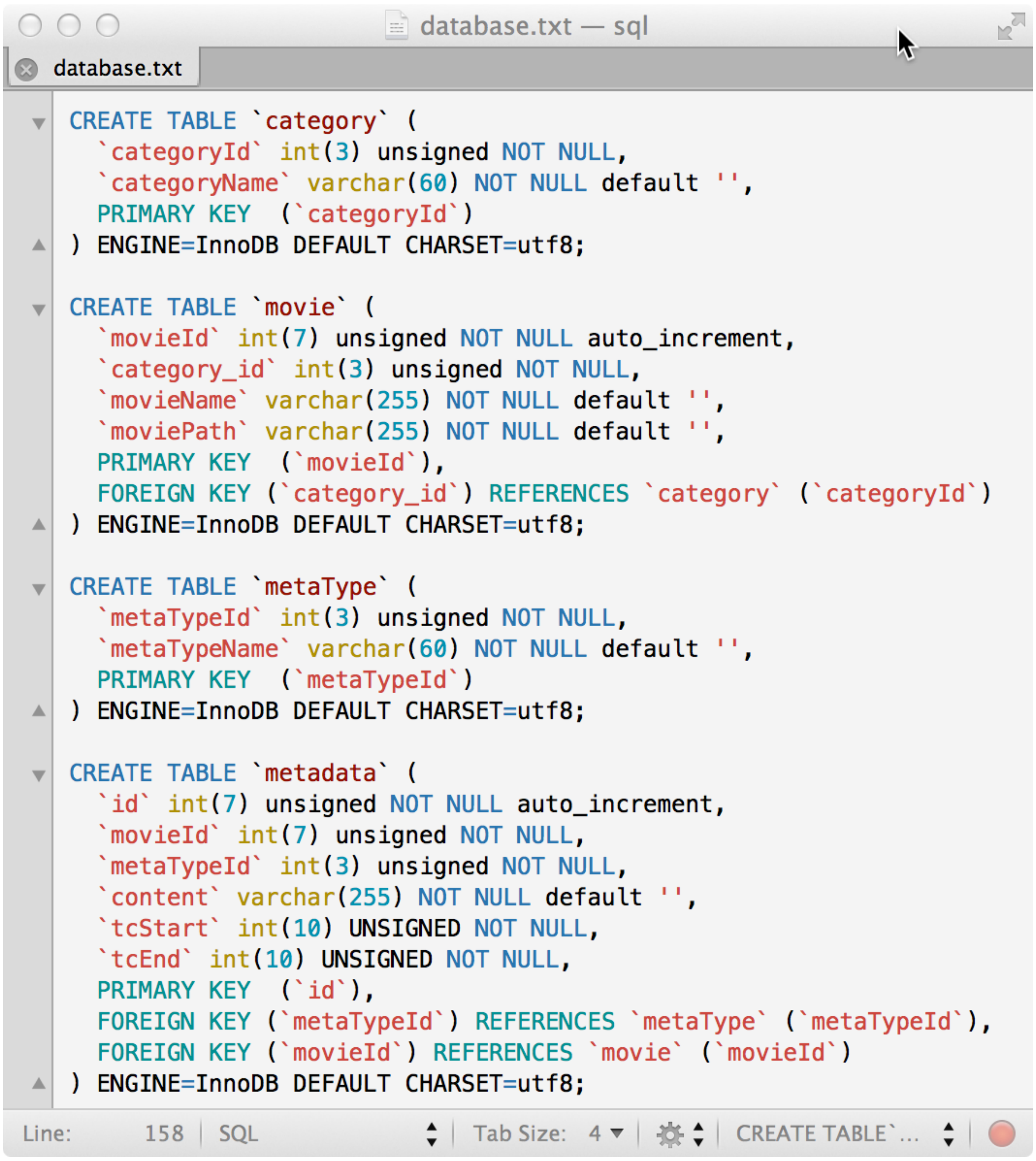
The number 25 indicates the frame rate per second of the movie this metadata is based upon. The timecode is the entire duration of the movie. If these values match the numbers extracted from the movie resource with FFmpeg, we can assume that these chapter metadata events will map approximately with our video segment ranges.

It is import to compare both values. We have already pointed out that correct frame rate is vital to map a metadata event accurately to a video segment range. The duration value ensures that we have the correct version of the movie. For example, another version might have removed some scenes because of cultural or political reasons in certain countries. In some cases a director shoots multiple versions of a scene to include a target market's actor. It is also normal to rerelease a movie as directors cut, including more scenes than the cinema release.

The website "chapterDb" include multiple versions of each movie. They are listed with small differences. Usually, different durations is a result of different frames per second. In some cases, the search result vary with one or multiple chapters as well.

3.7.2 Inserting chapter events into the database

We did some changes to the database to accommodate for different kinds of metadata in the system. The table "metadata" contain searchable text in the column "content". For example, this can be subtitle or chapter metadata events. The columns "tcStart" and "tcEnd" ensures the mapping between metadata text and a video segment range. Time codes are represented as integer values in seconds. It is also necessary to reference each row to a specific movie and metadata type. The "movie" table contain the movie name and in which folder the video segments are stored. The "metaType" table contain each metadata event type such as subtitles or chapters. The table "category" contain metadata on a higher level representing an entire movie. We have used IMDB's "Most popular by Genre" [54] as basis for different categories listed in this table. Figure 3.14 illustrates the new database structure.



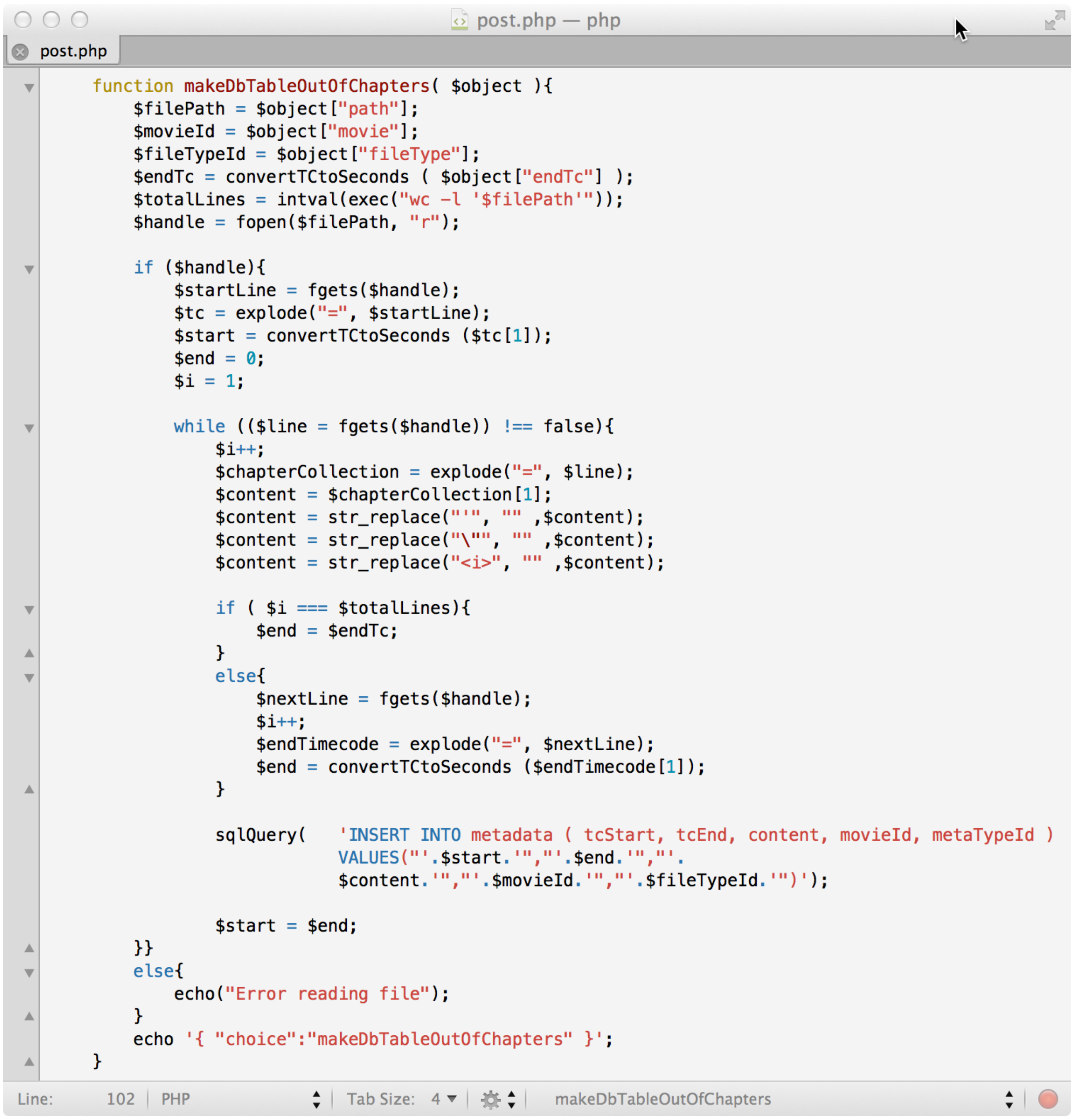
```
CREATE TABLE `category` (
  `categoryId` int(3) unsigned NOT NULL,
  `categoryName` varchar(60) NOT NULL default '',
  PRIMARY KEY  (`categoryId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `movie` (
  `movieId` int(7) unsigned NOT NULL auto_increment,
  `category_id` int(3) unsigned NOT NULL,
  `movieName` varchar(255) NOT NULL default '',
  `moviePath` varchar(255) NOT NULL default '',
  PRIMARY KEY  (`movieId`),
  FOREIGN KEY (`category_id`) REFERENCES `category` (`categoryId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `metaType` (
  `metaTypeId` int(3) unsigned NOT NULL,
  `metaTypeName` varchar(60) NOT NULL default '',
  PRIMARY KEY  (`metaTypeId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `metadata` (
  `id` int(7) unsigned NOT NULL auto_increment,
  `movieId` int(7) unsigned NOT NULL,
  `metaTypeId` int(3) unsigned NOT NULL,
  `content` varchar(255) NOT NULL default '',
  `tcStart` int(10) UNSIGNED NOT NULL,
  `tcEnd` int(10) UNSIGNED NOT NULL,
  PRIMARY KEY  (`id`),
  FOREIGN KEY (`metaTypeId`) REFERENCES `metaType` (`metaTypeId`),
  FOREIGN KEY (`movieId`) REFERENCES `movie` (`movieId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Figure 3.14: Database structure

We used the same approach as with subtitles to extract chapter metadata. Chapter information and time codes were extracted from a chapter text file and added to the "metadata" table. A sample from of the "From Russia with Love" chapter text file has the following structure:

```
CHAPTER01=00:00:00.000
CHAPTER01NAME=Rehersal For Murder
CHAPTER02=00:03:08.800
CHAPTER02NAME=Logos
```

We needed the timecode and chapter name from each event in the file. Figure 3.15 illustrates the function that extracts this information and inserts it into the database table.



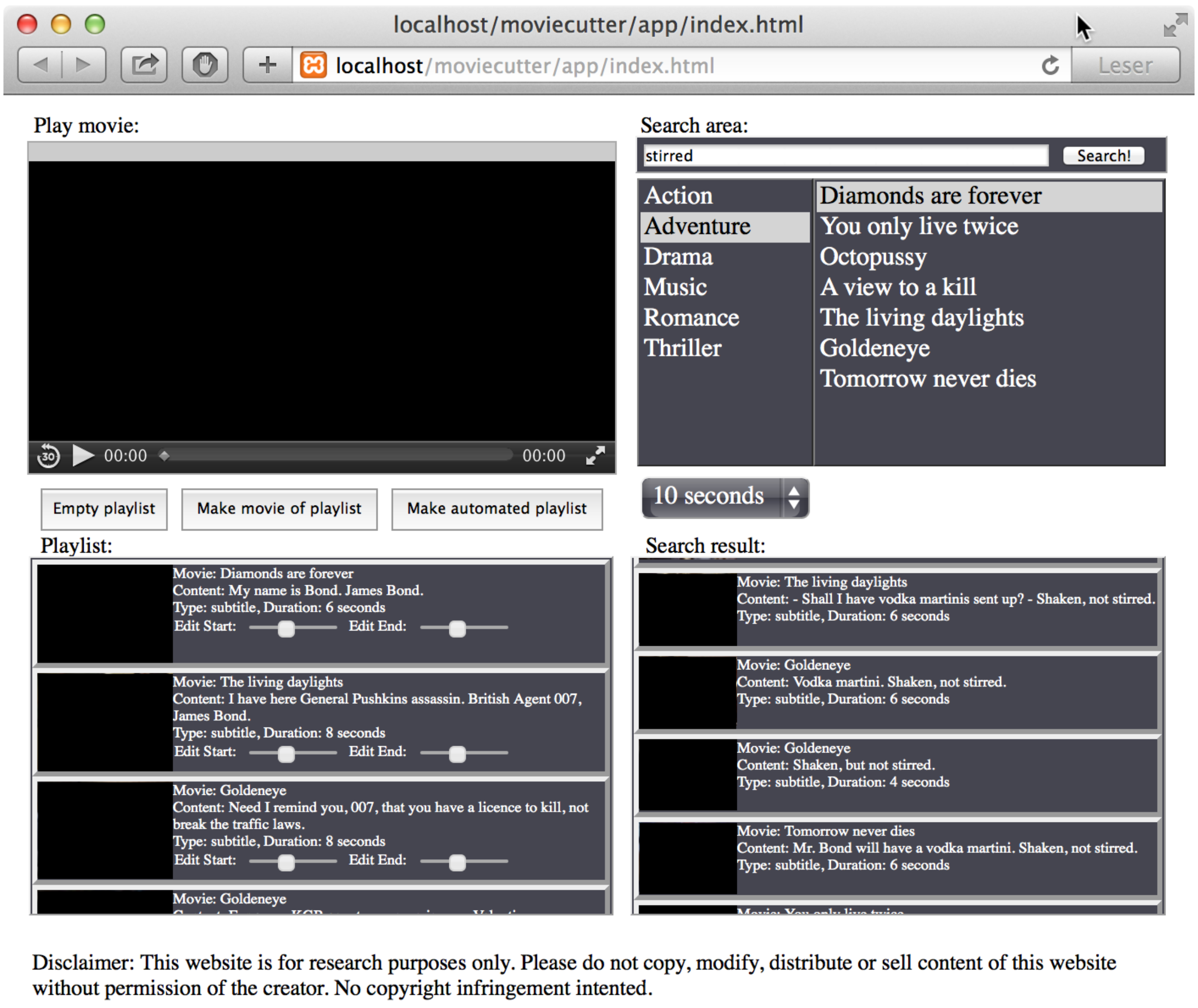
```php
function makeDbTableOutOfChapters( $object ){
    $filePath = $object["path"];
    $movieId = $object["movie"];
    $fileTypeId = $object["fileType"];
    $endTc = convertTCtoSeconds ( $object["endTc"] );
    $totalLines = intval(exec("wc -l '$filePath'"));
    $handle = fopen($filePath, "r");

    if ($handle){
        $startLine = fgets($handle);
        $tc = explode("=", $startLine);
        $start = convertTCtoSeconds ($tc[1]);
        $end = 0;
        $i = 1;

        while (($line = fgets($handle)) !== false){
            $i++;
            $chapterCollection = explode("=", $line);
            $content = $chapterCollection[1];
            $content = str_replace("'", "" ,$content);
            $content = str_replace("\"", "" ,$content);
            $content = str_replace("<i>", "" ,$content);

            if ( $i === $totalLines){
                $end = $endTc;
            }
            else{
                $nextLine = fgets($handle);
                $i++;
                $endTimecode = explode("=", $nextLine);
                $end = convertTCtoSeconds ($endTimecode[1]);
            }

            sqlQuery(   'INSERT INTO metadata ( tcStart, tcEnd, content, movieId, metaTypeId )
                        VALUES("'.$start.'","'.$end.'","'.
                        $content.'","'.$movieId.'","'.$fileTypeId.'")');

            $start = $end;
    }}
    else{
        echo("Error reading file");
    }
    echo '{ "choice":"makeDbTableOutOfChapters" }';
}
```

Figure 3.15: Function for extracting chapter metadata

The function is similar to extracting subtitle events from a .srt file. Note that each chapter only contains the start timecode. Hence, we need to use the next chapter's start timecode as the end timecode of the previous chapter. For the last chapter, we input the entire duration of the movie as end timecode. Special characters like hyphens are removed before inserted into the database. Such characters can cause SQL injection. Removing them has a minor effect on text based searches. Support for such characters might be implemented on a later stage. Since both subtitles and chapter metadata are stored in the same table, we can do text based queries on both types of events.

3.8 User interface

Adding more metadata events, requires a user interface that indicates what kind of data is presented. For example, the current user interface illustrated in figure 3.13 only shows the index value of each search result. Figure 3.16 illustrates an improved user interface.



Movie: Diamonds are forever
Content: My name is Bond. James Bond.
Type: subtitle, Duration: 6 seconds
Edit Start: Edit End:

Movie: The living daylights
Content: I have here General Pushkins assassin. British Agent 007, James Bond.
Type: subtitle, Duration: 8 seconds
Edit Start: Edit End:

Movie: Goldeneye
Content: Need I remind you, 007, that you have a licence to kill, not break the traffic laws.
Type: subtitle, Duration: 8 seconds
Edit Start: Edit End:

Movie: Goldeneye

Movie: The living daylights
Content: - Shall I have vodka martinis sent up? - Shaken, not stirred.
Type: subtitle, Duration: 6 seconds

Movie: Goldeneye
Content: Vodka martini. Shaken, not stirred.
Type: subtitle, Duration: 6 seconds

Movie: Goldeneye
Content: Shaken, but not stirred.
Type: subtitle, Duration: 4 seconds

Movie: Tomorrow never dies
Content: Mr. Bond will have a vodka martini. Shaken, not stirred.
Type: subtitle, Duration: 6 seconds

Disclaimer: This website is for research purposes only. Please do not copy, modify, distribute or sell content of this website without permission of the creator. No copyright infringement intented.

Figure 3.16: User interface square format

This user interface has a square format consisting of four different containers.

- Top left container: The client video player and make playlist button
- Bottom left container: The playlist
- Top right container: Different search options and automated playlist functionality
- Bottom left container: Search result list

3.8.1 Search results

The search bar has the same functionality as the previous version of the system. Pressing the "Search" button will return all types of metadata stored in the "metadata" table. This includes both chapter and subtitle events. Alternatively, a end user can search for chapters by clicking on the category list. A list of genre specific movies appears. By clicking on a movie, all chapters within this movies appears in the search result area.

We have included more information about each item in the search result list:

- A thumbnail image representing the event
- The title of the movie the event belongs to
- A description of a chapter event, or a subtitle event dialogue
- Type of metadata such as subtitle or chapter events
- The duration of the event

This information is represented inside a container. Each container is one instance in the search result list.

3.8.2 Tumbnails

To create thumbnails we wrote a little bash script using FFmpeg.

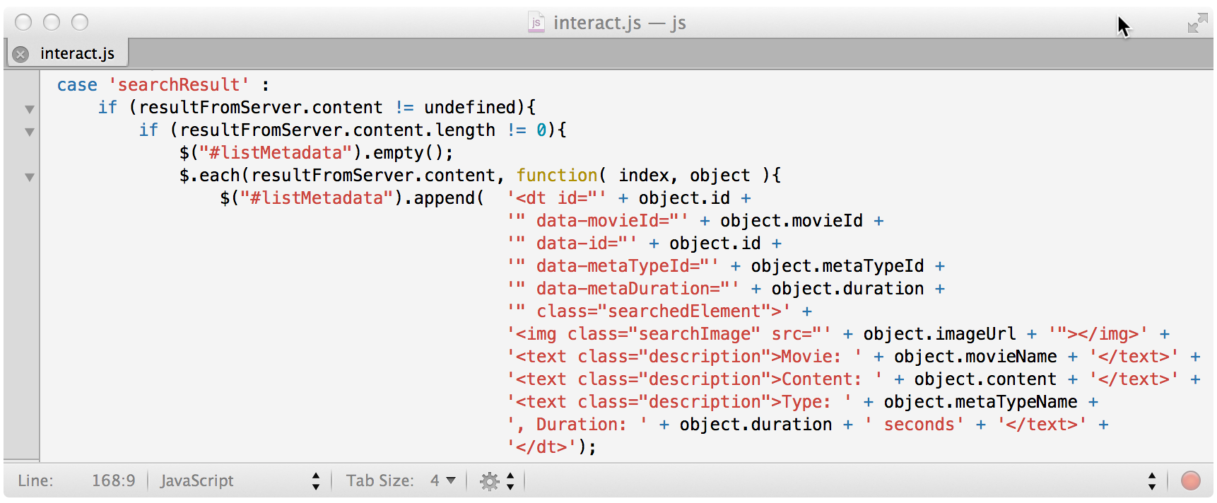
```
#!/bin/bash
i=0
for ts in sourceMovie/*.ts; do
./ffmpeg -i $ts -vf select="eq(pict_type\PICT_TYPE_I),
scale=90:-1,crop=70:ih:10:10" -vsync 0 -r 1 -vframes 1 -f image2
thumbnails/$i.jpg; i=$((i+1))
done
```

This script is run after a movie resource is segmented into small video fragments. One I-frame image is then extracted from each segmented fragment. They are stored in a separate folder with the same base filename. While being extracted from a .ts file, the I-frame image is first scaled down to a suitable size and then cropped. A movie image usually has a 16:9 [119] or wider format, which is too wide for our search result container. With a 4:3 format [121] without altering the aspect ratio, the thumbnail fits nicely within our container. Most movies follow traditional composition conventions like the Golden ratio [141]. This ensures that not to much image information is lost when cropping each side of the thumbnail.

Our system's search functionality is illustrated in figure 3.8. This function returns necessary metadata to be included in the search result container. The container's thumbnail is localized on the server using an URL.

3.8.3 Dynamic HTML

The content in the search result container list is built dynamically in a javascript function. Figure 3.17 illustrates how the different parameters from the JSON object is included as HTML tags and attributes.



```
case 'searchResult' :
    if (resultFromServer.content != undefined){
        if (resultFromServer.content.length != 0){
            $("#listMetadata").empty();
            $.each(resultFromServer.content, function( index, object ){
                $("#listMetadata").append(  '<dt id="' + object.id +
                                            '" data-movieId="' + object.movieId +
                                            '" data-id="' + object.id +
                                            '" data-metaTypeId="' + object.metaTypeId +
                                            '" data-metaDuration="' + object.duration +
                                            '" class="searchedElement">' +
                                            '<img class="searchImage" src="' + object.imageUrl + '"></img>' +
                                            '<text class="description">Movie: ' + object.movieName + '</text>' +
                                            '<text class="description">Content: ' + object.content + '</text>' +
                                            '<text class="description">Type: ' + object.metaTypeName +
                                            ', Duration: ' + object.duration + ' seconds' + '</text>' +
                                            '</dt>');
```

Figure 3.17: Building dynamic HTML in javascript

A class attribute references some CSS rules to format the HTML tags within the container correctly. Note the "data" attributes. These values are used when the client queries the server for a new manifest.

3.8.4 End user playlist

The playlist container list is similar to the search result container list. It includes the same thumbnail and metadata details. An end user can drag and drop items from the search result container to be copied into the playlist container. It is also possible to reorder this list using drag and drop. Optionally, dragging and dropping an item from the playlist outside the playlist border, will remove the item from the playlist container. The button "Empty playlist" will remove all items from the playlist container.

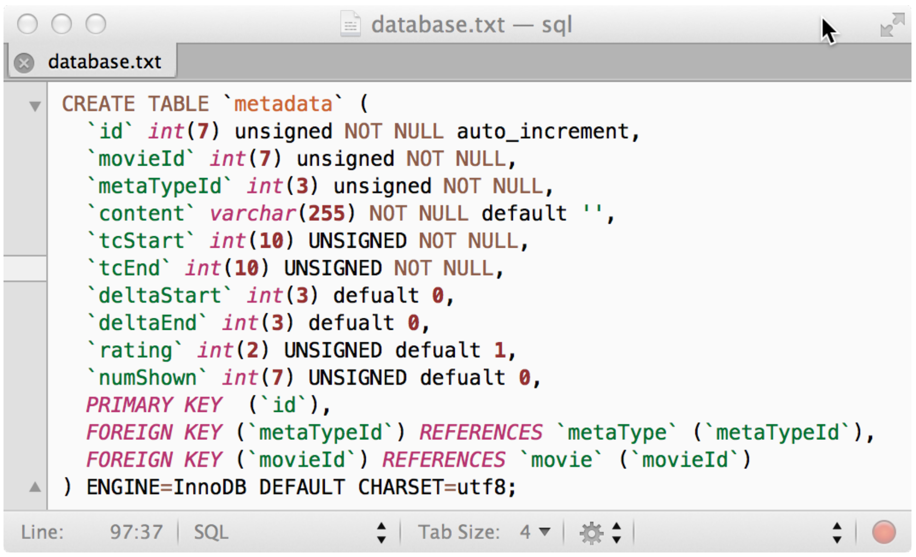
It is also possible to move search results items automatically to the playlist container. If an end user chooses an appropriate duration in the drop-down list, and presses the "Make automated playlist" button, a javascript function will calculate a total duration based upon items in the the search result container. Beginning with the first item, this function will continue to add search result items to the playlist until the selected duration value has been reached.

3.8.5 Video handles

In table 3.1, we illustrated that it can be difficult to perfectly map subtitle events to video segment ranges. Even with a perfect match, an end user might want access to video segments before or after the system's default subtitle event. For example, an end user searches for some specific dialogue, but wants to include what happened before this dialogue in the playlist container. Hence, we have included video handles [19] in our prototype.

We had to decide whether to store these end user changes in video duration, or restore default values for each end user. There are pros and cons to each approach. Storing changes in video duration might optimize search results. Using a video segment duration of two seconds will sometimes create an offset between the metadata event and the video segment range. If an end user adjusts this offset perfectly, then another end user will experience an optimized search result. On the other hand, if an end user creates a larger offset because of creative needs, then another end user might experience the system as inaccurate.

We decided upon something in between. During testing of our system, we found that metadata events mapped almost perfectly with the video segments. Only minor changes were necessary. We did not want to change the default start and end values in the metadata table. Instead, we added two new columns to the metadata table with a default value of 0. Figure 3.18 illustrates the new structure of the metadata table.



```
CREATE TABLE `metadata` (
  `id` int(7) unsigned NOT NULL auto_increment,
  `movieId` int(7) unsigned NOT NULL,
  `metaTypeId` int(3) unsigned NOT NULL,
  `content` varchar(255) NOT NULL default '',
  `tcStart` int(10) UNSIGNED NOT NULL,
  `tcEnd` int(10) UNSIGNED NOT NULL,
  `deltaStart` int(3) defualt 0,
  `deltaEnd` int(3) defualt 0,
  `rating` int(2) UNSIGNED default 1,
  `numShown` int(7) UNSIGNED defualt 0,
  PRIMARY KEY  (`id`),
  FOREIGN KEY (`metaTypeId`) REFERENCES `metaType` (`metaTypeId`),
  FOREIGN KEY (`movieId`) REFERENCES `movie` (`movieId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Figure 3.18: Metadata table

End user adjustments rewrites these columns. When calculating a video segment range in a manifest, we add or subtract these values together with the default values. An end user offset will still have an impact on how other end users will experience the system, but by setting a rule on how large the video handles can be, this will have minor implications.

Notice that we have also included two more columns in the metadata table for future development. The "rating" column is supposed to allow end users to rank each search result. The "numShown" column is supposed to count each time a metadata event is played back from a manifest file. These columns are intended for optimizing the system's search engine, so that the most popular dialogue and video scenes are prioritized when creating a search result list. This functionality might be implemented on a later stage.

3.8.6 Video playback

It is possible to preview a metadata event. If a list item is dragged from either the playlist or the search result container list and dropped in the player, it will playback this video segment range. An end user can then preview a scene before including it in a manifest file. If an end user presses the "Make movie of playlist" button, the system creates a manifest of the entire playlist.

3.8.7 Highlighting playlist and search result events

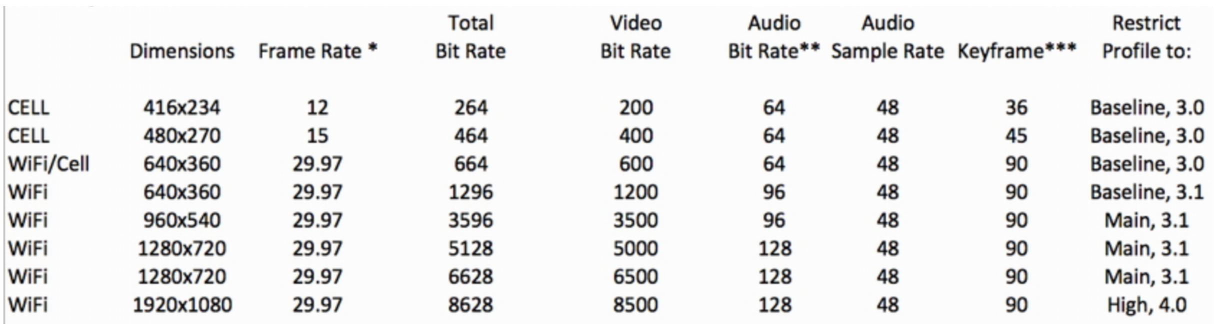
During playback, a javascript function changes background color on items being played back in the manifest. Metadata events from the search result container list changes color to black. This might be helpful when selecting items for a new manifest. In the playlist container only the item being currently played changes color. This enables an end user to read metadata about the metadata event being played back.

3.9 Adaptive player

The system has been tested with only one video quality for each movie. They have been chopped into small segments of two seconds duration. The same procedure is used for making a movie resource ready for adaptive streaming. The next step in our system is to create two different video qualities of each movie resource. This enables us to switch between bit rates as CPU and network conditions changes. This requires a variant manifest that lets the client player know the necessary information to change between the two video qualities.

3.9.1 Supported bit rates

Apple's best practices for HTTP live streaming to iPhone and iPad devices recommend certain encodings [17]. Figure 3.19 illustrates the different 16:9 aspect ratios recommended.



	Dimensions	Frame Rate *	Total Bit Rate	Video Bit Rate	Audio Bit Rate**	Audio Sample Rate	Keyframe***	Restrict Profile to:
CELL	416x234	12	264	200	64	48	36	Baseline, 3.0
CELL	480x270	15	464	400	64	48	45	Baseline, 3.0
WiFi/Cell	640x360	29.97	664	600	64	48	90	Baseline, 3.0
WiFi	640x360	29.97	1296	1200	96	48	90	Baseline, 3.1
WiFi	960x540	29.97	3596	3500	96	48	90	Main, 3.1
WiFi	1280x720	29.97	5128	5000	128	48	90	Main, 3.1
WiFi	1280x720	29.97	6628	6500	128	48	90	Main, 3.1
WiFi	1920x1080	29.97	8628	8500	128	48	90	High, 4.0

Figure 3.19: Recommended 16:9 encodings

We should ideally encode each movie in all dimensions and with different bit rates. This will provide as good as it gets experience for all receiving devices. However, not all encodings are suitable for our system. For example, two of the encodings for "CELL" devices recommend half the frame rate of a normal movie. This will have major impact on the movie's duration, and complicates the process of mapping metadata events to the correct video segment ranges. Most of our test data is Standard Definition (SD) quality [186]. The larger dimensions in this recommendation imply High Definition (HD) quality [146]. We also need to consider the system's client player CSS values. The current player has a width of 416 pixels and a height of 234 pixels. It is possible to preview the video in full screen, but then the player will upscale automatically. However, upscaled SD video content does not look as good as if the source video is HD.

We need two dimensions to test the adaptive streaming capabilities of our system. One dimension needs to be lower than the CSS values of 416x234, and the other dimension needs to be above these values. The main point being able to see the difference in quality as the variant manifest adapts to changes in network traffic or CPU usage. We have chosen an aspect ratio of 156x88 as the low dimension, and an aspect ratio of 960x540 as the high dimension. The latter dimension is in the middle of Apple's recommended dimensions. The following example segments a movie resource in with a high aspect ratio:

```
./ffmpeg -i ./fromRussiaWithLove.mp4
-bsf:v h264_mp4toannexb
-flags
-global_header
-map 0:v
-c:v libx264
-b:v 1800k
-preset medium
-profile:v main
-level 3.1
-s 960x540
```

```
-sc_threshold 0
-keyint_min 25
-g 25
-map 0:a
-c:a libmp3lame
-b:a 64k
-ar 44100
-f segment
-segment_time 2
-segment_list fromrussiawithlove.m3u8
-segment_format mpegts high/%05d.ts
```

The "-s" parameter defines the aspect ratio, and the "-b:v" parameter defines the bit rate.

3.9.2 Creating a variant manifest using Apple's tools

We need a variant manifest to change quality during playback. We tried using Apple's "variantplaylistcreator" command line tool. Version 4 of the manifest protocol requires three different files for each quality of the movie:

- Manifest file
- I-frame manifest file
- .PLIST file

To create these files, we use the "mediafilesegmenter" tool with these parameters:

```
sudo ./mediafilesegmenter -t 2 -i high.m3u8
-z high-iframe.m3u8 -I -f hq high.ts

sudo ./mediafilesegmenter -t 2 -i low.m3u8
-z low-iframe.m3u8 -I -f lq low.ts
```

The "I" parameter tells the "mediafilesegmenter" to create a .PLIST file. To create a variant manifest we used the "variantplaylistcreator" tool:

```
sudo ./variantplaylistcreator -o variantplaylist.m3u8
hq/high.m3u8 high.plist -i hq/high-iframe.m3u8
lq/low.m3u8 low.plist -i lq/low-iframe.m3u8
```

The .PLIST file has an XML format. One of our test files looked like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
<key>avgBitRate</key>
<real>642537.95918367349</real>
<key>iframe_MaxBitRate</key>
<real>75200</real>
<key>iframe_avgBitRate</key>
<real>45542.040816326531</real>
<key>maxBitRate</key>
<real>1443088</real>
<key>videoCodecInfo</key>
```

```
<string>avc1.4d401e</string>
<key>videoDimensions</key>
<dict>
<key>Height</key>
<real>600</real>
<key>Width</key>
<real>1391.304347826087</real>
</dict>
</dict>
</plist>
```

We assume the "variantplaylistcreator" tool uses this meta data when generating a variant manifest. For example, the inner "dict" tag contains aspect ratio information. Here is a sample from the created variant manifest file:

```
#EXTM3U
EXT-X-STREAM-INF: BANDWIDTH=1370934,
CODECS="avc1.44d401e",RESOLUTION="1391x600"
URI=lq/low.m3u8
EXT-X-I-FRAME-STREAM-INF:BANDWIDTH=71440,
CODES="avc1.4d401e",RESOLUTION=1391x600,
URI=lq/low-iframe.m3u8
```

Notice the EXT-X-STREAM-INF command tag in the variant manifest. Each tag represent a manifest file that the client player can switch between. Attributes such as BANDWIDTH, CODECS and RESOLUTION helps the client player choose the appropriate manifest.

3.9.3 Variant manifest rules

Our system uses dynamically created manifests. Using both a regular and the I-frame manifest file adds complexity to the system. We checked Apple's technical notes for the minimum requirements to create a variant manifest.

- "Every EXT-X-STREAM-INF command tag must include the BANDWIDTH attribute. The value is a decimal-integer of bits per second. It must be an upper bound of the overall bit rate of each media file, calculated to include container overhead, that appears or will appear in the playlist" [18].
- "For identifying different encodings of the same presentation the technical notes recommends using a PROGRAM-ID attribute. Multiple EXT-X-STREAM-INF tags can have the same PROGRAM-ID attribute value" [18].

For example, a variant playlist can look like this:

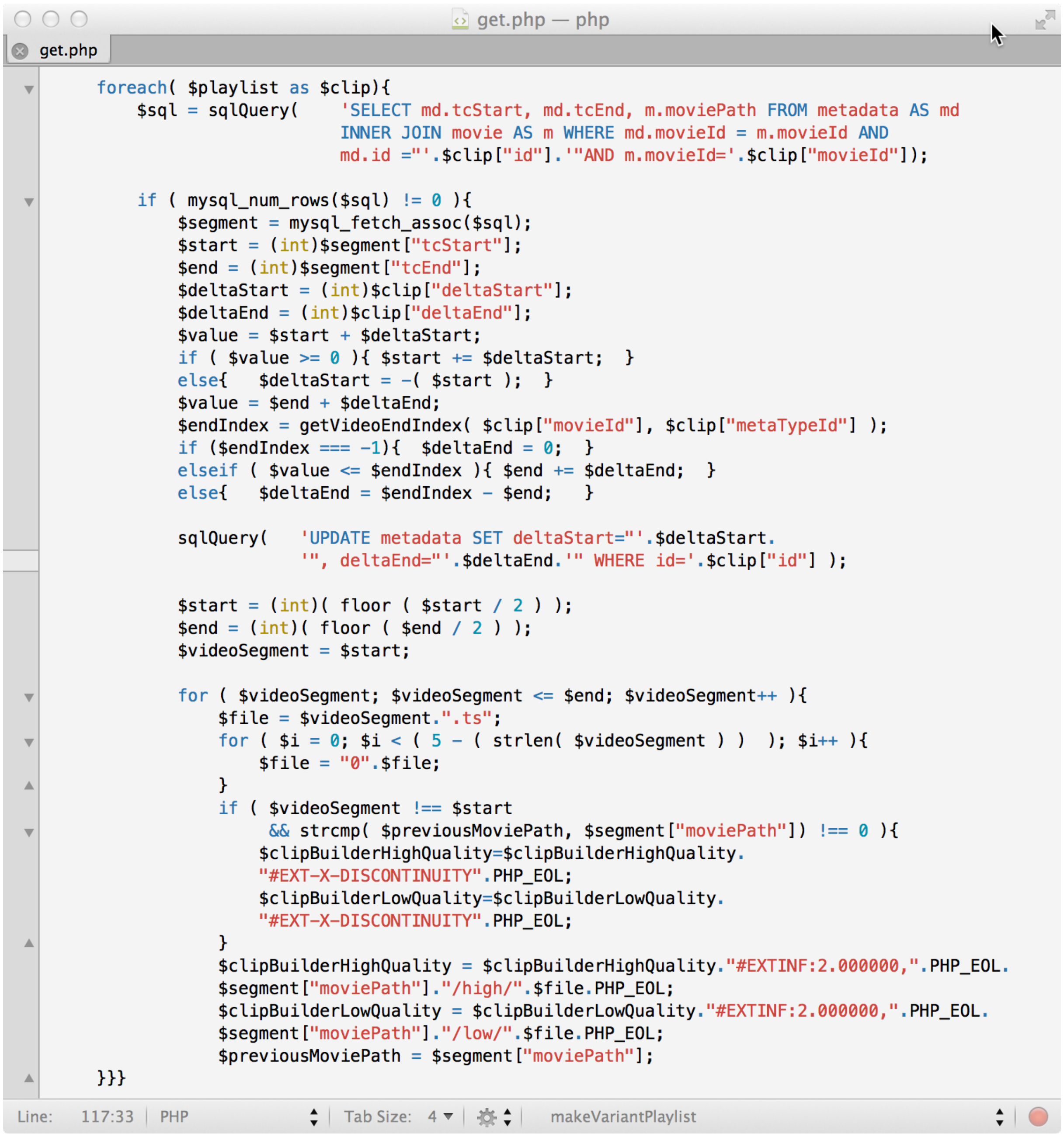
```
#EXTM3U
EXT-X-STREAM-INF: PROGRAM-ID=1,BANDWIDTH=264000,
RESOLUTION="156x88"
29d76bbfa9695b56707dadc81ba1a7d35d04aefc.m3u8
EXT-X-I-FRAME-STREAM-INF:PROGRAMID=1,
BANDWIDTH=1864000,RESOLUTION=950x540,
53ad1adb6cf7243a4a8dca37d346c6964e95d51c.m3u8
```

The hash generated manifest file names represent manifest files of different bit rates.

3.9.4 Creating a dynamic variant manifest

We will create a variant manifest file using these rules. All manifest files will also be auto generated. This enables multiple end users to generate manifests at the same time. This auto generating process uses a Secure Hash algorithm(SHA) [184] on a millisecond timer, ensuring that the temporary files get unique filenames. For each new manifest created, we create three manifest files. One manifest file for each bit rate quality, and one variant manifest file to allow the client player to switch between the bit rate qualities. We check the manifest timestamp when a new manifest is created to clean the system of outdated manifest files.

We have rewritten the "make manifest" function 3.12 and included a variant manifest in the function. Figure 3.20 illustrates the video segment range loop in this new "make variant manifest" function.



```php
foreach( $playlist as $clip){
    $sql = sqlQuery(    'SELECT md.tcStart, md.tcEnd, m.moviePath FROM metadata AS md
                        INNER JOIN movie AS m WHERE md.movieId = m.movieId AND
                        md.id ="'.$clip["id"].'"AND m.movieId='.$clip["movieId"]);

    if ( mysql_num_rows($sql) != 0 ){
        $segment = mysql_fetch_assoc($sql);
        $start = (int)$segment["tcStart"];
        $end = (int)$segment["tcEnd"];
        $deltaStart = (int)$clip["deltaStart"];
        $deltaEnd = (int)$clip["deltaEnd"];
        $value = $start + $deltaStart;
        if ( $value >= 0 ){ $start += $deltaStart;  }
        else{   $deltaStart = -( $start );  }
        $value = $end + $deltaEnd;
        $endIndex = getVideoEndIndex( $clip["movieId"], $clip["metaTypeId"] );
        if ($endIndex === -1){  $deltaEnd = 0;  }
        elseif ( $value <= $endIndex ){ $end += $deltaEnd;  }
        else{   $deltaEnd = $endIndex - $end;   }

        sqlQuery(   'UPDATE metadata SET deltaStart="'.$deltaStart.
                    '", deltaEnd="'.$deltaEnd.'" WHERE id='.$clip["id"] );

        $start = (int)( floor ( $start / 2 ) );
        $end = (int)( floor ( $end / 2 ) );
        $videoSegment = $start;

        for ( $videoSegment; $videoSegment <= $end; $videoSegment++ ){
            $file = $videoSegment.".ts";
            for ( $i = 0; $i < ( 5 - ( strlen( $videoSegment ) )  ); $i++ ){
                $file = "0".$file;
            }
            if ( $videoSegment !== $start
                 && strcmp( $previousMoviePath, $segment["moviePath"]) !== 0 ){
                $clipBuilderHighQuality=$clipBuilderHighQuality.
                "#EXT-X-DISCONTINUITY".PHP_EOL;
                $clipBuilderLowQuality=$clipBuilderLowQuality.
                "#EXT-X-DISCONTINUITY".PHP_EOL;
            }
            $clipBuilderHighQuality = $clipBuilderHighQuality."#EXTINF:2.000000,".PHP_EOL.
            $segment["moviePath"]."/high/".$file.PHP_EOL;
            $clipBuilderLowQuality = $clipBuilderLowQuality."#EXTINF:2.000000,".PHP_EOL.
            $segment["moviePath"]."/low/".$file.PHP_EOL;
            $previousMoviePath = $segment["moviePath"];
}}}
```

```
Line: 117:33 | PHP | Tab Size: 4 | makeVariantPlaylist
```

Figure 3.20: Function for creating a variant manifest playlist

Notice the "moviePath" variable at the end of the illustration. This is the only difference between the bit rate manifests. They have file paths to different folders, and contain URL's to video segments of different quality.

We noticed that the client player usually begins with the lowest quality while testing the variant manifest. After a few seconds, it changes to the higher quality. It is easy to spot the difference since the low quality seems out of focus. This is how an adaptive player should work. It starts with the low bit rate, so an end user can watch the video almost instantly. As the client buffers data, the player can switch to the high bit rate while maintaining a smooth playback experience.

If the client player is told to playback a manifest of less than five seconds it behaves differently. It begins with the high bit rate and then changes to the low bit rate. This looks strange and can happen during playback of a subtitle event. We have earlier pointed out that Apple recommend segments of 10 seconds duration. However, playing a single subtitle event is usually for preview purposes. A subtitle event will usually be combined with other events in the playlist before playback in the player. Hence, with a total duration longer than 10 seconds. We do not know if the HLS player differentiate between a segment of 10 seconds or multiple segments with a total duration of 10 seconds or more. However, playback of segments in sequence does not seem to show this strange behavior.

3.10 Integration

Our system uses existing technologies such as Apache web-server [10], MYSQL relational database [84], FFmpeg [33] encoding tools and Apple's HLS [13, 14, 31, 95]. These technologies are utilized by some of the following system components.

- Search component
- Playlist compont
- Metadata extraction component
- Video segment component
- Database component

We will now describe how these components interact. These interactions are illustrated in the system architecture overview figure 3.1.

3.10.1 Frontend

The combined functionality of the search and playlist component streams a new video from selected metadata events. The search container contains a text based search engine and a click based category list. Text based searches returns both dialogue and chapters events in the search result container. Alternatively, clicking on a category returns a list of available movies within that genre. Clicking on one of the movies, returns chapter events in the search result container. Both alternatives have similar search functions on the server side. They both return the same JSON [67] formatted metadata.

An end user can drag and drop events directly into the player or via a playlist. We have also created an automated alternative to drag and drop. Either way, the server uses the create manifest function. Each requested metadata event are iterated and included in the necessary manifest files. The server returns the URL of the variant manifest, so the client player can start playback of the playlist.

3.10.2 Backend

The metadata extraction component and the video segment component adds new media to the system. Our metadata resources is not embedded in the video resource. Hence, we need two different processes to include the necessary media.

Metadata events based on subtitles and chapters are both text file resources, but they are formatted in different ways. Hence, we need different functions to parse them and extract correctly formatted data for our metadata database table.

We use FFmpeg to segment a video resource in different bit rates and to extract thumbnails. After the segmentation and extraction process, we move the folders containing the media into our project folder. The folder structure is as following inside the project folder:

```
manifestAndMovies
|-->Movie name
      |-->High (video segments with high bit rate)
      |-->Low (video segments with low bit rate)
      |-->Thumbnails (I-frame still images)
```

Notice that the auto generated manifest files also is stored in the manifestAndMovies folder.

3.10.3 Database

The metadata table is an additional component that the other components use. Backend components adds new data into this table. Frontend components queries this table for available metadata events. The system also uses the database tables to locate video segments and thumbnails necessary for presentation.

3.11 Summary

We have extracted timecodes and textual descriptions from the metadata resources we found in the online databases. This data has been inserted into our own database. Our system maps metadata events to a specific range of video segments.

The user interface went through several iterations. Our first goal was to play a segmented movie. We used the HTML5 video tag to playback a manifest of an entire movie. The next step was to rearrange the scenes within amovie. We created a drag and drop playlist to reorder events within a movie. By selecting a few video segment ranges and rearranging them, we could create a manifest that the player could use to playback a montage of our playlist. Being able to create such a dynamic manifest was a major breakthrough. Knowing how to build the manifest, we could now play back movies of different sources and qualities. We made the sytstem adaptive by including a variant manifest.

We implemented different search options functionality. It is possible to do text based searches on metadata events and present specific scenes within a movie. Alternatively, chapter events can be found by selecting movie categories and movie names. By searching metadata events we can easily locate a specific scene within a movie. The search result can be played back almost instantly. In the next chapter, we will evaluate our system.

Chapter 4

Evaluation and Discussion

We implemented our system in the previous chapter. Through trial and error, we figured out how to create video segments of a fixed duration and how to create manifest files dynamically. We were able to develop a working prototype using the HLS protocol and adaptive streaming. In the following sections, we will evaluate some of the challenges implementing the system and present a user study. We will recommend future work to improve this system on the basis of these evaluations.

4.1 Mapping metadata resources to video segments

We needed to find metadata resources that mapped accurately with the video segments. For example, the subtitle metadata resource did not match the video segments in the "Goldfinger" movie. We used a 25 frames per second subtitle resource on an approximately 30 frames per second encoded movie. Meaning, some of the subtitle time codes referenced events later than the movie's total duration. A 30 frames per second subtitle resource did not match the movie either. The offset was smaller than with the 25 frames per second resource, but not a perfect match. Our version of the movie probably had a different duration than the metadata resources we were trying to map it with. For example, a director's cut edition or a special version edited for a specific market or country.

There are tools for extracting subtitles from commercial products such as DVD's. Using such a tool will probably give a perfect match between metadata resources and video segments. We only needed a few test movies for our system, and using an online database was less time-consuming. In such a database we found the movie's frames per seconds, but not it's duration. Therefore, there were some trial and error before we found a perfect match for all the movies we wanted to use.

Matching chapter metadata resources with video segments were easier. The ChaperDb online database described in subsection 2.16.8, included both frames per second and duration information about movies. These two variables combined is a good indicator of how good a metadata resource matches the video segments. We used FFmpeg commands for retrieving comparable frames per second and duration information from movie resources. Still, there were some divergences. A forum comment about the ChapterDb database pointed out some issues with public databases.

"Given the issues than can occur during importing and naming of chapters (timecode issues with FPS etc), or just general new-user tomfoolery as they learn the program; I think automatically uploading based on a file save is more trouble than it is worth. It would cost a lot of time and effort to scrub the site, for thousands of entries, especially if your user-base starts growing. Differences in naming standards and all that also create many duplicate entries, etc. I think it would be better to lock DB access, and take on approved users. Establish clear quality-control issues, etc." [89].

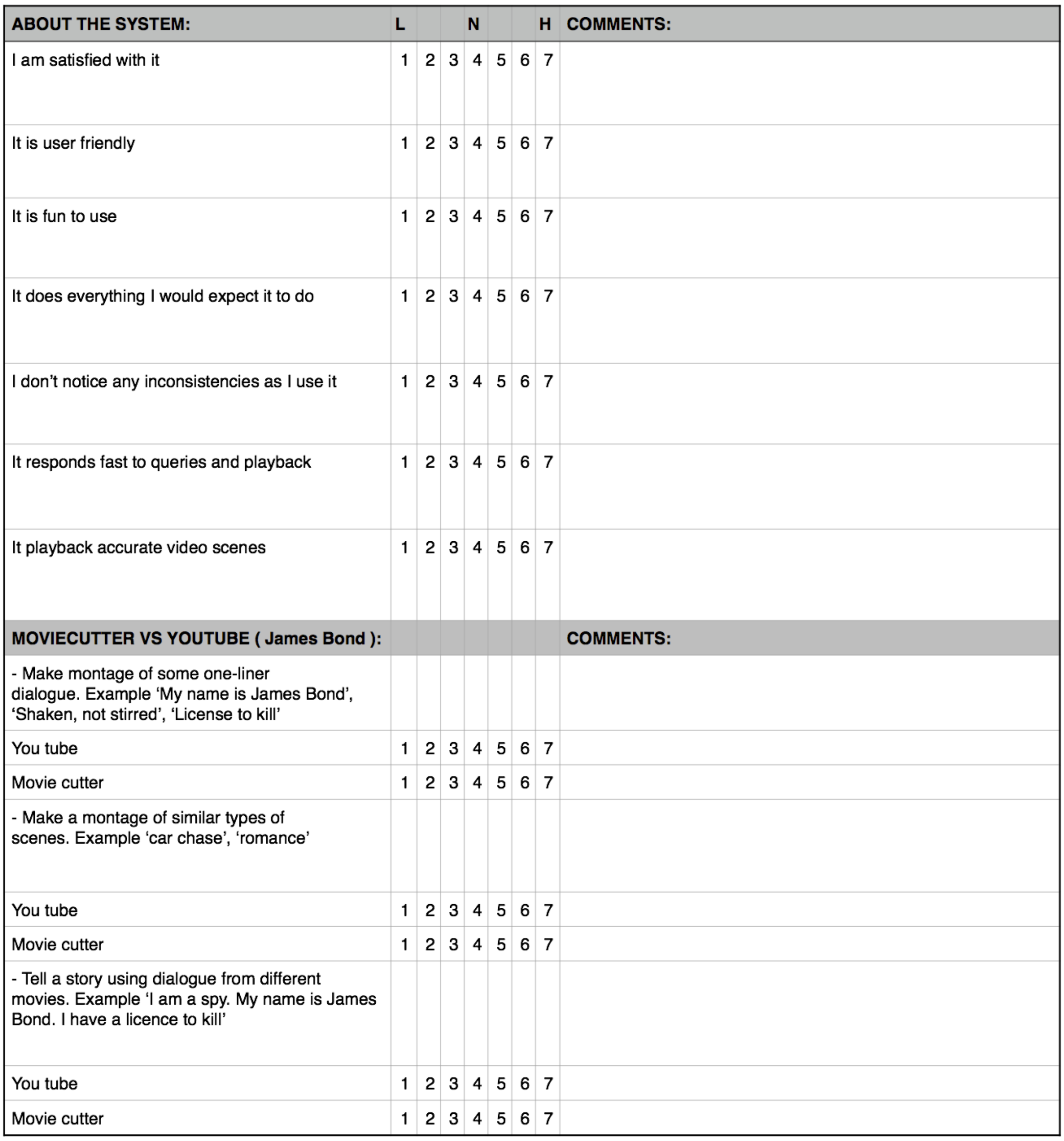
Such issues can reduced the quality of a metadata resource. If everybody using the ChapterDb software uploads metadata to the online database, there will be duplicate entries. However, it is possible to configure this program to not upload the created metadata. If we were unable to find a perfect match in the online database, we had the option of using this software to create our own chapter metadata from a video resource.

In general, we have learned that using metadata resources based on online databases can be of good or bad quality. To improve quality, we have the option of manually logging each metadata resource.

This gives us the flexibility to log any event we find interesting within a movie. For example, users of the "moviebodycounts" website described in subsection 2.16.6, will probably want to watch montages of movie counts. However, this is a time-consuming process and was not a practical approach for our prototype.

4.2 User study

We have performed a user study using a 7-point balanced keying Likert scale (1-useless, 2-strongly dislike, 3-dislike, 4-neutral, 5-like, 6-strongly like and 7-wow) to test the functionality of our system. We asked people on the fly if they were interested in doing a user study interview using a laptop. Most of the interviews happened in the halls of the University of Oslo. 25 assessors, both technical and non-technical persons, tested the system. Our survey is illustrated in figure 4.1.



ABOUT THE SYSTEM:	**L**			**N**			**H**	**COMMENTS:**
I am satisfied with it	1	2	3	4	5	6	7	
It is user friendly	1	2	3	4	5	6	7	
It is fun to use	1	2	3	4	5	6	7	
It does everything I would expect it to do	1	2	3	4	5	6	7	
I don't notice any inconsistencies as I use it	1	2	3	4	5	6	7	
It responds fast to queries and playback	1	2	3	4	5	6	7	
It playback accurate video scenes	1	2	3	4	5	6	7	
MOVIECUTTER VS YOUTUBE (James Bond):								**COMMENTS:**
- Make montage of some one-liner dialogue. Example 'My name is James Bond', 'Shaken, not stirred', 'License to kill'								
You tube	1	2	3	4	5	6	7	
Movie cutter	1	2	3	4	5	6	7	
- Make a montage of similar types of scenes. Example 'car chase', 'romance'								
You tube	1	2	3	4	5	6	7	
Movie cutter	1	2	3	4	5	6	7	
- Tell a story using dialogue from different movies. Example 'I am a spy. My name is James Bond. I have a licence to kill'								
You tube	1	2	3	4	5	6	7	
Movie cutter	1	2	3	4	5	6	7	

Figure 4.1: User survey

The results of this study is presented in tables using averages and standard deviations. A full list of feedback from the participants can be found in Appendix B.

4.2.1 General questions about the system

People were told to search for dialogue and scene events within movies. They were asked to create playlist montages of these events and evaluate the system. The first part of the survey listed a series of statements about our system. The results of these questions are presented in table 4.1.

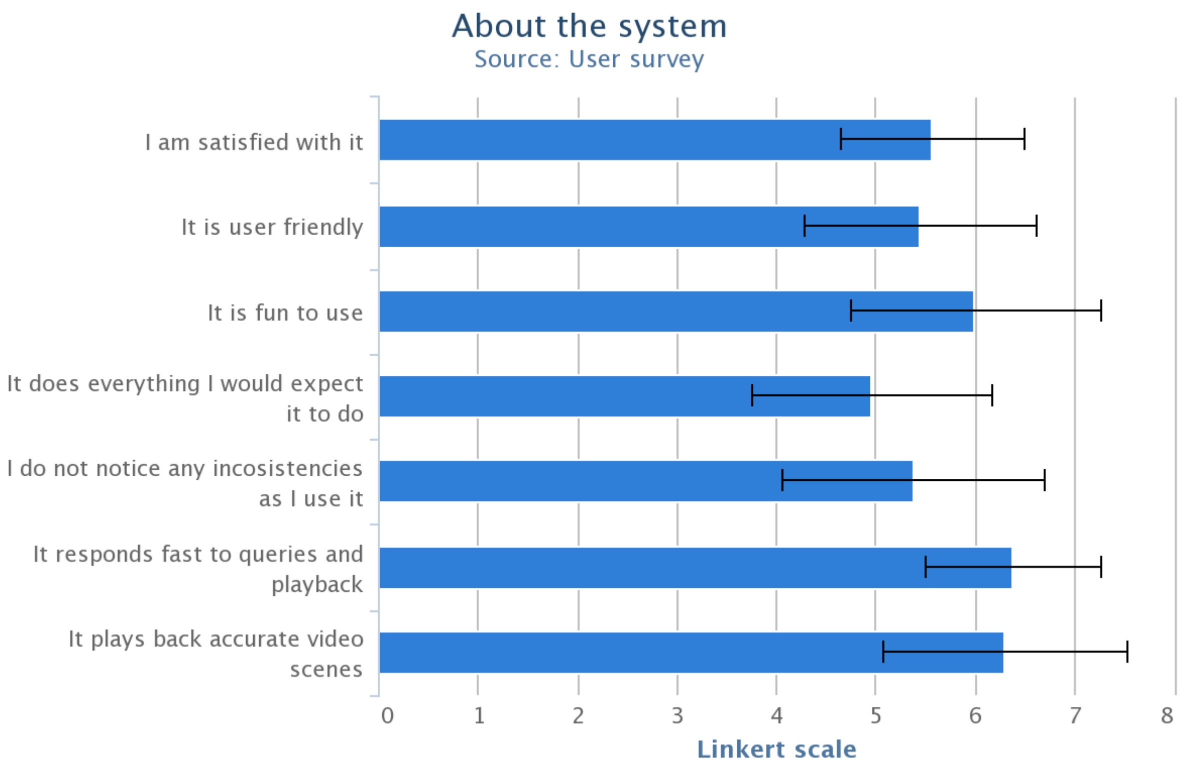
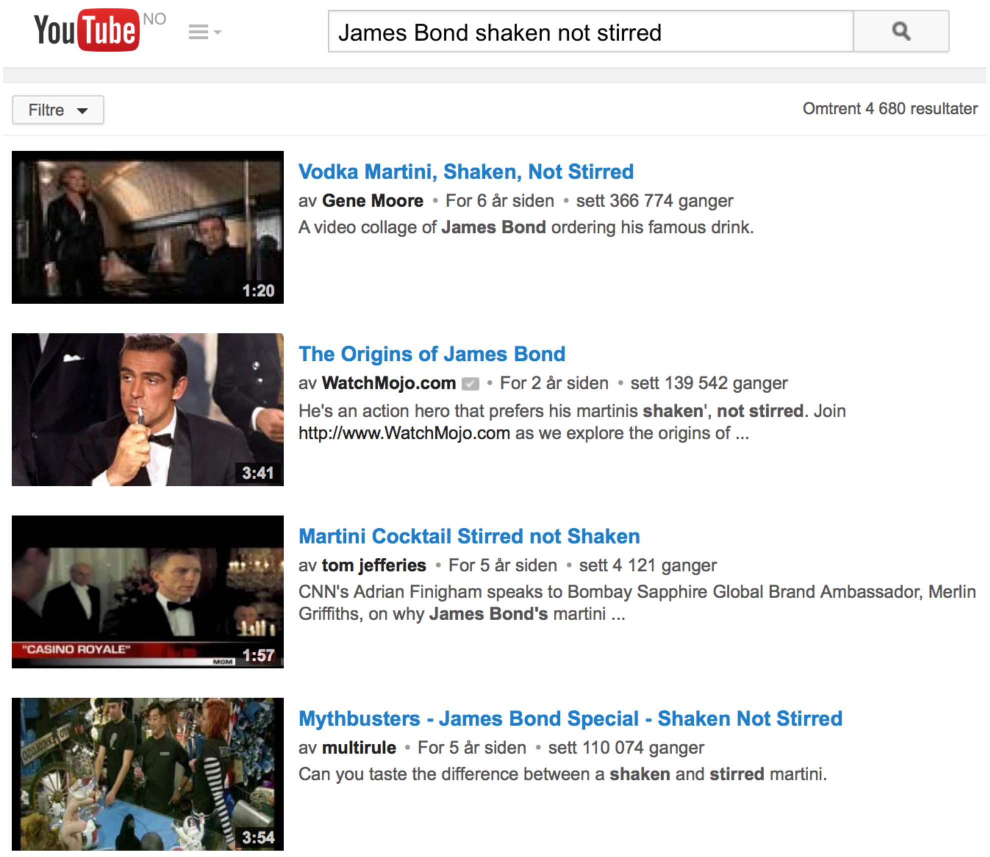


Table 4.1: "About the system" part of user study

Overall, people were positive to our system. For example, "Well-presented ... I am satisfied with the program. Especially since it is a proof-of-concept program ... Cool idea"[B.2]. All general questions in the survey received an average of approximately 5-like or better. We also wanted to know which functionality people were most satisfied with. Calculating the 7-wow responses, we found that 40 percent of the people asked found it fun to use. 58 percent found the system to respond fast to queries and playback. 63 percent were impressed with the system being accurate in finding and playing back video scenes. For example, "Works fine ... I can watch the movie I am looking for"[B.2]. Two of our primary goals with this system was fast response times and accurate mapping of metadata events to video segment ranges. This is a good indication of achieving these goals.

4.2.2 Montage comparisons MovieCutter and YouTube

The other part of the user study asked people to create montages they could compare with montages uploaded to YouTube [208]. This website has a much larger database and a search engine that is more advanced than our system. Since end users can upload their own montages in YouTube, they vary in quality. Some are carefully edited in professional editing tools, while others are of less quality. Given these differences, we still found it interesting to evaluate how people using our system would compare this with montages found in YouTube. Since we had prepared seven James Bond movies in our test database as an example of a film franchise, we suggested people using these movies for comparison. An example of a one-liner montage in YouTube is illustrated in figure 4.2.


James Bond shaken not stirred

Filtre

Omtrent 4 680 resultater

Vodka Martini, Shaken, Not Stirred
av **Gene Moore** • For 6 år siden • sett 366 774 ganger
A video collage of **James Bond** ordering his famous drink.

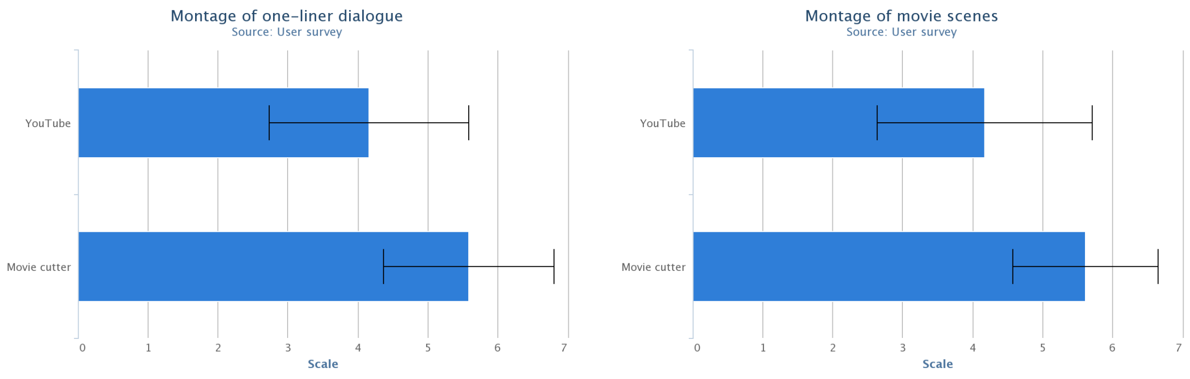
The Origins of James Bond
av **WatchMojo.com** • For 2 år siden • sett 139 542 ganger
He's an action hero that prefers his martinis **shaken**', **not stirred**. Join http://www.WatchMojo.com as we explore the origins of ...

Martini Cocktail Stirred not Shaken
av **tom jefferies** • For 5 år siden • sett 4 121 ganger
CNN's Adrian Finigham speaks to Bombay Sapphire Global Brand Ambassador, Merlin Griffiths, on why **James Bond's** martini ...

Mythbusters - James Bond Special - Shaken Not Stirred
av **multirule** • For 5 år siden • sett 110 074 ganger
Can you taste the difference between a **shaken** and **stirred** martini.

Figure 4.2: YouTube montage example [208]

The results of these montage comparisons are presented in table 4.2.



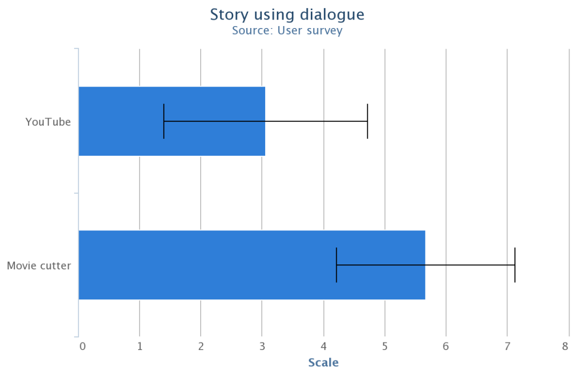


Table 4.2: "Montage comparisons MovieCutter and YouTube" part of user study

Not all participants in our survey chose to complete the YouTube comparison. Of the 25 people interviewed, 18 people used our system to do comparisons. Hence, our comparison is based on these 18 participants. Overall, people were positive to creating their own montages. For example, "Easy to make more scenes. Fast and efficient ... Creative. Good idea for new ideas ... Worked fine"[B.4]. All three comparisons received an average higher than 5-like for our system. "More flexible than YouTube ... Very good to find accurate scenes ... More specific results in single clips"[B.4]. For one-liner dialogue and movie scene montages, YouTube received an average of approximately 4-neutral. "Faster and easier to use on popular one-liners ... Stronger search engine ... Already done montages, but not movie specified."[B.4]. For creating a story using dialogue, YouTube received an average of approximately 3-dislike. "There are very few and hard to find ... Might be lucky and find something"[B.4]. The last comparison was a bit unfair, since it is difficult to find a similar story that somebody else has made. There are many carefully edited James Bond montages of high quality available on YouTube, but we can think of at least two reasons why people liked our system better for this purpose. Since YouTube provide a huge database of uploaded videos, a lot of search results have a different content than what you might be looking for. In figure 4.2 there is for example search results from a CNN [24] story and a scene from a Mythbusters [29] television program. They are related to the topic, and some might find it interesting to watch these videos as well. However, a lot of videos are off-topic and not what you are looking for. "If not everybody can upload movie clips, then the quality will be much better than YouTube. It will also be more well-presented"[B.2].

Another reason is because people are allowed to interact with our system. In the about the system survey, 40 percent gave the system a 7-wow response that it was fun to use. They like to be creative and make their own montages. "New and exciting. Very easy to use ... Definitely entertainment potential ... Funny new interaction"[B.2].

4.3 Speed/startup latency

Since our system encourages people to create their own montages, it is important that the system responds quickly to their interactions. Thankfully, the "about the system" survey showed that 58 percent gave the system a 7-wow response to fast queries and playback. For example, "Very fast and smooth ... No problem with speed or loading ... Fast start"[B.2].

We did observe that people interacted with the system differently. Some were reluctant to use the system without a proper introduction of its functionality. Originally, we wanted people to figure out the system themselves with minimal introduction. Being a good indicator on how understandable our user interface is, we planned to collect statistics based on how long it took people to get started using the system. However, since many people expected a proper introduction, we decided to drop this part of the survey.

We got feedback to improve some user interface functionality. Some users needed some time to get used to the drag and drop functionality. For example, "Did not know you should drag and drop, thought it was click based ... Maybe alter the design. Not everybody understands drag and drop."[B.2]. Others found the video handles a bit confusing. "Easy to understand, except maybe "EditStart" and "EditEnd", and which way to add seconds or subtract ... More feedback while using the video handles"[B.2]. People usually created a video summary in a short time after getting used to the drag and drop functionality.

4.3.1 Adaptiveness

Adaptive streaming is one of the reasons the system respond fast to user interactions. This technology starts playback using the lower bit rate. As the client buffers data, the player changes to an appropriate bit rate considering local CPU usage and network traffic. We implemented the system with two different bit rates to observe these changes in video quality. Such changes can easily be spotted since the lower bit rate seems constantly out of focus. We noticed this adaptiveness during the survey by running the system from our lab server. If the system were accessed from a desktop computer using a cable connection, the client player quickly adapted to a higher quality. If using a laptop with a wireless connection, the client player used the lower bit rate for a longer time period. We also tried the system in both the Norwegian towns of Oslo and Trondheim. As the server was located in Oslo, this gave us some feedback on how the system worked over distance. In all cases the video playback started almost instantly.

4.3.2 Segment duration

Another reason for fast video playback is the video segment duration. We learned that by including more I-frames in the video encoding process, we could create segments of fixed durations. I-frames are independent of images before and after in the video stream, but require more data to be streamed to an end user. It was a challenge to find an ideal duration for our system.

Apple recommends that the HLS protocol should be used with video segments of a ten seconds duration. "Here's why: if you've got live content being delivered through a CDN, there will be propagation delays, and for this content to make it all the way out to the edge nodes on the CDN it will be variable. In addition, if the client is fetching the data over the cellular network there will be higher latencies. Both of these factors make it much more likely you'll encounter a stall if you use a small target duration" [17].

Other adaptive streaming protocols such as Smooth Streaming and the Davvi prototype uses video segments of two seconds duration.

"For Davvi, we varied the size and experimented with lengths in this range to identify the optimal combination of granularity and quality. In this video search scenario, the lower range proved optimal, so Davvi uses two second segments. With shorter segment lengths, for instance one second segment, one can observe visible noise in the video playback due to flicker effects and decreased compression efficiency. With larger segments, we achieved negligible better compression, but since the search and quality adaption granularity is at segment boundaries, the two second length proved to be an efficient trade-off, i.e., a size also frequently uses in commercial systems" [63].

Both the stall argument and the search and quality adaption arguments, can have an impact on our prototype. We chose a fixed video segment duration of two seconds in our system. Hence, we choose using the HLS protocol with a shorter duration then Apple's recommendation. However, this made including functionality such as video handles [19] easier. When an end user increases or decreases a video handle slider, we simply add or subtract video segments. The system's accuracy could have been even better with video segments of one second duration, but then we would have experienced visible noise as described by the Davvi developers [63].

4.3.3 File management

We chose a two second segment duration without the HLS byte-range support. This resulted in a lot of segmented video files. For example, a movie with a duration of two hours segmented in two different bit rates, will create approximately ten thousand video segments. Add one thumbnail for each video segment, and we have additionally created approximately five thousand image files. A test database of ten movies easily exceeds one hundred thousand media files. Byte-range functionality in adaptive streaming technologies like Smooth Streaming and the newest version of HLS, decreases this number of files substantially. However, this complicated the process of creating dynamic manifest files.

During the survey we encountered some stalls from the most eager people using the system. For example, "Maybe something, but only because I impatiently stressed the system ... Rapid changes stops the system"[B]. However, we achieved a fast playback using segments of two second durations.

4.3.4 Fast raw editing capabilities

In section 1.1, we told a story about a friend creating a thoughtfully edited video summary using VHS technology. We believe he had spent many hours creating it using two VHS machines. Video editing equipment today is digital and a lot faster. However, a video resource still needs to be imported into a editing software and an editor needs some time to log the content of the video resource.

We demonstrated our system for a professional video editor with this in mind. He liked the system and told us this could be used for raw editing purposes. For example, creating a James Bond "shaken, not stirred" montage in a editing software will still require finding that specific scene within all the movies.

He liked the idea of using our system to quickly located these metadata events and afterwards export them to an editing software. For this purpose, he wanted some functionality in our system setting all video handles to a certain duration. This will enable him to trim video and audio transitions [138] in an editing software afterwards.

4.4 Future work

Functionality like globally setting all video handle durations, will improve the usability of our system. We also got a lot of interesting feedback during our survey. People interviewed expected functionality that they are used to in commercial products. For example, "Playlist should be horizontal, resembles an editing program. Used to this way of editing ... Information about text-based searches. Add help button"[B.6]. We will now discuss how to improve different components in our system.

4.4.1 Metadata resources

Our system needs more metadata resources. We have given examples of tribute movie websites with all kinds of information about movies. It is challenging to find websites containing both interesting metadata and specific time codes. Alternatively, we can create metadata manually. We have mentioned companies that use professional loggers to organize metadata as they see fit. With such resources, it is be possible to find almost any metadata event within a collection of movies.

Some improvements can be made using existing metadata in the system. For example, IMDb contains a few interesting topics about movies in their database. We tried "Quotes", a list of the most popular dialogue phrases within our test movie "From Russia with Love". For example, "(Rosa Klebb:)Training is useful, but there is no substitute for experience. (Morzeny:)I agree: we use live targets as well" [50]. We can match these phrases to our subtitle metadata resource, and rank these phrases higher than other subtitle events. Ranking functionality has already been prepared in our metadata table. Longer conversations needs to be merged into new metadata events. We cannot know what kind of dialogue every end user wants, but it is more likely that the most popular quotes catches their attention.

Another topic in IMDb is "Goofs". Using the same movie as example, we learn about scenes where something did not go as planned. "During the battle at the gypsy camp, Kerim Bey is shot in his right arm. You can clearly see him smearing fake blood on his sleeve with his left hand" [49]. We can manually log such goofs, or add functionality to the system that lets end users register such events. This also allows end users to share their own playlist with other users of the system. Such a feature will add interesting montages to the prototype.

4.4.2 Adaptiveness

Adaptive streaming technology changes bit rates based on network traffic and CPU conditions in receiving devices. We only used two different bit rates to demonstrate the system's adaptiveness. By creating video segments of multiple qualities, the system can provide a smooth user experience for both mobile devices and larger computers like desktops.

4.4.3 User interface

We got a lot of feedback on our user interface. People accepted that our prototype is a proof of concept application, but missed some of the functionality they used to from commercial applications. However, some of the feedback was a result of personal preferences. For example, one participant wanted us to "Add play button on each listing in the search result and playlist component", while another participant wanted "double-clicking on search result list should playback the item"[B.6]. Both feedback represent different ways of playing back a metadata event. Not everybody liked our drag and drop functionality, while others found it easy to use.

The system's user interface can also be adjusted for smart phones using responsive web design [181]. This means that the layout of web page will be adjusted dynamically, taking into account the characteristics of the device used (PC, tablet, mobile phone) [130]. For example, Bootstrap [20] is an open source front-end framework that supports responsive design. "It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components" [130]. Using such a framework in combination with an interoperable adaptive streaming protocol, will make our system accessible on almost any platform.

4.4.4 Interoperability

We have used the HLS adaptive streaming technology. During the development of this prototype, we have learned that the protocol offers sophisticated streaming solutions. The protocol works natively on Apple platforms such as the Safari browser. Optionally, we can use a plug-in technology like the JWPlayer [68] to playback HLS in other web browsers. This does not work on platforms that does not support Flash based plug-in technology like newer smart phones using the Android OS. As an alternative, we have discussed the emergence of the MPEG-DASH standard. This technology supports interoperability among the platforms. It can be interesting to redevelop the system using MPEG-DASH. Such an implementation allows for example more client player configuration than the HLS prototcol.

4.4.5 Languages

Our system could support multiple languages. For example, subtitle online databases such as Subscene includes subtitles in many languages. One participant in our survey pointed out that we should "add possibility for multiple languages / dubbing. It is fun to see how dialogue are interpreted in another language"[B.6]. Some of the system's program logic and database structure needs to be altered to support this feature.

4.4.6 Search optimization

We can add more search alternatives and implement an advanced search engine in the system. For example, an improvement in search alternatives is actor-based searches. Mapping actors to certain dialogue can provide one-liner search results. In subsection 2.16.4 we used an example with the actor Bruce Willis and the Die Hard franchise. His famous phrase "Yippee-ki-yay..." is used in every Die Hard movie. We can avoid off-topic search results like old western movies by combining the actor's name and the phrase in the search.

Our system does not use an advanced search engine. It uses wild cards on a column in a database table. Issues such as ranking of search results and filtering away unnecessary words or sentences is not supported yet. Adding an open source search engine like Apache Lucene in the prototype can help optimize our search queries. "The Lucene Core provides Java-based indexing and search technology, as well as spellchecking, hit highlighting and advanced analysis/tokenization capabilities" [12]. We have listed of some of the Apache Lucene features:

- Ranked searching – best result returned first
- Many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more
- Fielded searching (e.g. title, author, contents)
- Sorting by any field
- Multiple-index searching with merged results
- Allows simultaneous update and searching
- Fast, memory-efficient and typo-tolerant suggesters

Combining Lucene with Apache Solr [11] provides a high performance searching server. Our automated playlist generator will for example benefit from Lucene's ranked searching feature. Implementing Apache Lucene in our system will require changes to our database structure.

4.4.7 Graph Database

We are using MySql relational database in our system. It is possible to include for example ranking functionality using our system as it is. However, Apache Lucene is a tempting alternative. Another way of thinking is using graph databases. It is a database that uses graph structures with nodes, edges, and properties to represent and store data. "A graph database is any storage system that provides index-free adjacency. This means that every element contains a direct pointer to its adjacent elements and no index lookups are necessary" [142]. A Graph database is efficient when joining many tables. Such databases can for example used in social media applications. Noe4j [85] is an open source graph database. The Swedish company developing this product has created a tutorial using a movie graph database. Such a database can be useful for actor-based searches. In a large video archive an experienced actor or actress will probably appear in a wide range of movies. This might increase lookup times when joining tables using a relational database. Neo4j's movie graph database is very efficient for such purposes.

4.4.8 Video handles

We can optimize the duration of the video handles. Short metadata events such as dialogue, need video handles with shorter duration than chapter events. Instead of two separate sliders, we can combine this functionality into a single slider with two handles. A participant in the survey pointed out that "a double slider is more easy-to-grasp"[B.6]. Another participated wanted the "possibility to regret a slider change"[B.6]. Additionally, we can include a drop down list setting all video handles to a given duration. Combining this with an export video button can improve an editing workflow.

4.4.9 Video segment duration

Optimization is possible by using milliseconds when calculating video segment ranges. The current system does not consider milliseconds when converting time codes to seconds. This mean the video segment range might be calculated with a small offset. If searching for a single word like "Bond", this word can be in the beginning or the end of a subtitle event. Splitting the subtitle sentence into a array, it is possible to calculate at what particular video segment the "Bond" word is said. However, the current system have an accuracy of more or less two seconds due to the fixed video segment duration.

4.5 Summary

Our system still need improvement. Using the right technologies will result in a more robust and user friendly system. Video segments of multiple qualities will give a better viewing experience. Responsive design and interoperability makes the user interface available on more platforms. Apache Lucene can optimize our search queries. Graph databases is also an alternative. Configuring video handles can improve metadata events accuracy. Optimizing the video segment duration can also result in more specific search results. Most importantly, more metadata resources makes the system more interesting to use. This can be done manually, by the users of the system or by researching more online resources. In the next chapter, we will summarize this thesis.

Chapter 5

Conclusion

We have pointed out possible improvements to our system in the previous chapter. However, our current system is a proof of concept of the problems we set out out to solve.

5.1 Summary

We stated in section 1.2 that our overall objective was to create a system where an end user can arbitrary search for content and on the fly generate a personalized video summary in a matter of seconds. We made a list of goals to achieve this.

- "Compare streaming technologies". We researched how streaming has changed from using traditional streaming protocols like RTSP [42] to deliver content over HTTP [150]. Then, we explained why segmenting a movie into small segments in multiple qualities deliver a smooth experience for an end user. Two proprietary adaptive streaming technologies were compared with the emerging streaming standard MPEG-DASH [28, 99, 117].

- "Use a streaming protocol for our purposes". Having decided upon HLS [13, 14, 31, 95] as our adaptive streaming protocol, we researched the protocols tech notes [17, 18] for clues on how to create dynamic manifests. We experimented with creating playlist until we successfully managed to dynamically create variant manifests of different bit rates.

- "Research possible metadata resources". We researched how to locate scenes within a movie using both internal techniques such as image analysis and external techniques like separate descriptive metadata files. Then, we researched specific movie metadata resources. Finally, we successfully mapped these metadata resources to our video segment ranges.

- "Use different searching techniques". We create a text based search engine using wildcards. This search technique returned search results of both dialogue and chapter scenes within movies. Optionally, we also created a click based search engine based on genre. This option can be used to locate chapter scenes.

- "Create a website with search and playback capabilities". We have created a website for utilizing the system capabilities. By using drag and drop functionality, an end user can preview movie scenes and create a montage in a playlist. It is possible to edit the duration of metadata events.

- "Do a user study to further improve the system". This survey gave us a lot of useful feedback about our system strengths and possible improvements. Given that 58 percent was very impressed with the system responsiveness to queries and playback, is a good indication of reaching our main objective.

5.2 Main Contributions

We have created a system for extracting events within a video archive. This means we can search for a specific scene within a movie and extract that particular event. An end user can then create a personalized video summary. By dragging multiple events into a playlist, an end user can rearrange and customize the durations of events. Afterwards, this video summary can be played back using adaptive streaming.

5.2.1 Components

- Extracting accurate events is possible using our metadata extraction component. We parse text-based documents containing time codes and a textual description of an event. Metadata resources such as subtitle and chapter events are downloaded from online movie databases. During parsing, extracted metadata are stored in a relational database.

- Our search component includes a text based search engine using wild cards. It is also possible to search for movies and chapters using genre selection. An extracted event is then presented with a storyboard thumbnail image and information about that particular event. Such information includes event type, duration and a textual description.

- An end user can preview an extracted event by dragging it into the video player in the user interface. Alternatively, an end user can also create a personalized video summary using the system's playlist component. By dragging multiple events into a playlist, an end user can rearrange and customize the durations of events. Either way, the system creates manifest files to playback events. A manifest file is a playlist of file path references to video segments on the server. This function calculates one or many events included in a request from a client device. Then, it creates three temporary manifest files to prepare a video summary for adaptive streaming. A variant manifest file contains some adaptive logic and the file paths references to other manifest files of a low and high bit rate. The URL of the variant manifest file is returned to the client video player for play back.

5.2.2 Evaluation

We have performed a user study among both technical and non-technical persons to evaluate this system. During this survey, the participants were asked to create video montages and give feedback about the system's functionality. Overall, people were positive to our system. All general questions in the survey received a high score in the Likert scale. We scored especially high on statements like "It is fun to use", "It responds fast to queries and playback" and "It plays back accurate video scenes".

5.2.3 Conclusion

Low latency and accuracy were two of our primary goals with this system. We have implemented a running prototype that can reedit multiple movies in a matter of seconds. The system is able to present specific scenes within a movie by mapping accurate metadata with video segments. These events from multiple movies can then be rearranged in a playlist before being played back in the video player. The video player changes between different video qualities using adaptive streaming. On the basis of subtitle and chapter metadata resources, we can create our own directors cut.

5.3 Ongoing work

Our system can be used for different purposes. First, we will describe a use case of including our prototype into another research project. Later, we will describe some possible commercial use cases. Finally, we will list a few examples of using our system for storytelling in general.

5.3.1 Bagadus

In section 2.11, we introduced the sports analytics application Bagadus. Our system will be integrated into Bagadus and used for automatic event extraction purposes. Bagadus uses a sensor system to collect metadata about each player on the pitch. Such metadata can be used to find interesting events in soccer matches.

Coaches and soccer analytics might want to review certain situations. For example, event extraction can be used to evaluate offside situations. Some teams uses an offside trap [170] as a strategy during a match. This requires all players in defense to cooperate to form a trap for an opposite team player. A coach might want to extract previous offside situations to show and teach defense players how to do this efficiently.

We hope our system will provide extra functionality to the Bagadus application.

5.3.2 VOD streaming

This system offers a flexible tool for handling video. Many broadcasting and VOD companies have large video archives. It is a promising thought to be able to access specific parts of these videos resources instantly. For example, a streaming company like Netflix [86] can offer such functionality as an extra service for their customers. If an end-user only want to watch a specific scene in a movie, they can get instant access to it.

Netflix already uses adaptive streaming [27], so video resources in their archive are already cut into small segments of different quality. Some integration will be necessary. Netflix uses the Microsoft's Silverlight media player plug-in for some devices, and wants to replace it with a trio of HTML5 extensions [69]. Meaning, to use functionality from our prototype in their system, our system's streaming component HLS needs to be replaced with another streaming technology. In section 2.13, we described the end-to-end prototype Davvi [63, 64] that was integrated with a Microsoft technology to become another system VESP [39, 40, 63]. This required that Davvi's search component was integrated with Mircosoft's FAST enterprise search platform (ESP) [72]. Integrating necessary components enables functionality in our prototype to be used in a streaming company system.

5.3.3 Raw editing and promos

Another usage of our system can be for raw editing purposes. Editors, journalists or directors often have an idea of a final cut. They do not know if this cut will work before they watch the result. Our system gives them an opportunity to preview what it will look like. Creating a new video from parts of archived videos on the fly, saves both ingesting and editing time. Editing a movie is time-consuming and expensive. The movie needs to be imported into special software. An editor uses a timeline window in the software to do nonlinear changes. Afterwards, the final cut needs to be rendered to a desired format.

Using adaptive streaming, we already have access to pre-rendered movies of multiple formats. This requires of course relevant metadata. Video resources in the video archive needs to be properly logged to able to extract usable video summaries. A movie promo company can use our metadata resources like subtitles and chapters. Others, for example a sport promo company, needs to use other metadata resources such as they did with the Davvi prototype.

5.3.4 Other use cases

We wanted to help people illustrate their storytelling using video segments. Assuming relevant metadata is available for these purposes, then how can people use our system for storytelling?

- In section 1.1, we told a story about a friend who wanted to make a video highlight montage of his favorite movies. Our prototype could have helped him tell his story in seconds instead of hours.
- We also mentioned a storyteller wanting to illustrate key points to his audience. He could have used our system in preparation of his story, or while presenting his material.
- An editor is given the task of ingesting every James Bond movie and create a highlight movie of James Bond saying "shaken, not stirred". Why? Because somebody thinks it funny and is willing to pay money for it. Our system can edit this James Bond example in seconds.
- We should also mention some examples using different video resources than movies. For example, a reporter did a long interview with the United State President Barack Obama some years ago. Now, he want to use parts of this interview for a follow-up story on an environmental issue. With our system, he can access that part of the interview instantly.
- Today, many institutions such as the police, fire department, hospitals, coastguard or army uses video to document events. Given a new event, it can be helpful for the officer in charge to instantly playback how the previous event was resolved. If the previous event was properly logged, the officer can search for a critical part of the situation. For example, the coastguard had to board a foreign fishing boat. On board, they needed to confront the captain. Given a similar situation in an ongoing event, this recording can help the officer resolve the situation in a best way possible. Time is usually a critical factor in such events. Our system leverages instant feedback.

These are just a few examples. We cannot predict how people will use this system. New technology always spark creativity and they will think of use cases we cannot imagine. Our main concern is that this system helps them tell their story.

Appendix A

Code

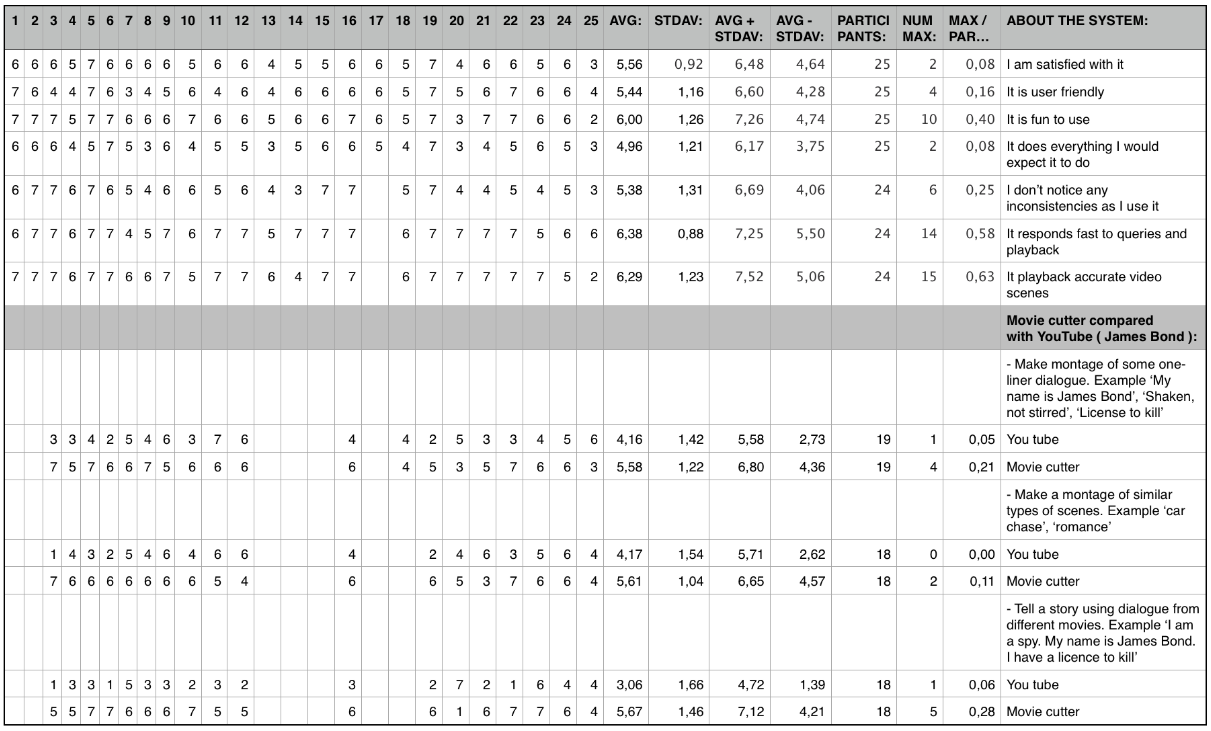
The system code and "how to" guide can be found in a Bitbucket repository:

`https://bitbucket.org/mpg_code/movie-cutter`

Appendix B

User study

B.1 User study raw data table



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	AVG:	STDAV:	AVG + STDAV:	AVG - STDAV:	PARTICI PANTS:	NUM MAX:	MAX / PAR…	ABOUT THE SYSTEM:
6	6	6	5	7	6	6	6	6	5	6	6	4	5	5	6	6	5	7	4	6	6	5	6	3	5,56	0,92	6,48	4,64	25	2	0,08	I am satisfied with it
7	6	4	4	7	6	3	4	5	6	4	6	4	6	6	6	6	5	7	5	6	7	6	6	4	5,44	1,16	6,60	4,28	25	4	0,16	It is user friendly
7	7	7	5	7	7	6	6	6	7	6	6	5	6	6	7	6	5	7	3	7	7	6	6	2	6,00	1,26	7,26	4,74	25	10	0,40	It is fun to use
6	6	6	4	5	7	5	3	6	4	5	5	3	5	6	6	5	4	7	3	4	5	6	5	3	4,96	1,21	6,17	3,75	25	2	0,08	It does everything I would expect it to do
6	7	7	6	7	6	5	4	6	6	5	6	4	3	7	7		5	7	4	4	5	4	5	3	5,38	1,31	6,69	4,06	24	6	0,25	I don't notice any inconsistencies as I use it
6	7	7	6	7	7	4	5	7	6	7	7	5	7	7	7		6	7	7	7	7	5	6	6	6,38	0,88	7,25	5,50	24	14	0,58	It responds fast to queries and playback
7	7	7	6	7	7	6	6	7	5	7	7	6	4	7	7		6	7	7	7	7	7	5	2	6,29	1,23	7,52	5,06	24	15	0,63	It playback accurate video scenes
																																Movie cutter compared with YouTube (James Bond):
																																- Make montage of some one-liner dialogue. Example 'My name is James Bond', 'Shaken, not stirred', 'License to kill'
		3	3	4	2	5	4	6	3	7	6				4		4	2	5	3	3	4	5	6	4,16	1,42	5,58	2,73	19	1	0,05	You tube
		7	5	7	6	6	7	5	6	6	6				6		4	5	3	5	7	6	6	3	5,58	1,22	6,80	4,36	19	4	0,21	Movie cutter
																																- Make a montage of similar types of scenes. Example 'car chase', 'romance'
		1	4	3	2	5	4	6	4	6	6				4			2	4	6	3	5	6	4	4,17	1,54	5,71	2,62	18	0	0,00	You tube
		7	6	6	6	6	6	6	6	5	4				6			6	5	3	7	6	6	4	5,61	1,04	6,65	4,57	18	2	0,11	Movie cutter
																																- Tell a story using dialogue from different movies. Example 'I am a spy. My name is James Bond. I have a licence to kill'
		1	3	3	1	5	3	3	2	3	2				3			2	7	2	1	6	4	4	3,06	1,66	4,72	1,39	18	1	0,06	You tube
		5	5	7	7	6	6	6	7	5	5				6			6	1	6	7	7	6	4	5,67	1,46	7,12	4,21	18	5	0,28	Movie cutter

Table B.1: User study raw data

B.2 User study: "About the system" user comments

- "I am satisfied with it":

```
- Cool program.
- Cool idea.
- Maybe an even better search engine.
- Make it compatible with YouTube!
- I am satisfied with the program. Especially since it is a
proof-of-concept program.
- Well-presented.
- Very fun to use.
- Just small improvements needs to be made.
```

- "It is user friendly":

```
- Easy, maybe Norwegian movies should be included in the
database.
- Did not know you should drag and drop, thought it was click
based.
- Easy to get access to the video segment ranges you need.
- Very user friendly.
- Easy to understand, except maybe "EditStart" and "EditEnd",
and which way to add seconds or subtract.
- I am missing "active" movie clips ( highlighting ) when I drag
a clip into the player.
- Explain duration.
- Nothing negative.
- Clarify what I am supposed to do.
- Maybe alter the desing. Not everybody understands drag and drop
functionality.
- More feedback while using the video handles.
- Easy-to-grasp, but maybe more explanation for each container
component.
```

- "It is fun to use":

```
- Fast to playback the movie clips / movies you want to watch.
- New and exciting. Very easy to use.
- Could have been a good plugin application to YouTube.
- Easy to use.
- I want to add my own music to the playlist.
- Hilarious funny to use.
- Defenitiely entertainment potential.
- Funny new interaction
```

- "It does everything I would expect it to do":

```
- Some of the buttons are unintuitive, and lack description.
- Add a function that searches through only one movie.
- Given a proof-of-concept application, I am happy with it.
- Easy to find clips, but where in the movie the clip belongs,
is not specified.
- It is a POC(L).
- Easy and ok.
- My requirement is to find the scene I am looking for. It does
that.
```

- "I don't notice and inconsistencies as I use it":

```
- If not everybody can upload movie clips, then the quality
will be much better than Youtube. It will also be more
well-presented.
- I could not notice any.
- Some scenes were indexed wrong.
- Maybe something, but only because I impatiently stressed
the system.
- Did not notice any types of bugs. Worked smoothly.
- Bug: a movie clip should not be given a duration of 0
seconds.
```

- "It responds fast to queries and playback":

```
- Very fast and smooth.
- No problem with speed or loading.
- Fast start
```

- "It playback accurate video scenes":

```
- I can watch the movie I am looking for
- Works fine
- Nice with video handles. Maybe better with one large handle
than two separate handles.
```

B.3 User study: "Montage one-liner dialogue" user comments

- "YouTube":

```
- Already done montages, but not movie specified.
- Stronger search engine
- Easier text based searches
- Definitely not suited for this purpose
- Faster and easier to use on popular one-liners
```

- "Movie cutter":

```
- More flexible than YouTube.
- Very good to find accurate scenes.
- More specifies results in single clips.
- Creative. Good idea for new ideas.
- Worked fine.
```

B.4 User study: "Montage movie scenes" user comments

- "YouTube":

```
- More information on what happens in the movie clip.
- Both visual and text specified. Could be more info etc.
```

- "Movie cutter":

```
- Add music videos to the database.
- Only text specified search results, but more info.
- Fun idea. Great and something new for the end-users.
- Advantage: Tailormade.
- Disadvantage: Time-consuming.
```

B.5 User study: "Story using dialogue" user comments

- "YouTube":

```
- There are very few and hard to find.
- Might be lucky and find something
```

- "Movie cutter":

```
- Easy to cut and add clips together, but still hard to search
for for example explosion scenes, because it is visual.
- Hard to divide whole chapters ( expand content of chapter ).
- Possible, but with 2 second duration clips a bit hard.
- Easy to make more scenes. Fast and efficient.
- More adapted for this purpose.
- Some small glitches, but works surprisingly well for being a
proof-of-concept application.
- Ok tool
```

B.6 User study: "Misc tips about user interface and functionality" user comments

- Buttons belongs to the player component.
- Needs information about drag and drop functionality.
- Add music functionality.
- Only one search option. Messy with two different options.
- Playlist should be horizontal, resembles editing program.
Used to this way of editing.
- Information about text based searches. Add help button.
- Difficult to know search options.
- Explain duration video clips.
- Add search button on category / select movie to avoid loosing
previous search results.
- text to movie support (HTML5 video tag/player support subtitle
functionality).
- Make it possible for end-users to upload their own movies / share
with others.
- Logic structure on webpage.
- Double slider more easy-to-grasp, change delta borders and
differentiate them depending on if it is subtitles or chapters.
- Use larger text in the webpage.
- Dynamically enhance what you are working with, for example
fill entire page with component you are working on.
- Add button as alternative to drag and drop.
- Search on dialogue inside a chapter.
- Include "all movie" search as category option.
- Add play button on each listing in the search result and playlist
component.
- Add advanced search / basic search
- Possible to type only part of a sentence in the search area,
add autocomplete.
- Possible to tag and save own playlists, caters for special needs.
- Drag and drop from the search result list to player should
highlight the clip (as in the playlist).
- Double-clicking on search result list should playback the item.
- Explain the video handles / sliders
- Rapid changes stops the system
- Where in the movie are the movie clips located?
- Possibility to regret a slider change.
- Highlighting changes item in playlist at wrong time because of
slider changes.
- Search dialogue in a specified chapter.
- Expand, divide a chapter into all video segments. Makes it
possible to build a playlist with visual imagery like explosions
an so on.
- YouTube wins because of the search engine. Important that people
can upload searchable material.
- If same movie clip in the playlist multiple times, will not play
all the time.
- Use double slider to avoid duration of 0.
- With YouTube you can search for entire movie titles.
In Movie cutter you search for parts of a movie.

- Add possibility for multiple languages / dubbing. It is fun to see how dialogue are interpreted in another language.
- Add tags. Adds search possibilities.
- Add possibility to search entire movie.
- Possible to search for time inside the movie. Can be done inside the player, but should also be possible via text based searches / drop down possibility.
- Add tutorial function and this will make the system easier to grasp for end-users.
- Add selected video handle length for all clips in playlist. This in combination with export video button, makes the system idle for raw editing purposes.

Bibliography

[1] 3GPP specification, April 2014. URL: http://www.3gpp.org/DynaReport/26247.htm.

[2] Adobe, April 2014. URL: http://www.adobe.com/.

[3] Adobe. Adobe Access, April 2014. URL: http://www.adobe.com/no/products/adobe-access.html/.

[4] Adobe. Flash Player, April 2014. URL: http://www.adobe.com/no/products/flashplayer.html.

[5] Adobe. HDS Dynamic Streaming, April 2014. URL: http://www.adobe.com/products/hds-dynamic-streaming.html.

[6] Amazon Expands X-Ray to TV, Powered by IMDb and Available on the Kindle Fire Family, August 2013. URL: http://phx.corporate-ir.net/phoenix.zhtml?c=176060&p=irol-newsArticle&ID=1801001.

[7] Amazon, April 2014. URL: http://www.amazon.com/.

[8] Amazon. Kindle, April 2014. URL: https://kindle.amazon.com/.

[9] Steve Anderson. Die Hard 4 presidential montage, March 2014. URL: http://www.criticalcommons.org/Members/ironman28/clips/dieHardPresidentialMontageCutup.mp4/view.

[10] Apache HTTP Server, April 2014. URL: http://httpd.apache.org/.

[11] Apache Solr, April 2014. URL: https://lucene.apache.org/solr/.

[12] Apache Lucene, March 2014. URL: https://lucene.apache.org/.

[13] Apple. HTTP Live Streaming Overview pdf, December 2013. URL: http://developer.apple.com/library/ios/documentation/networkinginternet/conceptual/streamingmediaguide/StreamingMediaGuide.pdf.

[14] Apple. HTTP Live Streaming Overview webpage, June 2013. URL: https://developer.apple.com/library/ios/documentation/networkinginternet/conceptual/streamingmediaguide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40008332-CH1-SW1.

[15] Apple developer website, April 2014. URL: https://developer.apple.com/.

[16] Apple. Safari, April 2014. URL: http://www.apple.com/safari/.

[17] Apple. Technical Note TN2224, February 2014. URL: https://developer.apple.com/library/ios/technotes/tn2224/_index.html.

[18] Apple. Technical Note TN2288, February 2014. URL: https://developer.apple.com/library/ios/technotes/tn2288/_index.html.

[19] Kylene Arnold. What are edit handles?, April 2014. URL: `http://www.ehow.com/info_8185004_edit-handles.html`.

[20] Bootstrap, April 2014. URL: `http://getbootstrap.com/`.

[21] Briefcam White Paper, March 2014. URL: `http://www.briefcam.com/PDFs/White_Paper_new.pdf`.

[22] ChapterDb, April 2014. URL: `http://www.chapterdb.org/`.

[23] ChapterDb example, March 2014. URL: `http://www.chapterdb.org/browse/43138`.

[24] CNN, April 2014. URL: `http://edition.cnn.com/`.

[25] Aaron Colwell, Adrian Bateman, and Mark Watson. Media Source Extensions, December 2013. URL: `https://dvcs.w3.org/hg/html-media/raw-file/tip/media-source/media-source.html`.

[26] D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Communications of the ACM*, 32(1):9–23, January 1989. URL: `http://dl.acm.org/citation.cfm?id=63239`.

[27] Chris Crum. Not everyone can get better Netflix picture quality, April 2014. URL: `http://www.webpronews.com/now-everyone-can-get-better-netflix-picture-quality-2013-09`.

[28] Dash Industry Forum, December 2013. URL: `https://github.com/Dash-Industry-Forum/dash.js/wiki`.

[29] Discovery. Mythbusters, April 2014. URL: `http://www.discovery.com/tv-shows/mythbusters`.

[30] Troy Dreier. Understanding DASH264 and Building a Client, October 2013. URL: `http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/Understanding-DASH264-and-Building-a-Client-92727.aspx`.

[31] Encoding. The Definitive Guide to HLS, June 2013. URL: `http://features.encoding.com/http-live-streaming-hls/`.

[32] FFprobe, April 2014. URL: `http://www.ffmpeg.org/ffprobe.html`.

[33] FFmpeg synopsis, March 2014. URL: `http://www.ffmpeg.org/ffmpeg.html#Synopsis`.

[34] Jonathan Foote and Andreas Girgensohn. Video classification using transform coefficients. *Proc. ICASSP'99*, 6:3045–3048, 1999. URL: `http://www.google.no/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CDMQFjAA&url=http%3A%2F%2Fwww.researchgate.net%2Fpublication%2F3794110_Video_classification_using_transform_coefficients%2Ffile%2F79e4150d34039f1402.pdf&ei=uG5NU6Jh69XgBPWAgYgF&usg=AFQjCNFbztrRd2dWhpqF7WbPEZSq8EdY3A&bvm=bv.64764171,d.bGE`.

[35] Jan Gunnar Furuly. De er filmhistoriens mest blodtorstige skuespillere, April 2014. URL: `http://www.aftenposten.no/kultur/De-er-filmhistoriens-mest-blodtorstige-skuespillere-7456240.html`.

[36] Vamsidhar Reddy Gaddam, Ragnar Langseth, Sigurd Ljødal, Pierre Gurdjos, Vincent Charvillat, Carsten Griwodz, and Pål Halvorsen. Interactive Zoom and Panning from Live Panoramic Video. In *Proc. of ACM NOSSDAV*, pages 19:19–19:24, 2014. URL: `http://home.ifi.uio.no/paalh/publications/files/nossdav2014.pdf`.

[37] Google, April 2014. URL: http://www.google.com.

[38] Google. Widewine, April 2014. URL: http://www.widevine.com/.

[39] Pål Halvorsen, Dag Johansen, Bjørn Olstad, Tomas Kupka, and Sverre Tennøe. vESP: A Video-Enabled Enterprise Search Platform. In *Proc. of ICDKE*, pages 534–541, 2010. URL: http://home.ifi.uio.no/paalh/publications/files/icdke2010.pdf.

[40] Pål Halvorsen, Dag Johansen, Bjørn Olstad, Tomas Kupka, and Sverre Tennøe. vESP: Enriching Enterprise Document Search Results with Aligned Video Summarization. In *Proc. of ACM MM*, pages 1603–1604, 2010. URL: http://www.researchgate.net/publication/221573410_vESP_enriching_enterprise_document_search_results_with_aligned_video_summarization.

[41] Pål Halvorsen, Simen Sægrov, Asgeir Mortensen, David K.C. Kristensen, Alexander Eichhorn, Magnus Stenhaug, Stian Dahl, Håkon Kvale Stensland, Vamsidhar Reddy Gaddam, Carsten Griwodz, and Dag Johansen. BAGADUS: An Integrated System for Arena Sports Analytics ? A Soccer Case Study. In *Proc. of ACM MMSys*, pages 48–59, March 2013. URL: http://home.ifi.uio.no/paalh/publications/files/mmsys2013-bagadus.pdf.

[42] IETF. Real Time Streaming Protocol, April 2014. URL: http://www.ietf.org/rfc/rfc2326.txt.

[43] IMDb. 300, April 2014. URL: http://www.imdb.com/title/tt0416449/?ref_=nv_sr_2.

[44] IMDb. Arnold Schwarzenegger, April 2014. URL: http://www.imdb.com/name/nm0000216/?ref_=nv_sr_1.

[45] IMDb. Bruce Willis, April 2014. URL: http://www.imdb.com/name/nm0000246/?ref_=nv_sr_1.

[46] IMDb. Citizen Kane, March 2014. URL: http://www.imdb.com/title/tt0033467/?ref_=nv_sr_21.

[47] IMDb. Die Hard, April 2014. URL: http://www.imdb.com/title/tt0095016/?ref_=nv_sr_1.

[48] IMDb. From Russia with love, April 2014. URL: http://www.imdb.com/title/tt0057076/?ref_=nv_sr_1.

[49] IMDb. From Russia with love - Goofs, April 2014. URL: http://www.imdb.com/title/tt0057076/trivia?tab=gf&ref_=tt_trv_gf.

[50] IMDb. From Russia with love - Quotes, April 2014. URL: http://www.imdb.com/title/tt0057076/quotes?ref_=tt_ql_3.

[51] IMDb. Goldfinger, April 2014. URL: http://www.imdb.com/title/tt0058150/?ref_=nv_sr_1.

[52] IMDb, April 2014. URL: http://www.imdb.com/.

[53] IMDb. Live Free or Die Hard, April 2014. URL: http://www.imdb.com/title/tt0337978/?ref_=fn_al_tt_1.

[54] IMDb. Most popular by Genre, April 2014. URL: http://www.imdb.com/genre/.

[55] IMDb. Orson Welles, April 2014. URL: http://www.imdb.com/name/nm0000080/?ref_=nv_sr_1.

[56] IMDb. Pulp Fiction, March 2014. URL: http://www.imdb.com/title/tt0110912/?ref_=nv_sr_1.

[57] IMDb. Quentin Tarantino, April 2014. URL: http://www.imdb.com/name/nm0000233/?ref_=nv_sr_1.

[58] IMDb. Rebel Without a Cause, April 2014. URL: http://www.imdb.com/title/tt0048545/?ref_=nv_sr_1.

[59] IMDb. Sean Connery, April 2014. URL: http://www.imdb.com/name/nm0000125/?ref_=nv_sr_1.

[60] IMDb. The Matrix, April 2014. URL: http://www.imdb.com/title/tt0133093/?ref_=nv_sr_1.

[61] IMDb. Uma Thurman, April 2014. URL: http://www.imdb.com/name/nm0000235/?ref_=nv_sr_1.

[62] IMSDb, August 2013. URL: http://www.imsdb.com/.

[63] Dag Johansen, Pål Halvorsen, Håvard Johansen, Haakon Riiser, Cathal Gurrin, Bjørn Olstad, Carsten Griwodz, Åge Kvalnes, Joseph Hurley, and Tomas Kupka. Search-based composition, streaming and playback of video archive content. *Multimedia Tools and Applications*, 2011. URL: http://www.google.no/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=0CEEQFjAC&url=http%3A%2F%2Fwww.researchgate.net%2Fpublication%2F226842506_Search-based_composition_streaming_and_playback_of_video_archive_content%2Ffile%2F72e7e52b160a2eda3e.pdf&ei=9bNOU9SrO-KBywOC7YDQCA&usg=AFQjCNFyrUkzc5ZwIJOsnAuToIgmgDp8ZA&bvm=bv.64764171,d.bGQ.

[64] Dag Johansen, Håvard Johansen, Tjalve Aarflot, Joseph Hurley, Åge Kvalnes, Cathal Gurrin, Sorin Sav, Bjørn Olstad, Erik Aaberg, Tore Endestad, Haakon Riiser, Carsten Griwodz, and Pål Halvorsen. DAVVI: A Prototype for the Next Generation Multimedia Entertainment Platform. In *Proc. of ACM MM*, pages 989–990, October 2009. URL: http://hjem.ifi.uio.no/paalh/publications/files/acmmm09-demo.pdf.

[65] jQuery, April 2014. URL: http://jquery.com/.

[66] jQuery user interface, April 2014. URL: https://jqueryui.com/.

[67] JSON, April 2014. URL: http://www.json.org/.

[68] JW Player, April 2014. URL: http://www.jwplayer.com/.

[69] Gregg Keizer. Netflix to dump Silverlight, Microsoft's stalled technology, April 2013. URL: http://www.computerworld.com/s/article/9238421/Netflix_to_dump_Silverlight_Microsoft_s_stalled_technology.

[70] Chris Knowlton. Adaptive streaming comparison, March 2014. URL: http://www.iis.net/learn/media/smooth-streaming/adaptive-streaming-comparison.

[71] Microsoft. Search lifecycle Microsoft products, November 2013. URL: http://support.microsoft.com/lifecycle/search/default.aspx?sort=PN&qid=&alpha=Silverlight&Filter=FilterNO/.

[72] Microsoft FAST ESP, April 2014. URL: http://www.google.no/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&ved=0CEcQFjAD&url=http%3A%2F%2Fdownload.microsoft.com%2Fdownload%2F1%2F4%2F8%2F1483939B-15B8-4DD3-B06D-204D03EC8A1E%2FFast_ESP_Prod_Guide.pdf&ei=ZbZOU8aYAqi6yAPgvYBI&usg=AFQjCNENWcOPR0uxGeapoe_Mfjc_W4lPJA&bvm=bv.64764171,d.bGQ.

[73] Microsoft. Playready, April 2014. URL: https://www.microsoft.com/playready/.

[74] Microsoft. Silverlight, April 2014. URL: http://www.microsoft.com/silverlight/.

[75] Microsoft. Smooth Streaming Deployment Guide, March 2014. URL: http://www.iis.net/learn/media/smooth-streaming/smooth-streaming-deployment-guide.

[76] Microsoft Expression Changes, March 2014. URL: http://www.microsoft.com/expression/eng/.

[77] Move networks, April 2014. URL: http://www.movenetworks.com/history.html.

[78] Movie Body Counts Boards, March 2014. URL: http://moviebodycounts.proboards.com/.

[79] Movie Body Counts, March 2014. URL: http://www.moviebodycounts.com/.

[80] Movie Body Counts example, March 2014. URL: http://moviebodycounts.com/From_Russia_With_Love.htm.

[81] Moviesubtitles, April 2014. URL: http://www.moviesubtitles.org/.

[82] Mozilla Developer Network. DASH Adaptive Streaming for HTML5 Video, December 2013. URL: https://developer.mozilla.org/en-US/docs/DASH_Adaptive_Streaming_for_HTML_5_Video.

[83] Myspace, April 2014. URL: https://myspace.com/.

[84] MySql, April 2014. URL: http://www.mysql.com/.

[85] Neo4j, April 2014. URL: http://www.neo4j.org/.

[86] Netflix, March 2014. URL: https://www.netflix.com/?locale=nb-NO.

[87] Mozilla Developer Network. HTML Media Elements, April 2014. URL: https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement.

[88] Openstack. Stateless vs stateful, April 2014. URL: http://docs.openstack.org/high-availability-guide/content/stateless-vs-stateful.html.

[89] osgZach. ChapterDb forum, March 2014. URL: http://forum.doom9.org/archive/index.php/t-40343.html.

[90] Yael Pritch, Alex Rav-Acha, and Shmuel Peleg. Nonchronological Video Synopsis and Indexing. *IEEE transactions on pattern analysis and machine intelligence*, 30(11), November 2008. URL: http://www.vision.huji.ac.il/video-synopsis/pami08-synopsis.pdf.

[91] Randomhouse. Have Blockbuster Movies Been Good for Comic Books?, April 2014. URL: http://www.randomhouse.ca/hazlitt/feature/have-blockbuster-movies-been-good-comic-books.

[92] Simen Sægrov, Alexander Eichhorn, Jørgen Emerslund, Håkon Kvale Stensland, Carsten Griwodz, Dag Johansen, and Pål Halvorsen. BAGADUS: An Integrated System for Soccer Analysis (demo). In *Proc. of ICDSC*, October 2012. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6470164.

[93] Andy Salo. Microsoft Smooth Streaming: What You Need to Know, April 2014. URL: http://www.rgbnetworks.com/blog/?p=2680.

[94] Andy Salo. Mobile Video and Adaptive Streaming, April 2014. URL: `http://www.rgbnetworks.com/blog/?p=2484`.

[95] Andy Salo. What is HTTP Live Streaming?, April 2014. URL: `http://www.rgbnetworks.com/blog/?p=2556`.

[96] Sandy Schaefer. 'The Hobbit: An Unexpected Journey' 48 FPS 3D - Good or Bad?, February 2014. URL: `http://screenrant.com/hobbit-unexpected-journey-48-fps-discussion/`.

[97] Jeremy R. Schwartz. What is an .srt file, February 2014. URL: `http://www.ehow.com/about_5066018_srt-file.html`.

[98] Shotdetect, April 2014. URL: `http://johmathe.name/shotdetect.html`.

[99] Iraj Sodagar. White paper on MPEG-DASH Standard, December 2013. URL: `http://mpeg.chiariglione.org/standards/mpeg-dash`.

[100] Spotify, April 2014. URL: `https://www.spotify.com/no/`.

[101] Håkon Kvale Stensland, Vamsidhar Reddy Gaddam, Marius Tennøe, Espen Helgedagsrud, Mikkel Næss, Henrik Kjus Alstad, Asgeir Mortensen, Ragnar Langseth, Sigurd Ljødal, Øystein Landsverk, Carsten Griwodz, Pål Halvorsen, Magnus Stenhaug, and Dag Johansen. Bagadus: An Integrated Real-Time System for Soccer Analytics. *ACM Transactions on Multimedia Computing, Communications and Applications (TOMCCAP)*, 2014. URL: `http://home.ifi.uio.no/paalh/publications/files/tomccap2014-bagadus.pdf`.

[102] Streamingmedia. Building a DASH264 Client: Streaming Media East 2013, October 2013. URL: `http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/Building-a-DASH264-Client-Streaming-Media-East-2013--93236.aspx`.

[103] Subscene, April 2014. URL: `http://subscene.com/`.

[104] Subscene example, March 2014. URL: `http://subscene.com/subtitles/from-russia-with-love-james-bond-007`.

[105] Sigvald Sveinbjornsson. Vil revolusjonere film sok, June 2013. URL: `http://www.digi.no/917752/vil-revolusjonere-film-sok`.

[106] tagChimp, April 2014. URL: `http://www.tagchimp.com/`.

[107] Techtarget. AC3 file format, April 2014. URL: `http://whatis.techtarget.com/fileformat/AC3-Dolby-Digital-audio-files-used-on-DVDs`.

[108] Marius Tennøe, Espen Helgedagsrud, Mikkel Næss, Henrik Kjus Alstad, Håkon Kvale Stensland, Vamsidhar Reddy Gaddam, Dag Johansen, Carsten Griwodz, and Pål Halvorsen. Efficient Implementation and Processing of a Real-time Panorama Video Pipeline. In *Proc. of IEEE ISM*, December 2013. URL: `http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6746472&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel7%2F6744523%2F6746450%2F06746472.pdf%3Farnumber%3D6746472`.

[109] The Internet Engineering Task Force, April 2014. URL: `http://www.ietf.org/`.

[110] TVsubtitles, August 2013. URL: `http://www.tvsubtitles.net/`.

[111] Shingo Uchihashi, Jonathan Foote, Andreas Girgensohn, and John Boreczky. Video Manga: Generating Semantically Meaningful Video Summaries. *Proc. ACM Multimedia*, pages 383–392, 1999. URL: `http://www.cs.sfu.ca/CourseCentral/820/li/material/source/papers/videomangaacm99.pdf`.

[112] Jarret Vance. Chapter Grabber, April 2014. URL: http://jvance.com/pages/ChapterGrabber.xhtml.

[113] Videomaker. What is mov and mp4?, April 2014. URL: http://www.videomaker.com/forum/topic/what-is-mov-and-mp4.

[114] Vimeo, April 2014. URL: https://vimeo.com/.

[115] w3. Hypertext Transfer Protocol - HTTP/1.1, April 2014. URL: http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[116] Matrix Wikia. Woman in the Red Dress, April 2014. URL: http://matrix.wikia.com/wiki/Woman_in_Red.

[117] Wikipedia. Dynamic Adaptive Streaming over HTTP, November 2013. URL: http://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP.

[118] Wikipedia. Metadata, August 2013. URL: https://en.wikipedia.org/wiki/Metadata.

[119] Wikipedia. 16:9, April 2014. URL: https://en.wikipedia.org/wiki/16:9.

[120] Wikipedia. 300 (film), April 2014. URL: http://en.wikipedia.org/wiki/300_%28film%29.

[121] Wikipedia. 4:3, April 2014. URL: https://en.wikipedia.org/wiki/4:3#4:3_standard.

[122] Wikipedia. Adaptive bitrate streaming, April 2014. URL: http://en.wikipedia.org/wiki/Adaptive_bitrate_streaming.

[123] Wikipedia. Advanced Audio Coding, April 2014. URL: http://en.wikipedia.org/wiki/Advanced_Audio_Coding.

[124] Wikipedia. Advanced encryption standard, April 2014. URL: http://en.wikipedia.org/wiki/AES_encryption.

[125] Wikipedia. Advanced Systems Format, April 2014. URL: http://en.wikipedia.org/wiki/Advanced_Systems_Format.

[126] Wikipedia. Aspect ratio, April 2014. URL: http://en.wikipedia.org/wiki/Aspect_ratio_%28image%29.

[127] Wikipedia. Audio to video synchronization, April 2014. URL: http://en.wikipedia.org/wiki/Audio_to_video_synchronization.

[128] Wikipedia. Bit rate, April 2014. URL: http://en.wikipedia.org/wiki/Bit_rate.

[129] Wikipedia. Blu-ray disc, April 2014. URL: http://en.wikipedia.org/wiki/Blu-ray_Disc.

[130] Wikipedia. Bootstrap, March 2014. URL: http://en.wikipedia.org/wiki/Bootstrap_(front-end_framework).

[131] Wikipedia. Compact Cassette, April 2014. URL: http://en.wikipedia.org/wiki/Compact_Cassette.

[132] Wikipedia. Content delivery network, April 2014. URL: http://en.wikipedia.org/wiki/Content_delivery_network.

[133] Wikipedia. Digital rights management, April 2014. URL: http://en.wikipedia.org/wiki/Digital_rights_management.

[134] Wikipedia. DivX, April 2014. URL: http://en.wikipedia.org/wiki/DivX.

[135] Wikipedia. DVD, April 2014. URL: http://en.wikipedia.org/wiki/DVD.

[136] Wikipedia. Elementary stream, April 2014. URL: http://en.wikipedia.org/wiki/Elementary_stream.

[137] Wikipedia. Extensible Markup Language, April 2014. URL: http://en.wikipedia.org/wiki/Extensible_Markup_Language.

[138] Wikipedia. Film transition, April 2014. URL: http://en.wikipedia.org/wiki/Film_transition.

[139] Wikipedia. Fragmentation (computing), April 2014. URL: https://en.wikipedia.org/wiki/Fragmentation_%28computing%29.

[140] Wikipedia. Frame rate, April 2014. URL: http://en.wikipedia.org/wiki/Frame_rate.

[141] Wikipedia. Golden ratio, March 2014. URL: http://en.wikipedia.org/wiki/Golden_ratio#Painting.

[142] Wikipedia. Graph Database, March 2014. URL: http://en.wikipedia.org/wiki/Graph_database.

[143] Wikipedia. Group of pictures, April 2014. URL: http://en.wikipedia.org/wiki/Group_of_pictures.

[144] Wikipedia. Gzip, April 2014. URL: http://en.wikipedia.org/wiki/GZIP.

[145] Wikipedia. H264, April 2014. URL: http://en.wikipedia.org/wiki/H264.

[146] Wikipedia. High-definition video, April 2014. URL: https://en.wikipedia.org/wiki/High-definition_video.

[147] Wikipedia. High-Efficiency Advanced Audio Coding, April 2014. URL: http://en.wikipedia.org/wiki/HE_AAC.

[148] Wikipedia. HTML5 video, April 2014. URL: http://en.wikipedia.org/wiki/HTML5_video.

[149] Wikipedia. HTTP Secure, April 2014. URL: http://en.wikipedia.org/wiki/HTTP_Secure.

[150] Wikipedia. Hypertext Transfer Protocol, April 2014. URL: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[151] Wikipedia. ID3, April 2014. URL: http://en.wikipedia.org/wiki/Id3.

[152] Wikipedia. Initialization vector, April 2014. URL: http://en.wikipedia.org/wiki/Initialization_vector.

[153] Wikipedia. International standard, April 2014. URL: http://en.wikipedia.org/wiki/International_standard.

[154] Wikipedia. Internet information services, April 2014. URL: http://en.wikipedia.org/wiki/Internet_Information_Services.

[155] Wikipedia. Internet television, April 2014. URL: http://en.wikipedia.org/wiki/Internet_television.

[156] Wikipedia. ISO base media file format, April 2014. URL: http://en.wikipedia.org/wiki/ISO_base_media_file_format.

[157] Wikipedia. Likert scale, April 2014. URL: http://en.wikipedia.org/wiki/Likert_scale.

[158] Wikipedia. LP record, April 2014. URL: http://en.wikipedia.org/wiki/LP_record.

[159] Wikipedia. Luma (video), April 2014. URL: http://en.wikipedia.org/wiki/Luma_%28video%29.

[160] Wikipedia. Macromedia, April 2014. URL: http://en.wikipedia.org/wiki/Macromedia.

[161] Wikipedia. MIME, April 2014. URL: http://en.wikipedia.org/wiki/MIME.

[162] Wikipedia. MP3, April 2014. URL: http://en.wikipedia.org/wiki/Mp3.

[163] Wikipedia. MPEG-2, April 2014. URL: http://en.wikipedia.org/wiki/MPEG-2.

[164] Wikipedia. MPEG-4 Part 2, April 2014. URL: http://en.wikipedia.org/wiki/MPEG-4_Part_2.

[165] Wikipedia. MPEG transport stream, April 2014. URL: http://en.wikipedia.org/wiki/MPEG_transport_stream.

[166] Wikipedia. Multiplexing, April 2014. URL: http://en.wikipedia.org/wiki/Muxing.

[167] Wikipedia. Multiview video coding, April 2014. URL: http://en.wikipedia.org/wiki/Multiview_Video_Coding.

[168] Wikipedia. Nonlinear narrative, April 2014. URL: http://en.wikipedia.org/wiki/Nonlinear_narrative.

[169] Wikipedia. Object recognition, April 2014. URL: http://en.wikipedia.org/wiki/Object_recognition.

[170] Wikipedia. Offside, April 2014. URL: https://en.wikipedia.org/wiki/Offside.

[171] Wikipedia. Optical character recognition, April 2014. URL: http://en.wikipedia.org/wiki/Optical_character_recognition.

[172] Wikipedia. PAL, April 2014. URL: http://en.wikipedia.org/wiki/PAL.

[173] Wikipedia. Parsing, April 2014. URL: https://en.wikipedia.org/wiki/Parsing#Computational_methods.

[174] Wikipedia. Proof of concept, April 2014. URL: http://en.wikipedia.org/wiki/Proof_of_concept#Software_development.

[175] Wikipedia. Proprietary software, April 2014. URL: http://en.wikipedia.org/wiki/Proprietary_software.

[176] Wikipedia. Quicktime, April 2014. URL: http://en.wikipedia.org/wiki/Quicktime.

[177] Wikipedia. Real-time Transport Protocol, April 2014. URL: http://en.wikipedia.org/wiki/Real-time_Transport_Protocol.

[178] Wikipedia. RealNetworks, April 2014. URL: http://en.wikipedia.org/wiki/RealNetworks#History.

[179] Wikipedia. Regular expression, April 2014. URL: http://en.wikipedia.org/wiki/Regular_expression.

[180] Wikipedia. Representational state transfer, April 2014. URL: http://en.wikipedia.org/wiki/RESTful.

[181] Wikipedia. Responsive web design, April 2014. URL: http://en.wikipedia.org/wiki/Responsive_web_design.

[182] Wikipedia. Scalable video coding, April 2014. URL: http://en.wikipedia.org/wiki/Scalable_Video_Coding.

[183] Wikipedia. Search engine optimization, April 2014. URL: https://en.wikipedia.org/wiki/Search_engine_ranking.

[184] Wikipedia. SHA-1, April 2014. URL: http://en.wikipedia.org/wiki/SHA-1/.

[185] Wikipedia. SQL injection, April 2014. URL: http://en.wikipedia.org/wiki/SQL_injection.

[186] Wikipedia. Standard-definition television, April 2014. URL: https://en.wikipedia.org/wiki/Standard-definition_television.

[187] Wikipedia. Storyboard, April 2014. URL: http://en.wikipedia.org/wiki/Storyboard.

[188] Wikipedia. Streaming media, April 2014. URL: http://en.wikipedia.org/wiki/Streaming_media.

[189] Wikipedia. SubRip, April 2014. URL: http://en.wikipedia.org/wiki/Subrip.

[190] Wikipedia. Timecode, April 2014. URL: http://en.wikipedia.org/wiki/Timecode.

[191] Wikipedia. Timed Text, April 2014. URL: http://en.wikipedia.org/wiki/Timed_Text.

[192] Wikipedia. Timeline of audio formats, April 2014. URL: https://en.wikipedia.org/wiki/Timeline_of_audio_formats.

[193] Wikipedia. UTF-8, April 2014. URL: http://en.wikipedia.org/wiki/UTF-8.

[194] Wikipedia. VC-1, April 2014. URL: http://en.wikipedia.org/wiki/VC-1.

[195] Wikipedia. VHS, April 2014. URL: http://no.wikipedia.org/wiki/Vhs.

[196] Wikipedia. Video compression format, April 2014. URL: http://en.wikipedia.org/wiki/Video_compression_format.

[197] Wikipedia. Video compression picture types, April 2014. URL: http://en.wikipedia.org/wiki/Video_compression_picture_types.

[198] Wikipedia. Video encoding, April 2014. URL: http://en.wikipedia.org/wiki/Video_encoding#Video.

[199] Wikipedia. Video on demand, April 2014. URL: http://en.wikipedia.org/wiki/Video_on_demand.

[200] Wikipedia. Video Synopsis, March 2014. URL: http://en.wikipedia.org/wiki/Video_Synopsis.

[201] Wikipedia. Video tracking, April 2014. URL: `http://en.wikipedia.org/wiki/Video_tracking`.

[202] Wikipedia. Web scraping, April 2014. URL: `http://en.wikipedia.org/wiki/Web_scraping`.

[203] Wikipedia. Wildcards, April 2014. URL: `https://en.wikipedia.org/wiki/Wildcards#Computing`.

[204] Wikipedia. Windows Media Audio, April 2014. URL: `http://en.wikipedia.org/wiki/Windows_Media_Audio`.

[205] Wikipedia. Windows Media Player, April 2014. URL: `http://en.wikipedia.org/wiki/Windows_Media_Player#History`.

[206] Wikipedia. XAMPP, April 2014. URL: `http://en.wikipedia.org/wiki/XAMPP`.

[207] Ed Wolf. The pain of live streaming on android, February 2013. URL: `http://www.longtailvideo.com/blog/31646/the-pain-of-live-streaming-on-android/`.

[208] YouTube, March 2014. URL: `http://www.youtube.com`.

[209] YouTube statistics, March 2014. URL: `http://www.youtube.com/yt/press/en/`.

[210] Alex Zambelli. A history of media streaming and the future of connected TV, March 2013. URL: `http://www.theguardian.com/media-network/media-network-blog/2013/mar/01/history-streaming-future-connected-tv`.

[211] Alex Zambelli. IIS Smooth Streaming Technical Overview, December 2013. URL: `http://www.microsoft.com/en-us/download/details.aspx?id=17678`.

[212] Alex Zambelli. Streaming the Olympics: How we got there, April 2014. URL: `http://blogs.iis.net/alexzam/archive/2010/02/16/streaming-the-olympics-how-we-got-here.aspx`.

[213] Brightcove Zencoder. Concatenation hls to the rescue, June 2013. URL: `http://blog.zencoder.com/2013/01/18/concatenation-hls-to-the-rescue/`.

[214] ZXY Sport Tracking, April 2014. URL: `http://www.zxy.no/`.