University of Oslo
Department of Informatics

Caching of Interactive Branching Video in MPEG-4

Simen Rekkedal
simenre@ifi.uio.no

Thesis for the Candidatus Scientiarum degree.

July 12, 2004

Abstract

The goal is to investigate the minimum amount of knowledge needed by a proxy server for consistent caching of interactive multimedia scenes encoded in MPEG-4 systems.

We limit the interactive multimedia scenes to be of the type *branching video*. The caching scheme proposed is a specialization of *partial caching*. The *extent* of a branching video is the number of alternative *branches* available to the user at a *branching point*.

The proposed caching scheme is *extent domain caching*. It works by limiting the number of alternative branches stored in the cache. Cache misses are served from the source server to provide a service transparent for the users. An implementation with test runs is provided. In the implementation the interactivity is limited to building blocks of complete ES.

For content with several alternative scenes within a single ES, a different approach is needed. The proxy have to construct ad hoc objects from the AUs that constitute the alternative scenes. These ad hoc objects should be identifiable and have defined boundaries. Caching replacement decisions are then made on the ad hoc objects, and not on the complete ES. Identities can be constructed from the sequence number of the AUs and the ES id. The boundaries can be found by analysing where the users shift playback point.

Preface

This document is a thesis in partial fulfilment of my Candidatus Scientiarum degree at the University of Oslo, Department of Informatics.

First let me thank my fiancee Irene Gjøsund for putting up with me through all of this.

I would also like to thank my tutors Prof. Dr. Carsten Griwodz and PhD. Student Frank Trethan Johnsen for guiding me through this task.

Many of the ideas about narrative and interactive navigation through hypermedia comes from fellow student Odd Joachim Carlsen, big thanks !

University of Oslo, Department of Informatics
10.7.2004
Simen Rekkedal

To ease navigation of the electronic document the pagenumbering is absolute, rather than starting at 1 again on page 11.

Contents

Chapter 1

Introduction

This thesis investigates caching of interactive multimedia. Interactive multimedia content can be encoded using MPEG-4. We discuss how MPEG-4 provides descriptors that can make consistent caching possible.

Structure

This chapter briefly describes the conceptual framework for this thesis. The full definitions of these concepts can be found in chapter 2. Chapters 4, analysis, and 5, design, build upon these concepts. An overview of MPEG-4 is given in chapter 3.

Chapter 6 presents an implementation of caching of branching video. Chapter 7 discusses the results from the implementation in context of the design, and presents possible future work.

1.1 Definitions

This section defines the research field for this thesis. A brief presentation of the needed definitions is provided here. More detailed definitions can be found in chapter 2. The topics of the related work is divided into two parts; central topics and related topics. This is done to emphasize the central research field.

1.1.1 Central Topics

The two fields that we have combined to provide a new research field is streaming branching videos and caching, figure 1.2. The field of streaming is not a central topic of its own, rather we focus on the knowledge needed for caching a streaming branching video. The analysis is presented in chapter 4.

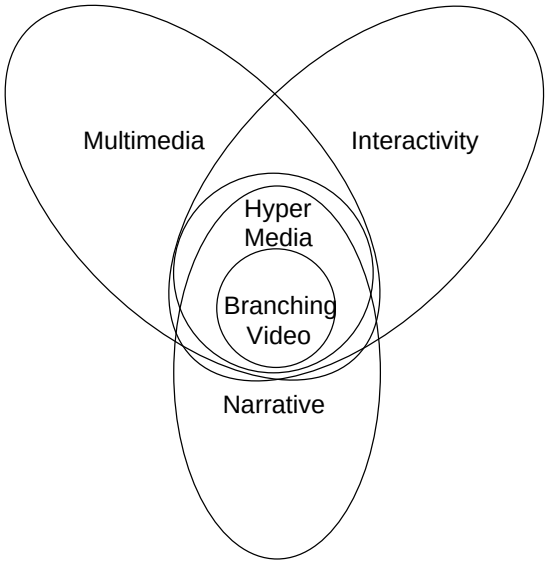

Figure 1.1: Branching Video

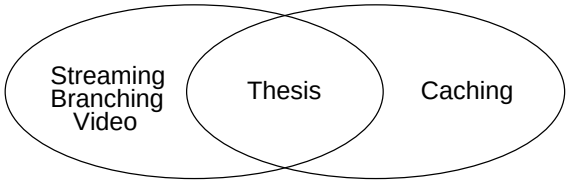

Figure 1.2: Central Topics

Caching

This thesis contributes to the field of caching, defined in 2.1. Caching as a subject is a vast research topic of its own. In this thesis' scope caching is limited to the partial caching of interactive multimedia content in proxies. This thesis proposes a third domain of partial caching, in addition to the well defined quality and time domains, [59]. *Extent domain caching*, presented in chapters 4 and 5.

Branching Video

Branching Video (BV) is a subset of hypervideo, which in turn is a subset of hypermedia, see figure 1.1. These concepts are fully defined in section 2.3. It is assumed that the BV is encoded using MPEG-4 systems descriptors. MPEG-4 is defined in chapter 3.

1.1.2 Related Topics

In order to provide a solid foundation for the thesis, a few other topics need to be defined. These topics are:

Streaming

Delivery of multimedia content over a network in a time-dependent manner, for playback by a user before the complete set of data has been transmitted. Streaming is defined in 2.2.2.

Content Distribution Networks

Multimedia content must be distributed to consumers in some way. Traditionally it has been on removable media such as tapes, disks or cds. In recent years, an increasing amount of multimedia content has become available on the Internet. Providers of multimedia have a vested interest in controlling the distribution of their content. Consumers want to have easy access. A Content Distribution Network (CDN) provides content over a network. CDNs may be accessible through the Internet. In this thesis we limit CDNs to Internet or WAN based nets, even though the argument could be streched to other kind of networks, e.g TV cable networks. CDN is defined in 2.2.3.

User's Perception

Multimedia content is intended for a human audience, and not for consumption by a computer. This is why we can use lossy compression. Still, a human will not wait for any length of time for a product, or accept

any level of quality. A foundation for human usability of experiential systems is given in 2.2.5.

A video's popularity with the users changes throughout its lifetime. Consequently the content stored in the proxy cache should change accordingly. More about this in [17].

1.2 Method

Building on the framework provided by chapters 2 and 3 we analyse how MPEG-4 descriptors can be used to consistently cache interactive multimedia. The analysis is presented in chapter 4. Chapter 5 presents a design. To test our design we present a partial implementation in chapter 6 and discuss the results in chapter 7. In the empirical method, test data is compared with empirical data. We don't have any empirical data, since no comparable implementation exists in real world deployment. This is why the final part, where the test data is compared with empirical data, is left for future work.

Chapter 2

Related Work

Presenting the related work needed for the analysis and subsequent design. Particular emphasis should be placed on sections 2.1.7, 2.3.3 and 2.3.5.

2.1 Caching

[63] define caching or replication as the storage of content on other servers than the source server. Caching is when the algorithms run on the extra servers. In replication, the process is controlled from the source server.

2.1.1 Proxy Cache Servers

A proxy server is an extra server between a source server and an end user, [51]. The word extra refers to that the proxy is not needed just for transport. [39] expand upon existing web caching proxies to that of caching videos, they present initial and selective caching. [53] presents a quality adaptive approach, which reduces file sizes by reducing quality. [33] uses layered encoding to adapt to heterogeneous Internet access. [44] explore the convergence of caching and streaming with an RTSP based proxy. [57] uses quality adjustments considering user demands and available bandwidth. [70] address optimal allocation of the proxy prefix cache to save bandwidth, also considering transmission scheme. [16] uses prefix caching and periodic broadcast of the remaining suffix. [11] caches thumbnails to positions in a video for interactive access. [5] has designed and implemented a streaming media proxy cache for the Internet. [3] considers byte hit ratio and the granularity of replaced objects. [2] describes a collection of cooperating proxy servers in a local area network. [59] uses MPEG-4 to adapt the content in the proxy server, it also defines the time and quality domains of partial caching that this thesis expands upon. [34] proposes a time-sensitive adaptive approach to reconcile the best effort service model of the Internet with the timeliness demands of streaming video. [21] proposes

extensions to RTSP to aid in prefix caching. [78] presents a technique of retransmitting missing segments, called fair share claiming. [54] presents a fine grained replacement algorithm for layered media, and describe a pre-fetching scheme to smooth out variations. [74] present and evaluate dividing the videos into variably sized segments, the initial segments are given preferential treatment.

In this thesis a proxy will always be a proxy caching server, illustration in figure 2.1. An end user requests content from the proxy, rather than from the source server. If the proxy can't serve the request, it is called a cache miss. Cache misses are either served from the source server, or the proxy can retrieve the content on behalf of the end user. For a proxy to update its cache, retrieval of missing content must happen at some point.

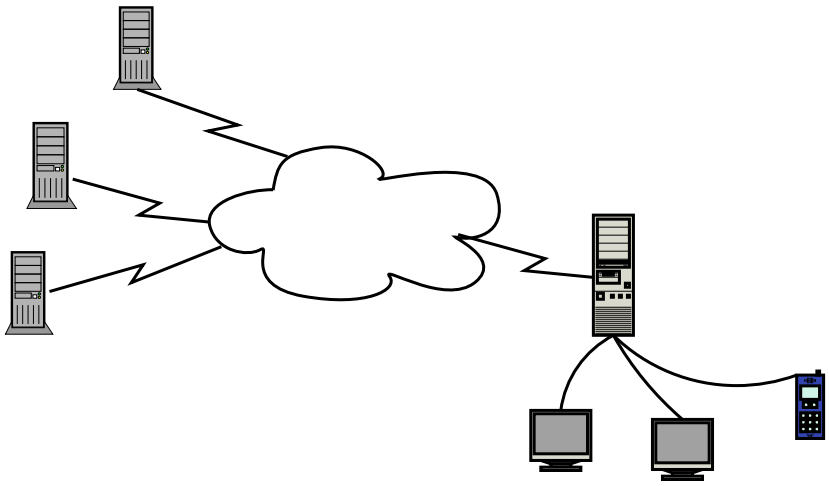

Figure 2.1: Proxy Caching Server

2.1.2 Proxies in Content Distribution Networks

Content Distribution Networks (CDN) are described in more detail in 2.2.3. A CDN can have servers that distribute the content purely as means of transport. But when these servers also may store content for reuse, they become proxies or replication servers. The discussion in the sections that follow assumes that we have a CDN with at least one proxy.

2.1.3 Architectures

By architecture we refer to the amount of proxies used in the network, and their inter communication. [71] describes the ideal caching scheme for web content, and pointed out inadaquacies in proposed systems. Since this thesis considers streaming of pre-fabricated stored content, the problem

of stale data is not central, [59]. The main types of architectures are summarized below:

Autonomous

In an autonomous caching scheme the proxy considers only its own data, see figure 2.2. There might be only one proxy, or there might be several that just don't communicate. This scheme is simple and easy to deploy. A disadvantage is that a single proxy means a single point of failure. If the proxy has more requests than it can serve, it will be a bottleneck, especially if all requests have to pass through it. [18, 56, 71].

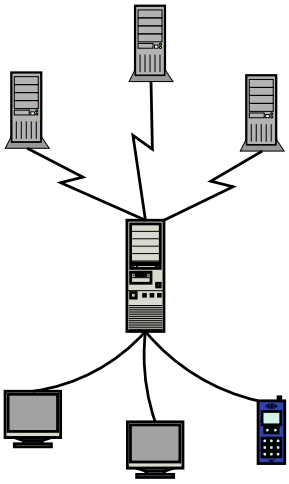

Figure 2.2: Autonomous

Hierarchical

A hierarchical system uses a treelike network of proxies, see figure 2.3. The proxies may have different tasks depending on their position in the hierarchy. A hierarchical system is more bandwidth efficient . There are no single points of failure. Proxies might use algorithms that take the other proxies into consideration. Such a system is more difficult to deploy. Proxies must use resources to relate to each other. Proxies near the source servers may become bottlenecks. Each level of proxies between the users and source servers might introduce delays. [18, 56, 71].

Cooperative

Cooperative caching schemes seek to increase the global hit ratio. This is done by sharing the cache. In a flat distributed caching scheme, there

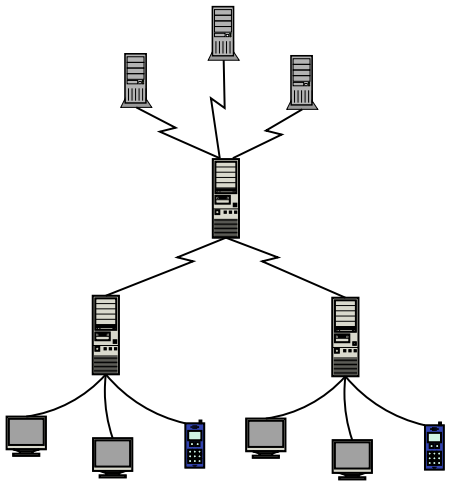

Figure 2.3: Hierarchical

is only one level of proxies, and they all serve each others' misses, see figure 2.4. In a hybrid scheme there might be more levels, see figure 2.5. Proxies keep meta information about the content stored in proxies in the same level. If a level does not hold the document, the request is issued up into the next level, or to the source server. Such systems are more fault tolerant, and have better load sharing. Still higher bandwidth usage might occur. Content may be retrieved from a slow peer proxy which will increase delay. If the system crosses administrative domains in the Internet, this can introduce new problems. [18, 56, 71].

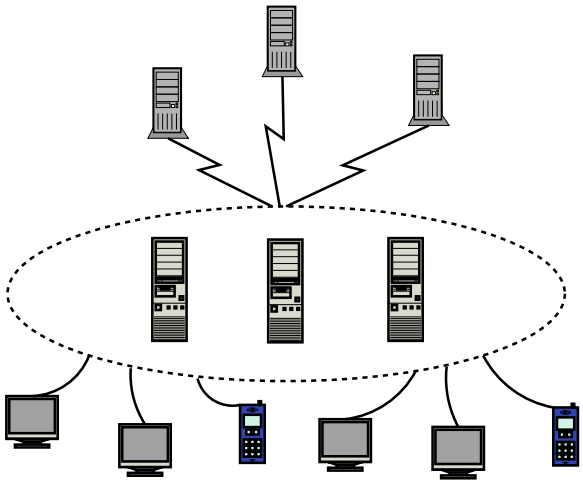

Figure 2.4: Flat cooperative

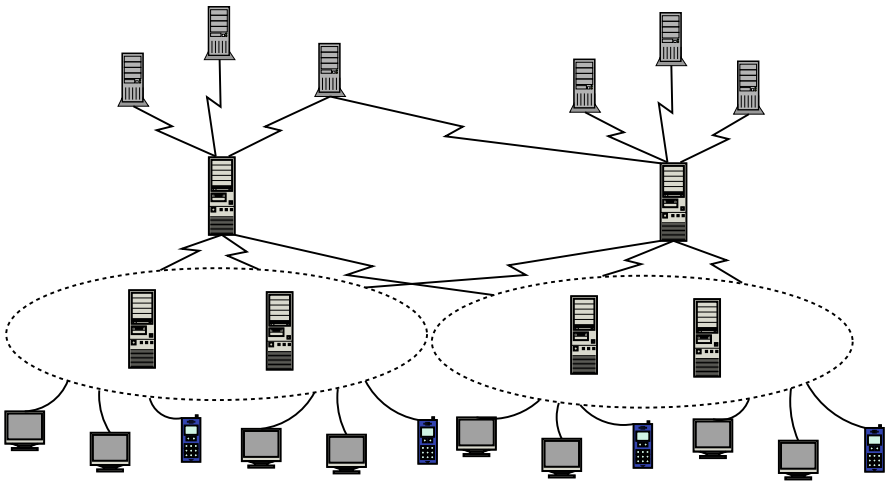

Figure 2.5: Hybrid cooperative

Relevance for thesis

In this thesis we will choose the autonomous caching scheme. It is sufficient for the purpose of illustrating communication between a group of users and source servers, with an intermittent proxy. The added complexity of hierarchical or cooperative schemes, though interesting is not within the scope of the thesis. The proposed caching system in chapter 5 cache objects, in the same way as any other cache. It should be possible to extend the system into hierarchical or cooperative schemes in some way, though this is left for future work.

2.1.4 Replacement Strategies

In addition to the caching algorithms itself, which are outlined below, there are a few other strategies that can be used in a cache. Reversely administrators might want to focus on cache hit ratio to boost the performance of all requests, then it is often better to cache smaller objects. If a high byte hit ratio is wanted, partial caching can be employed, discussed further in 2.1.7. If the service is tailored to a specific group of users, it is also possible to only cache a specific type of content. Another technique that might be useful if the system has a period of time with significantly lower traffic, is prefetching, or replication. In prefetching the proxies ask source servers for interesting content, according to some policy. In replication, the source servers push specific content onto the proxies, [49, 51, 71].

Relevance for thesis

The design and implementation will have a specific strategy that are tailored to emphasize the test of consistent caching and the minimum amount of knowledge needed. This thesis will not have analyse all such strategies in detail. We will make a suggestion for a new strategy in future work 7.2.

2.1.5 Caching Algorithms

The content of the cache should be the most requested objects, this will increase the likelihood of reuse. In order to keep the most popular objects in cache, several types of algorithms have been developed. Some of these purely statistical algorithms are outlined below:

Least Recently Used

The reference algorithm for research in caching efficiency is the Least Recently Used (LRU) algorithm, [49, 51, 71]. It displaces the objects that have the oldest access time. The access times are updated when the objects are accessed. Only one access time is stored for every object. It is very simple, and for many purposes is efficient enough.

Least Frequently Used

The second most common algorithm is the Least Frequently Used (LFU), which displaces the object with the fewest accesses, [49, 51, 71]. A problem with LFU is that objects hold on to their popularity indefinitely. This means that objects that no users request any more might still be in the cache, because their hit rating is higher than the younger objects. This can be circumvented by limiting the hit rating to a set time frame, or by slowly aging the value.

Key Property Algorithms

Many algorithms employ a test for additional properties, such as size, content type, position in the source material, relation to other objects, etc, [49, 51, 71]. Key property algorithm displace candidates according to a specific property such as size or latency. They are often refined versions of LRU or LFU. And often has primary, secondary and tertiary keys to solve ties.

Function Algorithms

Function Based algorithms take more variables into acount, [49, 51, 71]. Common for all these algorithms is increased use of CPUs, whilst it is not guaranteed to increase performance of the cache. The algorithm must be designed to complement the business decisions made by the owner or administrators of the network. No known algorithm is perfect for all situations.

Relevance for thesis

We will use both LRU and LFU in the sets of test runs on our implementation. The implemenatation is presented in chapter 6 and the results are in the same chapter. Discussion about future work and the design of a key property or cost based algorithm is in chapter 7. Statistical algorithms are sufficient for the purpose of demonstrating successfull cache replacements. This thesis is more interested in what is needed of meta information about the cache candidates than optimizing algorithms.

2.1.6 Binary Caching

In binary caching policies complete movies or objects are kept in the cache, figure 2.6 displays the replacement of a film in the cache, [49, 59]. The algorithms will then replace the movie in its entirety if selected. Usually implementations use some form of partial caching, and stream the remaining part from the server. Films are simply to big to cache, they can be replicated though , but that is a different technique, and controlled from the source server.

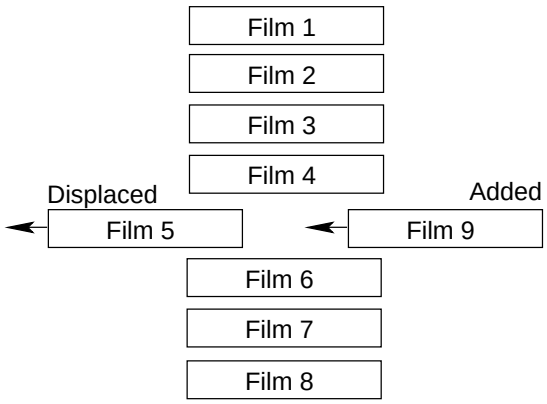

Figure 2.6: Binary caching

Relevance for thesis

This thesis will not use binary caching, since it may be impractical to cache complete movies with redundant narrative information, more on this in 2.3 and chapter 4. If it is expected that the users will always use all the available information, binary caching is of course advantageous. If this is the case, then only a very small set of movies may be cached. The worst case scenario for partial caching should mimic that of binary caching. The proposed system does this as explained in chapter 5.

2.1.7 Partial Caching

Partial caching can be especially useful if the content consists of large objects that doesn't change to much over time. This is true for streaming video in a CDN that delivers stored content. Streaming video from teleconferences or webcams are not suited for caching, since there is no reuse of the streams. In News on Demand 2.2.3 the popularity of the objects might change quicker than in Video on Demand 2.2.3 or Learning on Demand 2.2.3. Additionally users often start a playback and then stop the transmission before the video is finished. If the initial segment of a presentation is available with low start up time, the following segments can be prefetched, patched or batched later. The following illustrates the two different ways of scaling a cache object for partial caching.

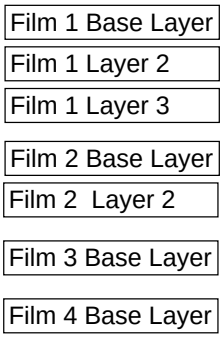

Figure 2.7: Quality domain caching

Quality Domain Caching

Quality domain caching needs content that is either encoded in layers or split into several interdependent objectsa, as in figure 2.7. MPEG-4 is an example of this, see chapter 3. Rescaling content within the proxy is very CPU intensive if the content is not prepared for scaling. Popular content should be kept in the cache with a higher level of quality than less popular content. Additionally the quality of content often dictate the

size of the bandwidth and playback resources needed by the end users' terminals. Quality caching can be used as a portal for terminals with limited capabilities, [59]. Quality caching is explored in [33,44,53,57].

Time Domain Caching

Reducing the files in size by cutting of the timeline of the multmedia content is effective to increase the number of files that can be stored. Even if the end users are intent on watching whole movies, some quit after a little while. The opening parts of the movie can be served to the clients in a speedily fashion, which may be enough on its own. Other techniques can be used to catch up with the streaming. Such as Batching, Prefix caching, Chen & Tobagi Solutions. Time domain caching has been investigated by: [2,3,5,11,16,39,70,74,77] described earlier. [60] uses proxy prefix caching. Figure 2.8 shows a scheme that has a floating size prefix according to the popularity of the film. Figure 2.9 shows the film partioned into several segments, and a popular film is cached with more segments than a less popular.

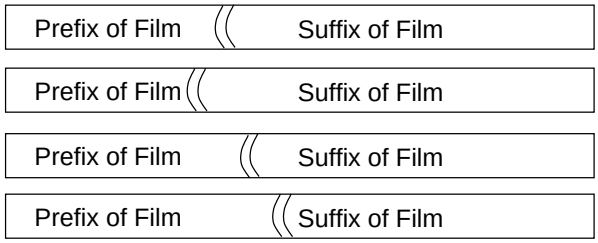

Figure 2.8: Time domain caching I

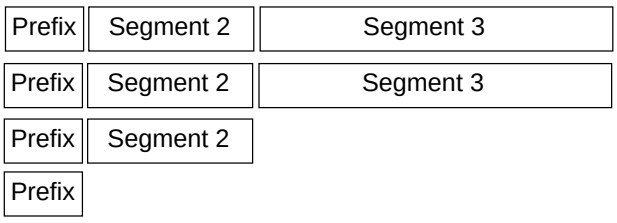

Figure 2.9: Time domain caching II

Relevance for thesis

This thesis builds on partial caching and proposes a third domain, the interactivity extent domain, more on this in 4.2.3.

QBIX in details

The work presented in [59] is similar to the work presented in this thesis, caching of stored content encoded in MPEG-4. In [59] however they limit the adaptive replacement policy to partial caching in the quality domain. They mention partial caching in the time domain, but write that off, as less central to their work. Partial caching in the extent domain were not mentioned at all, which is what we will propose in chapter 5.

The intention was that the proxy in [59] could be used simultaneously as a proxy cache server and as an adaptive portal for terminals with limited playback capabilities and network resources. They talk about doing this in one of two ways; either with partial caching of alternative layers for each media object, or with system level adaptation. They did not implement system level adaptation in [59]. The extent caching we propose will need elements from MPEG-4 systems, in order to recognize the caching candidates.

2.1.8 Transmission Policies

The transmissions between server and proxy, and between proxy and users are called transmission policies. Though not central to this thesis, they are important in that several efficient ways to keep the bandwidth usage low can be achieved. Such as periodic broadcast, [16].

2.1.9 Admission Policies

Whether to cache the object on the first appearance or to wait until subsequent requests are made, can aid in bringing the steady state quicker.

2.1.10 Concurrent Thrashing

If concurrent users request content that is to great to fit in the steady part of the cache, the content can loose its position in the cache before the users' playback has finished. This is called thrashing, and should of course be avoided. Obejcts that are currently being played, should be given preferential treatment.

2.2 Multimedia and Internet

The term multimedia means multiple types of content bundled together to make one whole presentation. The usual minimum being one video stream and one audio stream. Additionally the term media can refer to text or images. Recent years have seen an explosive growth in both media and

multimedia on the Internet. The following subsections defines the needed background theory.

2.2.1 Quality of Service

This definition of Quality of Service (QoS) is paraphrased from [9]: Once users are provided with the functionality that they require of a service, we can go on to talk about the quality of the service provided. The main non functional properties of systems that affect the quality of the service experienced by the clients and users are reliability, security and performance. Adaptability to changing system configurations and resource availability has recently been recognized as a further important aspect of service quality.

The abbreviation QoS has effectively been used to refer to the ability of systems to meet the deadlines of time critical data, such as multimedia streams. The packet loss ratio can affect the quality of the stream as frames are interdependent in for instance MPEG-2. QoS is a requirement for the system to provide guaranteed computing and networking resources at the appropriate times, and in a sufficient amount to complete each task in time.

Each critical resource must be reserved by resource managers along the way. If the required reservation cannot be met it is rejected. The Internet today does usually not provide QoS explicitly.

For multimedia streams the permissable limits of jitter, throughput, delay and errors are the major elements in the QoS.

In this field the term QoS simply refers to how many packets the system can deliver to the end user in time for playback of the content. Without compromising the quality of the playback to much, with regards to motion, color, sound, skipping frames and other effects the user perceive. Effects are jitter, start up delay, frame loss, loss of image or sound quality due to scalable transmisson, lagging or complete loss of service.

Relevance for thesis

This thesis contributes indirectly to the QoS of multimedia delivery over Internet. We introduce a system that also enables the caching of interactive content, in chapter 5. This system should decrease latency and thereby improve the QoS. However, the details of QoS are not a central topic in this thesis. The work in this thesis builds upon the field of caching, 2.1, the relation to QoS is well defined in that field already.

2.2.2 Streaming Multimedia

Research about streaming is vast and is investigated in these articles [18, 20, 28, 34, 46, 54, 59] all described previously. [26, 52] describe RTP and

RTSP which are protocols for streaming over the Internet. Video or audio are the types of media that are prepared for streaming. Still images and text are easily downloaded. However, in a presentation using multiple still images and changing text, it could be possible to create a streaming presentation of such content as well. The MPEG-4 Systems has support for this, refer chapter 3. In streaming timeliness, bandwidth usage and the quality of the user experience are key issues. The usual scenario is a source server with content prepared for streaming. A set of users that have links to the streaming content, via for instance RTP/RTSP, [26, 52]. The end users then either request the content actively from the source server, or alternatively tap into a predetermined broadcast of the content. This can be done using unicast, multicast or broadcast routing algorithms, [46].

Relevance for thesis

The various factors involved in streaming multimedia over the Internet is not a central topic in this thesis. Rather the thesis contributes to this topic indirectly by proposing a new type of partial caching, see chapter 5, as mentioned above.

Multimedia Server

The source server that provides the multimedia content has a set of parameters that is slightly different from other web servers. Multimedia files are usually very large and don't change as often as other web content. End users perusing classical types of web content, such as html webpages with a few images, download relatively small files, and then spend a little time watching the content. Multimedia is data heavy per second of playback. The multimedia servers need to access the very large multimedia files almost continuously to serve the end users. The way a multimedia server accesses disks and memory becomes an important issue, [22].

Relevance for thesis

The multimedia servers may be improved greatly, but this is nontrivial and lies beyond the scope of this thesis. Still, the proxy is itself a multimedia server, and this topic is relevant for future work.

2.2.3 Content Distribution Networks

Theoretical presentations about CDNs are available in [2, 17, 54]. [56] provide an analysis of Internet content delivery systems. [18] presents an overview of work done to support large scale VoD systems. [59], puts the proxy in an end-to-end adaptive video delivery system. ISMA, [28], work

with the adoption and deployment of open standards for streaming rich media content over Internet protocols. RFC3016, [20], is the RTP Payload Format for MPEG-4 Audio/Visual Streams, and needed for streaming. iTunes, [31], is a well known service that distributes media, though not using streaming. RN, [42], is another service that distribute media, but using streaming. The field of multimedia streaming on the Internet is invariably one of content distribution as well. Content distribution may mean many things in addition to our subset called streaming. This section tries to put the subject into a greater context. Distribution of multimedia may be done in several ways. The traditional way has been to print removable media, such as diskettes and CDs. With the onset of Internets popularity delivery of multimedia through alternative means have increased. The various such networks are denoted as Content Distribution Networks [18]. A brief discourse of these follow.

Video on Demand

Video on Demand (VoD) systems are still under development, since many of their intrinsic properties have yet to be implemented in a suitably efficient manner, [18]. The large filesizes of the videos and the limited bandwidth in the delivery networks are the main problems. This has led to limited versions of VoD such as Near VoD to be developed. And also systems with a limited amount of different videos, which greatly improves the benefit of caching. At present very few of these systems run commercially on anything else than a classic TV broadcast network. True VoD, defined as delivering any video to any subscribing customer at any time without limiting quality has not been implemented yet. But recently limited services have emerged, such as NextGenTel, [75].

News on Demand

News on demand (NoD) is similar to VoD that the objects are multimedia clips that are streamed to the end users terminals upon request, [41]. However, the clips are smaller and subject to change in a much shorter time span. This will have an effect on caching efficency and caching policies.

Learning on Demand

Learning on demand (LoD) is also similar to VoD, but the multimedia objects might be more numerous and smaller than for a VoD session, [36]. LoD is the field of these that have the most to gain from the results of this thesis, since interactivity may be on an internal level. That is, users might change the content they want to request several times for each presentation, and not just complete presentations, as in VoD or NoD.

Relevance for thesis

This thesis will not make specific contributions to the fields of NoD or LoD directly. Though they are likely to benefit from the work done to enhance caching of interactive multimedia scenes in VoD systems.

2.2.4 Services

In addition to streaming video, some other types of services could benefit from caching. If these services use media streaming or multimedia delivery in a similar manner, and are likely to be improved with interactive capabilities, then they can benefit from the work presented in this thesis. Interactivity is defined in 2.3.2. Also, these are provided here as a broader background:

Digital TV

News broadcasts and entertainment shows now exhibit multiple features that would prove impossible without computer assistance. However the viewer remains passive except in a few TV shows that utilize the SMS service of mobile phones. Additionally the TV broadcast networks are not compressed with as great a ratio as digital content. For these reasons many commercial companies are presently in the process of migrating their service to digital platforms [45]. Naturally customers will have to buy new TV sets, but in a transition period over ten years or so, this will not be a problem. With TV going digital new types of services are likely to emerge, many which will be simply copied from the Internet of today, such as chat or bulletin news boards [38, 72].

Digital Radio

Digital radio is standardised as DAB [73] and is presently implemented in Norway by NRK [43]. The same arguments as for TV apply, albeit the lack of image naturally limits the number of services that may be copied of the Internet. When DAB was released one of its capabilities presented was that if a customer heard a music song on the radio, the customer could buy that song at the press of a button, and receive the CD in the mail, normal mail that is. This service is obviously extensible to one akin to Apple's iTunes, the technical capability is here, only problems concerning content copyrights and piracy are holding back.

Handheld Devices

Many of the new interactive services in popular media are driven by the SMS capabilities of the mobile phones. With new and more powerful

handsets these services are also likely to mimic those already on the Internet, and may, combined with digital TV prove to be more attractive and easier to use for non computer professionals than the Internet. At present there is a delineation between Personal Digital Assistant (PDA) and mobile phone, although they are rapidly converging. The proliferation of other small digital equipment, known as wearables will also contribute to new and complex services. Although many of these devices have limited multimedia capabilities, they are rapidly gaining them, and additionally may increase the potential of consumer interaction with services on the web or TV.

Single State Games

Games with a single state for every user is not much different from the browsing done in a normal website or navigation in an interactive movie, caching might increase efficiency. The player in a single state game has no other players or nodes to be influenced by. For this reason subsequent players playing the same game might choose the same path through the game experience, at the very least choose to visit the same popular places that is central to the narrative of the game. In fact playing a single state game is not much different from navigating through a LoD or other type of application, such as NoD or BV, that releases information at the users interactive request. These various types includes help functions, school programs, interactive documentaries and such.

Shared State Games

In shared state games the players at various end nodes all share an experience in the same virtual world, the information regarding one player may and often do directly impact the other players experience. For this reason there is no way such precise information might ever be needed again in a cache scheme, at least not quickly enough to warrant having a caching scheme at all considering the overhead that caching introduces.

If several players share the game at any given time, it is necessary for them to also share the state of the game. This means that graphics and sound will be influenced by what the others do, and advanced graphics engines are needed to render the result. In effect each situation is unique, and that leaves us with nothing to cache for later reuse. Except possibly generic primitives that define how the rendering should be done, such as openGL [4] and vrml [30].

Since the protype information for rendering complex objects might be the same, even if the end result after rendering is unique. It may be beneficial to cache such prototypes, but this will need a sophisticated algorithm. See chapter 7 for more on this topic.

Relevance for thesis

Games are not a central topic as such, it is the interactive nature of them that lends itself well to illustrate the functionality needed for a proxy cache of interactive multimedia content. Still the design and implementation will use the example of an interactive feature film. A single state game however might not be much different from such a feature film. And the same logic should apply.

2.2.5 User's Perception

The end user is a human and will judge the presentation by its aesthetics. This is a highly non cumputational notion and before we proceed it is helpful to have a foundation for human usability of an experiential system, [10, 14].

Relevance for thesis

The thesis will assume that the users behave in a certain way to get the results needed. We assume that long tailed Zipf distribution is a valid representation of users' request, [17] explains that this may not always be the case. And we assume that the quality compressed content, the QoS and all such considerations that may affect the users perception of the presented content would not affect the results in adverse ways. This is of course far from the real world. Still we believe that it is interesting to test the strategy in a test program rather than planning to implement it in full scale.

2.3 Interactive Multimedia

A number of proprietary formats exists, but all of these require a plugin in the HTML browser and are limiting in other ways as well. A seamless mix of application interfaces, HTML and multimedia in an open and efficient standard is yet to be implemented. MPEG-4 might offer a solution, but again, only if all presently used HTML browsers have a full ISO / IEC 14496 standard implementation plugin. MPEG-4 is presented in full in chapter 3.

The display of pictures and text in a formatted way are successfully defined in HTML. However, authors of most websites wish to utilize more powerful graphics or multimedia such as; HyTime, [29] which is a language to describe time based insertion of media in hypermedia documents. SMIL, [69],integrates independent multimedia objects into synchronized presentations. SVG, [68] is a language to describe graphics in XML, [65]. Flash, [12] is a language that builds complete interactive graphical presentations. Javascript, [27] bring the flexibility of a programming language to the web page.

Standards and Plugins

The full standard plugin that the above section warrants has not been made yet. The closest is the Envivo plugin, which unfortunately only works with Quicktime, Windows Media Player (WMP) [40] and Real Networks, which are far from being HTML browsers. This means that a consumer would get the content in a small player window outside the website, which is not the perfect case. The Quicktime and WMP plugins may start a small frame embedded within the browser, however their use is still not as flexible as that of embedding images into webpages. Moreover the two way communication that the interactive services needs are not supported properly by the streaming plugins. Albeit many players claim to be able to play MPEG-4 compliant videos, only a few applications recognize the Binary Format for Scenes (BIFS) and Object Description (OD) frameworks, more on these in chapter 3. Additionally the players that do recognize BIFS and OD doesn't necessarily interoperate. The ISMA [28] group works to enable interoperability with regard to the streaming of MPEG-4, and has released the RFC 3016 [20], which specify a one to one mapping of MPEG-4 compliant video and audio to RTP packets without the use of ISO / IEC 14496-1 Systems. Work with streaming MPEG-4 over IP is presently being conducted by MPEG itself.

Relevance for thesis

Interactive multimedia must be represented in some media content format, and we believe that the logic that applies to MPEG-4 like content in this thesis should apply in the same manner to content in other formats. The thesis will not consider other such formats and their similarity or any other characteristics further.

2.3.1 Narrative

We start with [6] to define narrative, of more recent issue the work of [10] provides us with a foundation. In short, the narration is linear in presentation, though the events portrayed may not be presented in the same order as they occurred within the world of the narrative, illustration in figure 2.10. This separation of storyline and plotline is a tool that authors use for dramatic effect. The symbols of a language may be put together according to a set of rules to provide new content. Like the letters in an alphabet. The scenes in a narration can be viewed as such primitives or symbols, put together by the author to provide a drama. In [15] this principle is used. A user might choose to peruse the narrative in a nonlinear way, in a book it is easy to skip chapters, or jump back to reread a previous section. More on nonlinear reading of narrative in 2.3.2.

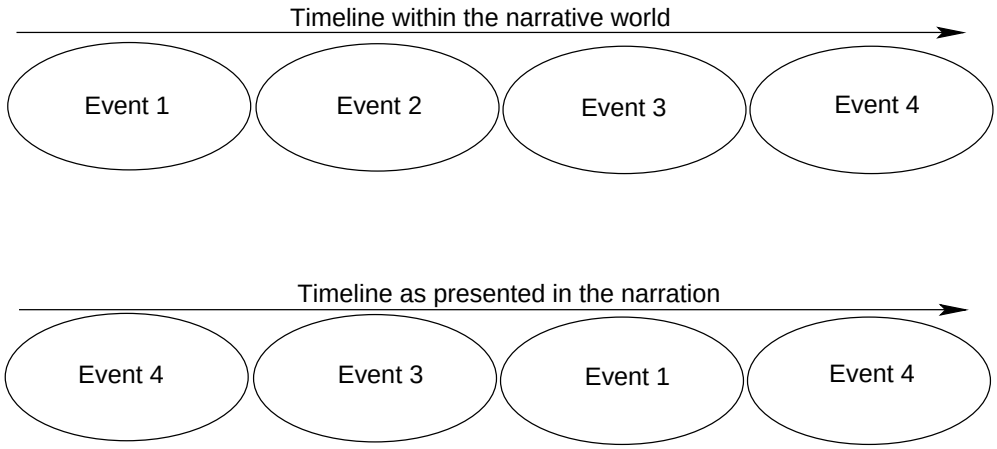

Figure 2.10: Events in narrative

Narrative in Artificial Intelligence

In the field of Artificial Intelligence the word narrative has a slightly different interpretation, [47, 76]. From [47]: Narratives is a possibly incomplete specification of actions or events that are known to occur at specific time points. In this field they talk about narratives in the context of a situation calculus, that is beyond the scope of this thesis. Still the branching structure of the emergent possibilities have a semblance to the structures that arise in interactive multimedia, as the following sections will discuss. Advanced computer games have benefited from research done in this field, and it is fully possible that the other fields listed in 2.3.6 can benefit from this as well.

Relevance for thesis

The work presented in situation calculus and other forms of emergent storytelling will not influence this thesis much. Rather we focus on premade stored content that can be reused as building blocks in some presentation. The inherent logic between the building blocks are not considered at all. Beyond of course the simple logic of containment in the interior of an aggregated object.

2.3.2 Interactivity

The term interactivity needs a specific definition in this context, and one is presented in [32]. Though Manovich say it is a tautology to talk about interactivity with media, since a user can always inter act with any media [35]. In the context of streaming media over the Internet, the user has a set of choices that are distinctly defined by the application. A radio or TV

broadcast for instance can only be swithced off, which in turn will lead to lowered ratings for the network and then the content change some time in the future, [32]. It is the speed, frequency and granularity of the interaction that is interesting. The figure 2.11 shows how content is constructed by symbols acording to rules given both implicitly by the media itself and by the author and selection of the user, [1,35].

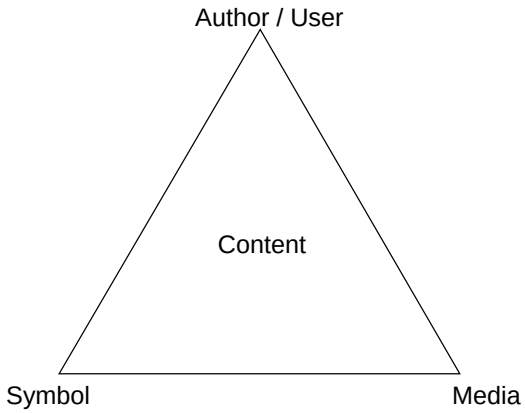

Figure 2.11: Narrative Content

Low Level of Interactivity

When a user stops, restarts or otherwise repositions the playback point of a presentation [37], we will call this low level interactivity. The figure 2.12 shows shifting of playback point.

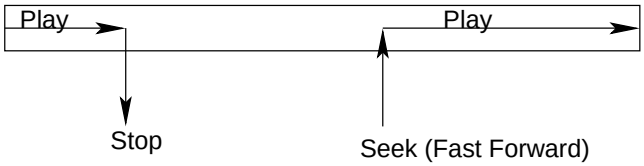

Figure 2.12: Low level of interactivity

Medium Level of Interactivity

Taking user interaction one step further, hypermedia allows a user to follow predefined links in the content that exchange the currently presented scene, figure 2.13 show this for a document of small film cuts. This is implemented very successfully in the Internet itself, with hypertext, in narrative hypertext [14] and in hypervideo [58]. Figure 2.14 shows how

medium level interactivity works for news on demand, each film cut would be a single news piece.

The common denominator for these medium level interactivity schemes, is that a set of prefabricated media objects, are put together into a dramatic presentation following both the rules of an author and the choices of the user. This is further investigated in [7, 8, 14, 48, 62, 64, 76].

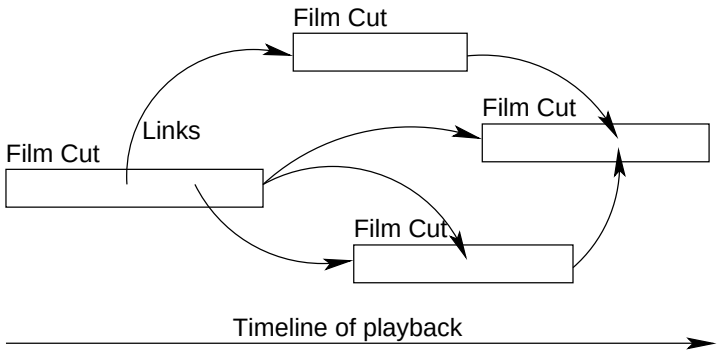

Figure 2.13: Hypermedia

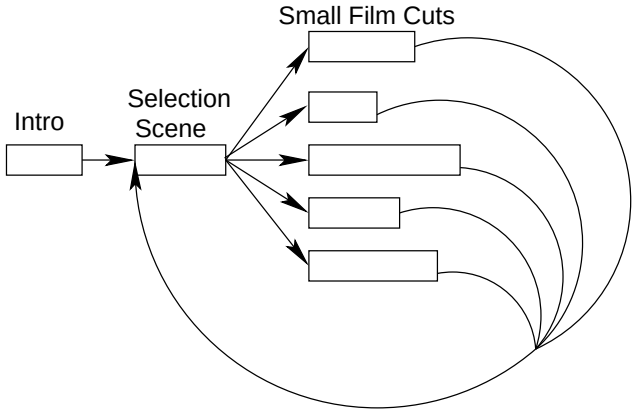

Figure 2.14: News on Demand

Advanced Level of Interactivity

Though some of the aforementioned articles use concepts from the AI research field to implement the syntactic rules of the symbols, and brings the content into the high level of interactivity definition. In this group we also find advanced systems that continously generate content from minute primitives such as vrml [30] and openGL [4] into multiuser games and flight simulators, etc.

Relevance for thesis

The thesis will consider medium level interactivity as the most interesting. Low level interactivity merely time shifts the play back point of available objects. It is the possiblity of the cache not having the correct object for transmission to end users at all that has the adverse effect on the cache's efficiency. Advanced level of interactivity is left as a work item for the AI field.

2.3.3 Branching Video

In which we define Branching Video as a compromise between the classical author centric narrative and the free form user centric interactive environment. A classical narrative [6] is predefined in its structure and presentation, the user is assumed to peruse the content in a front to end manner. In as much as the user breaks with the predefined serial way of reading or viewing the content, we say that the user interacts with the content to redefine it in the users own mind. Lev Manovich describes this in [35], in the principle of variability, where users can vary the available content in their own way, and for complex content, trace a new version every time. The available techniques for redefining the content defines the type of interactivity possible.

The hypervideo concept introduced in the previous section is based on a markup language scheme similar to html [66] called sgml [67]. Hypervideo is further described in:

[58] defines some common patterns in hypermedia. Hytime, [29], is a time based structuring language for hypermedia. The video sequences are objects within a context similar to that of a web browser, and playback is left to a plugin in the web browser.

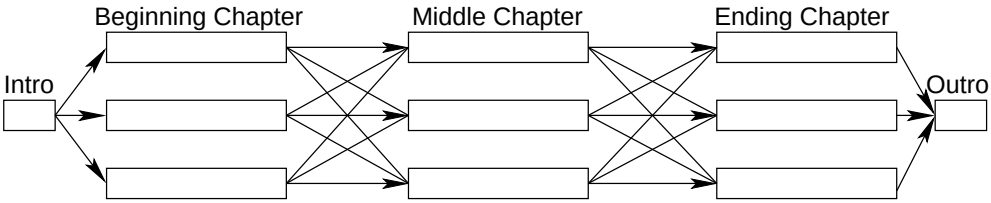

Figure 2.15: Branching Video

In the Branching Video scheme, we limit this to be similar to a video playback in a user terminal, with text and vrml capabilities as needed, suitably rendered as per the requirements in MPEG-4 Systems. An example structure is given in figure 2.15. Furthermore the primitives needed for rendering the audio visual content are provided by MPEG-4 Systems conformant Descriptors delivered inside Elementary Streams. The entire

presentation is within one application.

In a Branching Video, a video has a set of chapters that devides the timeline of the video, for every such chapter, a set of alternative Branches are available, each providing a different narrative. A scene where the user is presented with the choice of which content to play is called a Branching Point (BP). A scene where two alternative paths come together is called a Merging Point (MP). Finally, the segment of content in between these two points, is called a Branch. A film may be divided into smaller segments along its length and also along its width, where the width is taken to be the amount of Branches available at any given Branching Point.

The hypervideo in [58] had three different types of links, a spatial, a temporal and a textual. These links correspond to BP, and MP. Normally the BP and MP are temporal, according to the scenes presented so far in the narrative. But it is fully possible to keep a link available for a length of time whilst other content play back, whether that link is presented as dependent upon camera positioning, as in [58] or on text or graphics constituting a type of button is immaterial. The terminal issues a request to retrieve the content needed for playback if it is not already in local cache, and the object that is the target of the link is a Branch just as if it was a target of a temporal link.

Interactive Cinema

Interactive Cinema can be made with pre-programmed paths that a user follows and interacts with, or by a more recent approach generated in tandem between the user and the content application. Advanced content generators such as [13] are not central to this thesis. The well established method from the 70's and onward of fixing a set of media chunks in a structured graph that can be navigated by the user, called branched video suffices for this thesis [15].

Standard Efforts

[38] describes how the Blendo language developed by Sony can be used to create interactive television. [38] state that audiences used to high quality TV will demand the same of interactive TV, which is why the aesthetic of the content presented to the user is central. The term steerable media denotes continuous interactivity. The work is similar to the MPEG-4 [55], BroadcastCL [50] and X3D [19] standard efforts.

Two other interactive content systems are WebTV [72] and Flash [12] both in use today, both have less flexibility with joining content from various sources than the standards efforts mentioned .

Relevance for thesis

Branching video is very relevant for this thesis, it is this structure that will be considered in the analysis, design and implementation. However, we will use a rather limiting version of it, similar to the one presented in figure 2.15.

2.3.4 Navigation and Patterns

The user will navigate through the landscape provided by the audio visual primitives and their syntactic rules and map a path through the narrative. Though this might seem rather fleeting it is fully possible to analyse such paths a little. It is not a central topic in this thesis, rather we rely on the work done in [37, 61]. In the Branching Video examples we assume that the narrative is such that a video has a beginning and an end, when a user has navigated a path through the video, that video is finished as far as that single user is conserned. In learning applications [48] though a user might trace a path that eventually touches every branch. Still for large learning applications it is probable that there are many more branches than any single user will need to trace a complete path through the presentation. Figure 2.16 shows a branching video with fixed timeline, the user select different branches for every chapter, but no jumping back and forth in the timeline is permitted. Figure 2.17 shows a set of media content that has no internal structure, and the user can peruse them in any sequence, and it is not given that all clips will be accessed. The accordion pattern in figure 2.18 is a blend of steering the user in a single direction and allowing freeform style navigation.

Relevance for thesis

The exact patterns used by users to navigate through the interactive multimedia content is less important for the work in this thesis. Rather we assume that the patterns that the author has made available are sufficent for the presentation and that the users follow these.

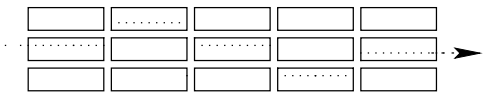

Figure 2.16: Fixed timeline

2.3.5 Caching Interactive Content

Caching interactive content is not trivial. For instance if a presentation is given as Tutorial.mp4 and is a very large file, a user will start playback of

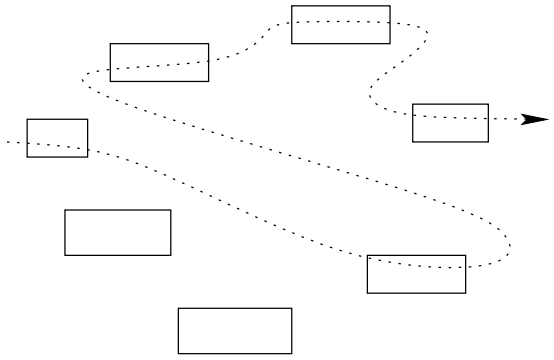

Figure 2.17: Freeform timeline

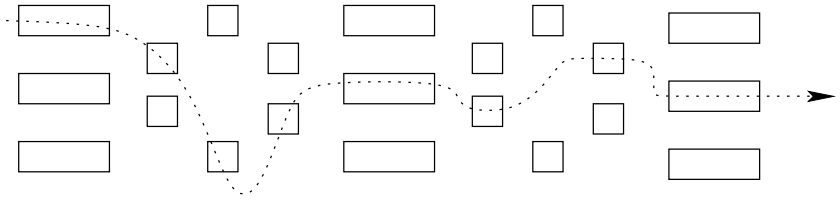

Figure 2.18: Accordion pattern

the beginning of the presentation and requests a set of branches that trace a path. If this presentation is then stored in a cache using the filename as an identifier, the next user will receive the exact same content as the first user, and any user terminal to server terminal communication will either fail or result in the content bypassing the proxy cache. The figure 2.19 shows the caching of a branched video with three branches cached, tracing a path from beginning to end. If content is stored unframed as pure media in the proxy, then the same path through the story as the first user traced out will be available. If the content is stored framed in a streaming protocol or as referenced content in a system of object descriptors such as MPEG-4, the missing content will not be available for playback in the user's terminal. In order to cache interactive content consistently, it is necessary to utilize object recognition of the same size and type as the primitives used to create the interactive content.

Relevance for thesis

This is the core point of this thesis. We hope to show that the proposed strategy will allow this to be possible and consistent. More on this in chapters 4,5 and 6.

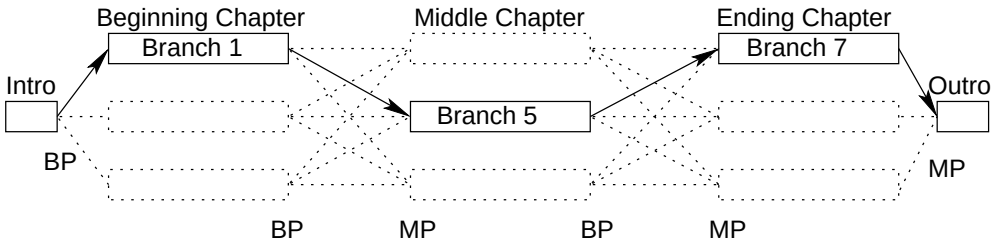

Figure 2.19: Caching a path

2.3.6 Applications of Branching Video

The following is a nonexhaustive list of different fields that could benefit from Branching Video.

- Feature movies
- Sets of short movies
- Sports shows
- News on Demand
- Company presentations
- Learning applications
- Webshops that display the items for sale
- Single state games
- Public Services eg., Health or Municipal

All these have in common that a fully functioning product can be presented by using just prefabricated audio visual objects. For more advanced simulators and AI assisted content generators to take advantage of this approach, prototypes shared by more than one user must be frequently reused without changing states.

Relevance for thesis

The thesis will not analyse this further. This section merely shows various fields that could benefit from the work presented in this thesis.

Chapter 3

MPEG-4

This chapter is an overview of the ISO 14496, also known as MPEG-4, with an emphasis on what is relevant for this thesis. Functions and detailed information that does not have a bearing on the thesis will only partially be presented here.

3.1 Overview

In this section we present the MPEG-4 standard as a short overview.

The MPEG-4 format provides standards for:

- A representation of media objects, either still images, video or audio, natural or synthetic, as well as animated graphics and scene descriptions.
- Composition of these objects into compound media objects form an audiovisual scene.
- The end user can interact with the scene through ways specified by the author of the source file.
- Elementary streams may be interleaved to ease transportation
- A new multimedia data interchange and storage file format, mp4.

The standard also codes other objects as text and graphics, talking synthetic heads, and synthetic sound.

A media object consists of elements that describe the objects within the scene, and any associated streaming data. Media objects are independent of surroundings and background, although logically they might not fit in anywhere else. For instance, cutting out a human from the background will look quite poor, since the outline of the object should blend in with the other objects. The lighting, colour and shadows would not be right

even if the perimeter of the object were cut out correctly. The figure 5.1 shows a scene description graph that contains three scenes with contained subobjects.

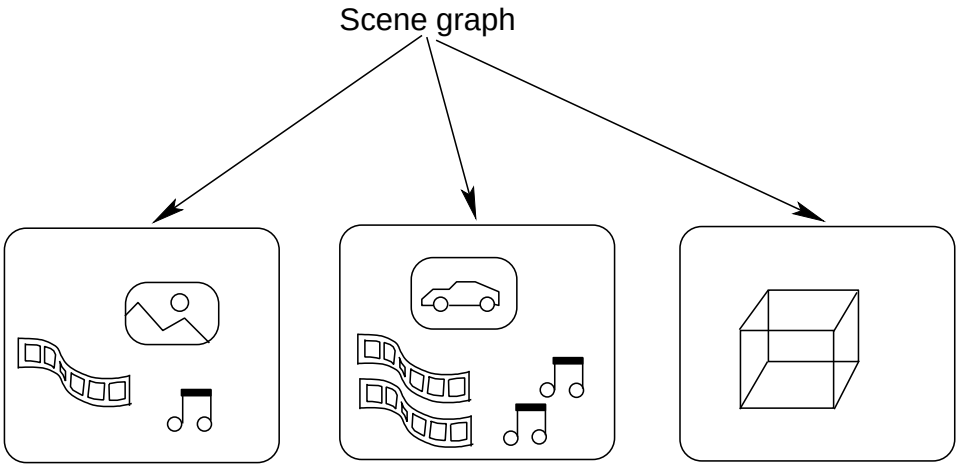

Figure 3.1: Media objects

The composition of these media objects might either be to form a single scene, or a set of scenes following each other or running in parallel in an arbitrarily complex manner. A compound media object can be a subtree in a larger compound object. This hierarchical structure allows flexibility for the authors in creating the content.

Functionality for changing the flow of the scenes is also defined. This consists of viewing or listening points within the scenes, and the startup of alternative scenes, or streams. If defined by the author, the end user might enjoy a high degree of interactivity with the content.

The standard also has new features to manage and identify intellectual property rights. This is implemented by storing unique identifiers issued by international numbering systems to each media object. The interface to these property rights can be used by other applications or operating systems.

The format is hierarchically layered into CoDec, Adaptation, FlexMux and TransMux layer. The synchronized delivery of streaming information from source to destination, exploiting different QoS, as available from the network, is specified in terms of the synchronization layer and the delivery layer containing a two layer multiplexer. Figure 3.2 depicts these layers,the figure is from [55]. The TransMux layer is an interface, either to a file system or a transport system over the Internet. The FlexMux layer interleaves elementary streams with compatible QoS together. The Adaptation layer synchronizes elementary streams, this is done with time stamping. The CoDec layer encodes and decodes the media objects.

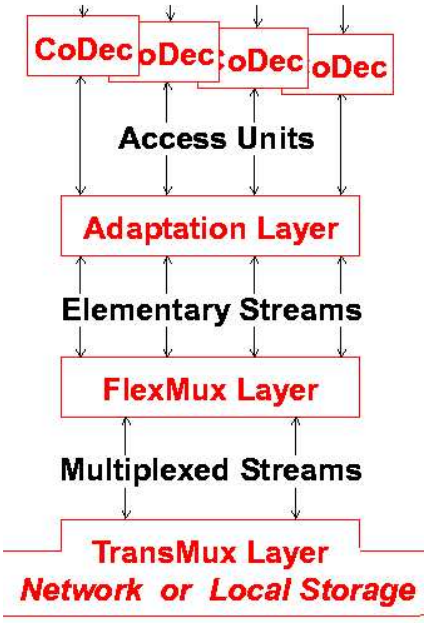

Figure 3.2: MPEG Layers

3.2 System

In this section we look into the central aspects described in ISO / IEC 14496-1 Systems. The full ISO / IEC 14496 standard consists of several parts, only part 1 Systems is necessary for this thesis.

3.2.1 Terminal

The unit that composes and sends or receives and presents the coded representations is called a terminal. The terminal is either a standalone application or part of a system. The architecture of the terminal is layered into Delivery Layer, Sync Layer and Compression Layer. The Delivery Layer is partly specified in ISO/IEC 14496-6. Above the Compression Layer is the Compositor that consumes the coded representations for presentation. The compositor is not specified in ISO/IEC14496-1. The boundary between the Compression Layer and the Sync Layer is called the Elementary Stream Interface (ESI). The boundary between the Sync Layer and the Delivery Layer is called the DMIF-Application Interface (DAI).

3.2.2 Delivery Layer

The Delivery Layer provides transparency from delivery technology. Furthermore the Delivery Layer manages real time QoS sensitive channels,

resource logging, and ensures end to end interoperability. The DL is implemented as DMIF instances, pertaining to specific technologies. FlexMux channels multiplex SPS, creating FlexMux packets and headers. The FlexMux operate toward the DMIF Application Interface on top and toward the protocol stack downward. The FLexMux interleaves SPS to provide easy embedding into existing transport protocols. Which storage or transport media that are used in the protocol stack at the bottom of the Delivery Layer is implementation dependent, these are called TransMux channels. The TransMux channels may be based on interactive network technology, broadcast technology or storage technology. Ranging from udp/ip to mp4.

3.2.3 DMIF Application Interface

DMIF Application Interface (DAI) lies between the Delivery Layer and the Sync Layer. The units passed between the two layers are SL-Packetized Streams (SPS). A SL-Packet is either a whole Access Unit (AU) or a partial AU, along with a SL-Packet header. SPS is then a stream of SL-Packets.

3.2.4 Sync Layer

The Sync Layer syntax is configurable, and can be empty. To parse SL-Packet headers the SLConfigDescriptor must be known. The SL adapts the streams coming down from the ESI to be sent over the DAI. Providing timing and synchronization information, fragmentation and random access information. Incoming SPS are stripped of SL-Packet headers and delivers AUs to the Decoding Buffers over the ESI. The SL may duplicate SL-Packets and AUs for error resilience, such duplicates follow immediately after the original.

3.2.5 Elementary stream Interface

The Elementary stream Interface, between (ESI) the Sync Layer and the Compression Layer, models the interchange and control of Elementary Streams (ES). Decoding Buffers (DB) consumes Access Units and delivers them to the Decoders. The streams of AUs coming out of buffers are considered Elementary Streams.

3.2.6 Compression Layer

The Compression Layer (CL) holds the decoders and encoders. The decoders breaks up an AU into an integer number of Composition Units (CU) which in turn is provided to the composition memory. The composition memory is available to the Compositor, which rebuilds the scenes

according to the Scene Description. The Scene Description is carried as the compact binary form BIFS in BIFS Access Units. Object Descriptors (OD) are the building blocks of the object description framework which links the elementary streams to each other and provide descriptive information regarding each stream. The various ODs are also carried in AUs. The ES_Descriptors are linked to ES, and are the most important. The IPMP provides copyright protection. The OCI may provide additional information. The ODs may build a complex recursive structure. The compositor uses or skips CUs that are available (unavailable are skipped) at the time corresponding to its Composition Time Stamp (CTS). Encoding terminals produces AUs from its CUs with encoders. How the CUs fit into the AUs is determined by the encoder. A receiving terminal may send Upstream Information in return to the sending terminal, such information might be user interactions or any other function the sending terminal implementation allows. Java Byte code may be downloaded to enhance functionality in the receiving terminal. Upstream Information pass through the same layers as the normal content in reverse. Upstream Information Streams are always dependent on one normal elementary stream. There are one Decoder Buffer, one Decoder and one Composition Memory for every stream.

3.2.7 Timing

The timing model presented in ISO/IEC14496-1 is designed for push applications. The terminal keeps a System Time Base (STB). The STB is not a global clock for all terminals, merely the notion of time for one terminal. A data stream keeps an Object Time Base (OTB) which may be configured in a number of ways. An OTB may be a reference to another OTB. The STB of a terminal doesn't have to be in reference to any OTB. The OTB may be carried in a stream created for this purpose only. The sending terminal conveys an OTB to the receiving stream decoder with an Object Clock Reference (OCR) which is the time stamp set by the sending encoder in the SL-Packet header. Each access unit has a Decoding Time Stamp, which is the precise time it shall be available in the decoding buffer. Each Composition Unit has a Composition Time Stamp, which is the time it must be available in memory. The exact frequency and usage of the time stamps are dependent on the application and chosen profile. Objects that constitute dependent elementary streams for scalability purposes may have the same time stamps.

3.2.8 Object Description Framework

The Scene Description and the Elementary Streams are the parts needed to build ISO 14496 content. However the scene description has no direct information about the ES. The Scene Description has links to Object

Descriptors which indirectly links to the ES. As shown schematically in figure 3.3, the figure is from [23]. This allows the Scene Description and the ES to be changed independently. Additionally the OD may aggregate several ES that form one object in the Scene, or several alternative ES that may provide scalability or interactivity. Furthermore the OD may hold new Scene Descriptions in a recursive manner, providing a very flexible system (inline). Such new Scene Descriptions may have ODs pointing to another set of streams already available, or to URLs to remote streams. The first Scene Description is within the first Scene Description Stream pointed to by the initialObjectDescriptor, which must be conveyed to the receiving terminal in a way not specified in ISO/IEC 14496. The initialObjectDescriptor and the Elementary Streams is shown in figure. The initialObjectDescriptor also points to the first OD stream. The ES that contain visual, audio or other data are given by ES_ID a numeric held by the ODs within the OD stream. The ObjectDescriptor Identifier (ODID) is unique number within each naming scope. The Elementary Stream Identifier (ES_ID) is also unique within the same naming scope. An inlined node opens a new naming scope. Inlined nodes point to object descriptors that points to a new set of Scene Descriptor Stream and Object Descriptor Stream, and possibly more ES. The Intellectual Property Management and Protection (IPMP) system is not specified in ISO/IEC 14496 but is implementation dependent. IPMP descriptors components in ODs may point to a system, or to an ES of IPMP_Descriptors that convey time varying keys and such. ES pointed to by an IPMP ES implies that the objects therein is protected by the IPMP system. Object Content Information (OCI) is another OD component that may be associated with an OD or conveyed in a stream of its own. The OCI components specify various optional meta information about some ES.

3.2.9 Scene Description Framework

Since the ISO/IEC 14496 standard has coded representations of many different types of objects, the composition of these to a complete scene must also be represented. The Scene Description Representation is called BInary Format for Scenes (BIFS). The Scene Description is then a tree of BIFS nodes with internal structure. Each BIFS node represent an object within the scene. The coded representation provides the spatial and temporal information needed, attributes like sound volume, behavior of audio-visual objects as well as the links between objects. In every such node there is a pointer to the OD that again points to the ES that make up the object in question. One BIFS AU contains either a BIFS CommandFrame or AnimationFrame, which may describe a complete scene or a change to an existing scene. The framework relies heavily on VRML ISO/IEC 14772-1:1998. User interaction on the receiving terminal side is enabled by this

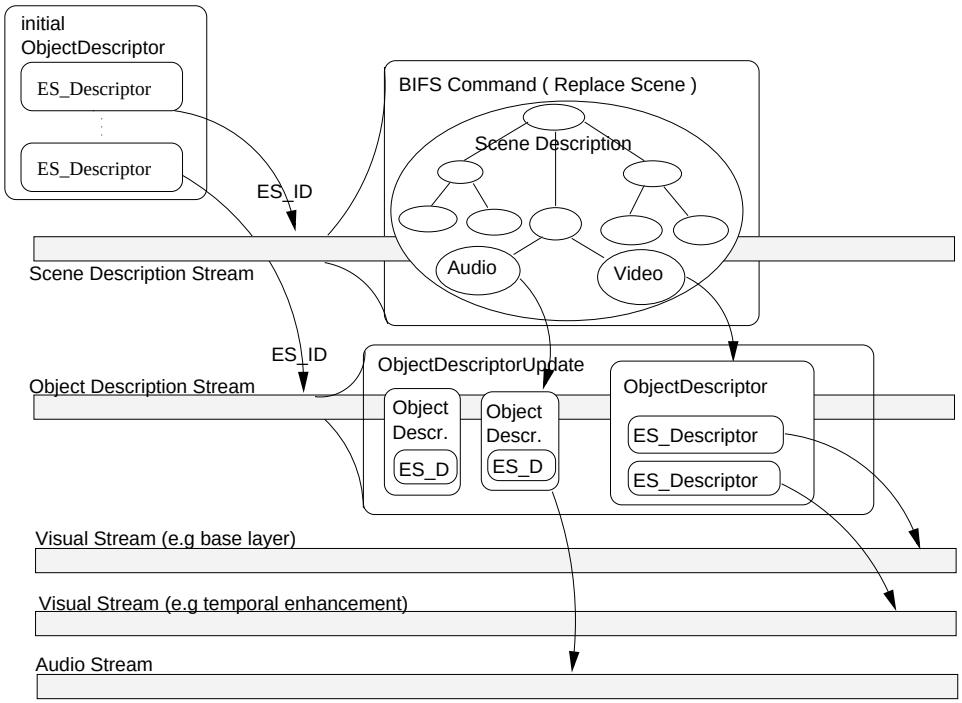

Figure 3.3: Elementary Streams and the Object Descriptors

framework. Since the BIFS nodes are within an ES that may be pointed to by an OD, the scenes may be dynamically altered at the receiving terminal and even jump to new content not available at the beginning of the playout. Figure 3.4 shows how a chapter might be implemented in MPEG-4, note that chapter is not a MPEG-4 descriptor.

3.2.10 MPEG-J

The option to have downloadable Java Byte code provides two uses. The first is that the player may adapt to changing characteristics and degrade the streams according to the resources available. The second is the increased interactive functionality. The MPEG-J is a programmatic system as opposed to parametric, and specifies interfaces for an MPEG-4 media player.

3.2.11 MP4 File Format

The MP4 file format is designed to support TransMux in general, but is independent of the specific TransMux mechanism. The file format may be used in different ways. As an interchange format all the media are contained in one file and the file don't reference media in other files. The

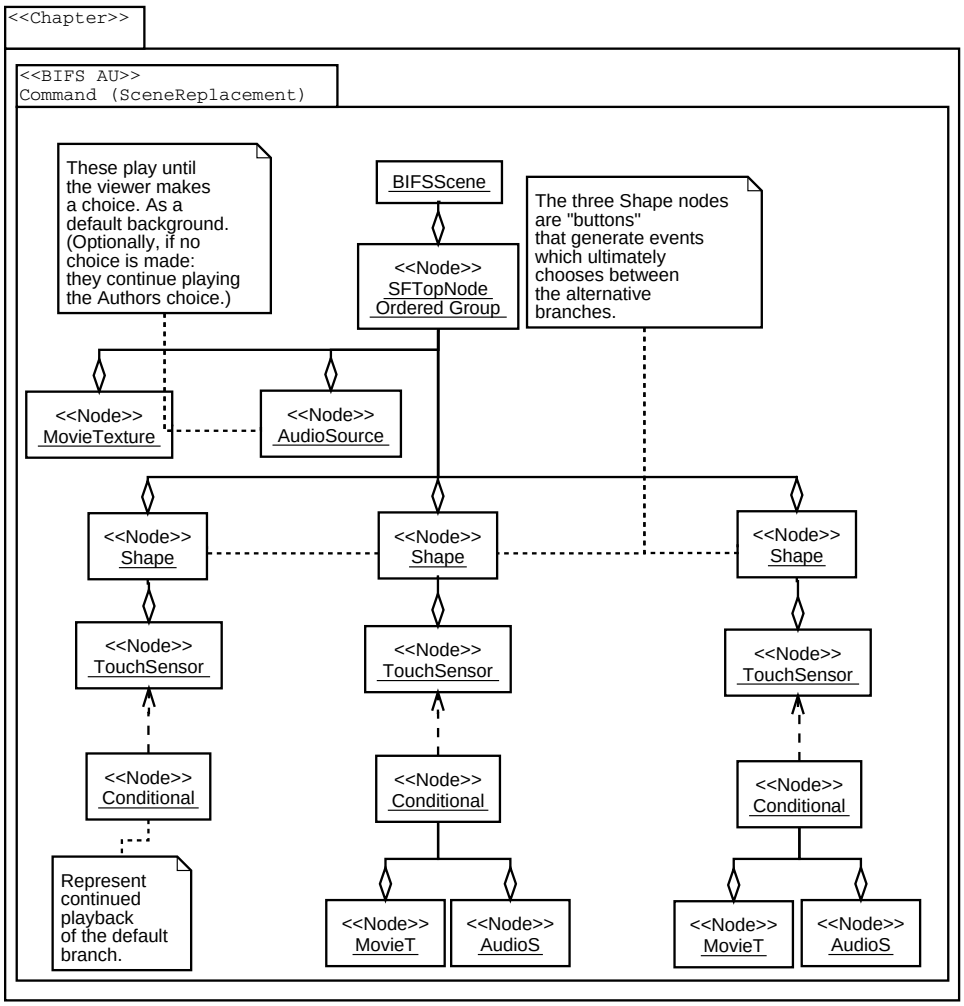

Figure 3.4: Scene Description in Branching Video

interchange format does not contain any TransMux information. When supporting content creation the file format is more flexible and may reference media in other files. As a preparation for in streaming the file format must contain information for the streaming server. Furthermore interleaving of the various media is helpful to avoid seeking during streaming. Local presentation needs a file format that supports full random access, as well as interleaving to avoid seeking on DVD and CD. A streamed presentation should not contain any information about the file format itself, but comply with the specified protocol. It is possible to keep the media data on read-only media and just augment it prior to streaming. The file structure is object oriented. The media is not framed by the file format, but appears in their 'natural' state, as access units. Meta data (hint tracks) is used to reference the media. If FlexMux is used the hint tracks must be designed in a such a way that TransMux independence is lost. A presentation may be contained in several files. Meta data about the entire presentation is stored within an object called the movie atom (moov). The file containing the moov object may also contain all the other media objects, or merely reference them. Every TransMux mechanism has its own hint track format.

Chapter 4

Analysis

This chapter will analyse what is needed for consistent caching of streaming interactive premade stored multimedia content. We will work with the work items presented in chapter 2 and build upon this in the investigation to provide a foundation for the design presented in chapter 5.

4.1 Caching interactive content

This section presents the fundamental problems that arise if the content delivery network does not treat interactive content correctly. The type of content this thesis considers are premade by the author and intended to be played back in a presentation in an author determined way. The content is stored on server, which is an important difference from for instance webcams, which are not stored but still premade. Furthermore the content may be a mix of real or artificially made content, this is not important for the analysis or design in this thesis. Mixing of real and artificial content is defined fully in MPEG-4 and both are treated as being elementary streams. This thesis will consider elementary streams as the lowest level of detail we regard. The type of codec, amount of frames per second and so on is not important for this discussion. We will try to keep the analysis on a general note. The final but most important describing characteristic of the content is that it is interactive. A few problems arise from this final characteristic. We will examine them more closely in the following subsections. The figure 4.1 show how the MPEG-4 descriptors might be for a branching video of 3 chapters and 3 branches, where the concept of chapter is merely a single SD AU and not a MPEG-4 descriptor of its own.

4.1.1 Consistency

For caching to be useful it must be consistent, this is not a performance metrics, rather a minimum requirement. Non-interactive content may be

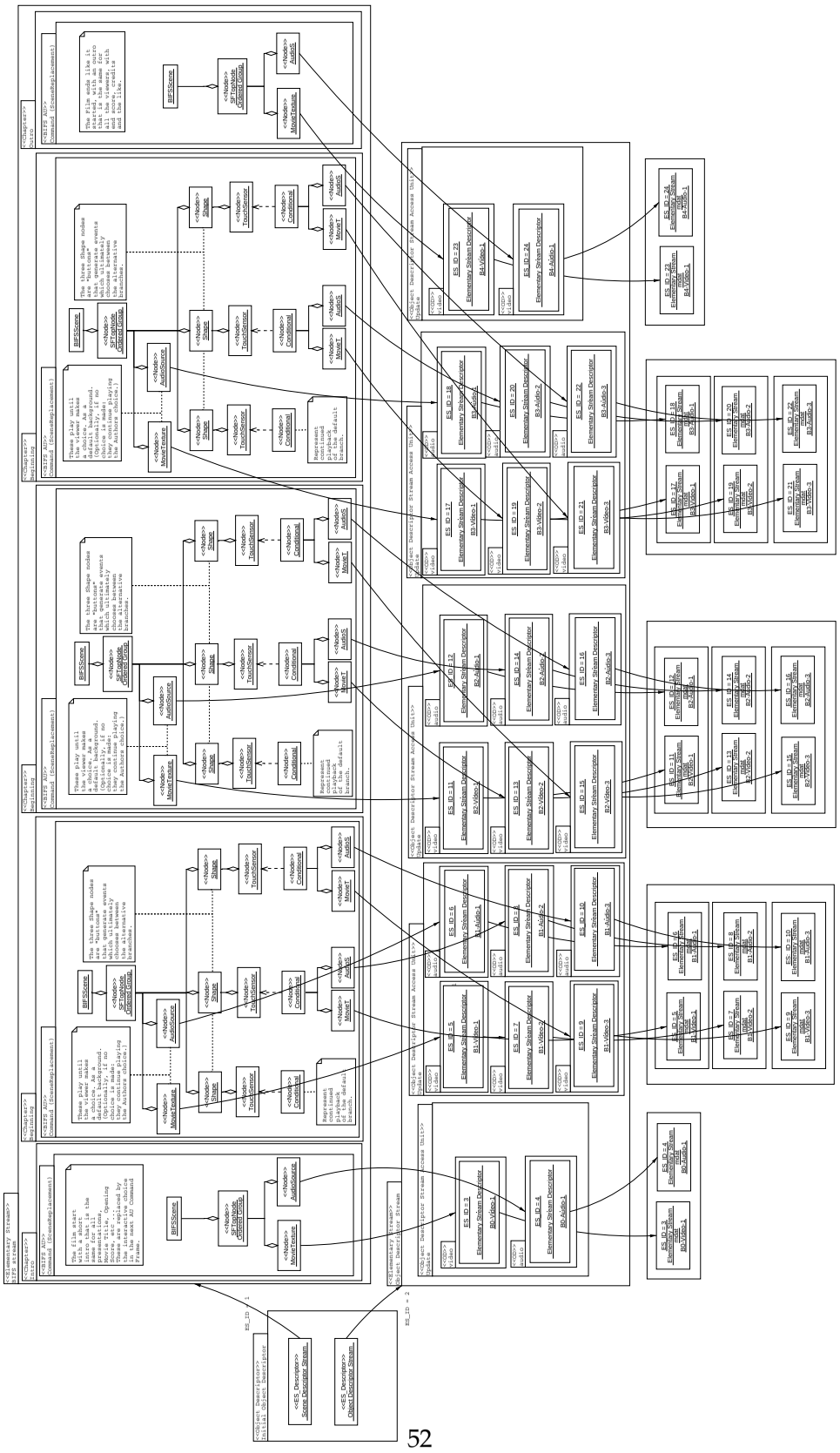

Figure 4.1: 3x3 Branching Video in MPEG-4

played in a presentation in a linear manner, and though the presentation may consist of multiple elementary stream, it is after all multi media. The timeliness of the multiple elementary streams are defined by one timeline, and sound and picture go together. In interactive mulitmedia the objects in the presentation may appear in seemingly random places, and the playback may not be linear. This means that algortihms in the caching scheme in the proxy cache server will not necessarily recognize the correct objects when trying to cache the content. Thus the requirement of consistency is broken.

4.1.2 Object Reuse

The intent of any caching scheme is to reuse objects stored in the proxy for several user requests. For reuse to be possible it is vital that the objects stored in the cache are eligible for reuse and identifiable as such. Authors of different presentations may use different methods to define the interactivity in the presentations. This means that the objects that are the building blocks of the presentations may change in type and size from one presentation to the next. This means that the algorithms in the caching scheme in the proxies will have a hard time swapping one object for another. Additionally since the objects are in the interior of the presentations' namescope, it is not certain that the objects are identifiable at all, see figure 4.2.

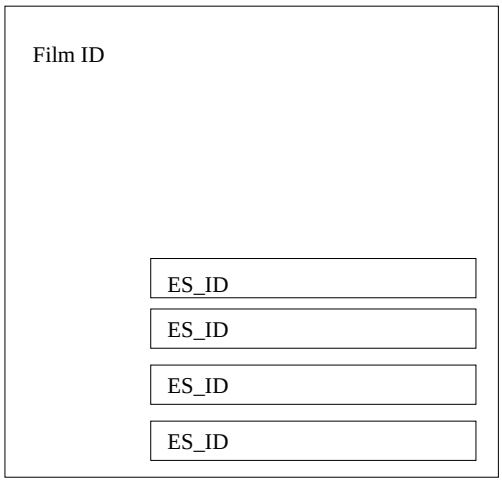

Figure 4.2: Hidden ID

4.1.3 Binary or Partial caching

The proxy must be able to cache the content in a consistent manner, meaning that not only are the objects in the cache reused, but they are

reused in the correct context and according to the timeliness requirements of the presentations. This can be done by either caching the presentations in their entirety, and providing the same kind of service as the source server. This method is called replication if the candidates for caching is chosen by the source server, replication is not the theme of this thesis. If the caching candidates are selected by the proxy we call this binary caching. Binary caching of interactive presentations are similar in algorithm logic to that of non-interactive content. Binary caching of non-interactive content is a vast research topic of its own, and is not within the scope of this thesis. Interactive content is usually larger in size than, due to the alternative paths the users might navigate through the content. This means that interactive content will loose in competition with non-interactive content in a binary caching proxy. Since not all the objects in the interior of an interactive presentation will be requested by users, this means that caching an interactive presentation in its entirety on a cache is a very efficient way of wasting resources. The next section will look into partial caching of content, and this is the chosen method in this thesis.

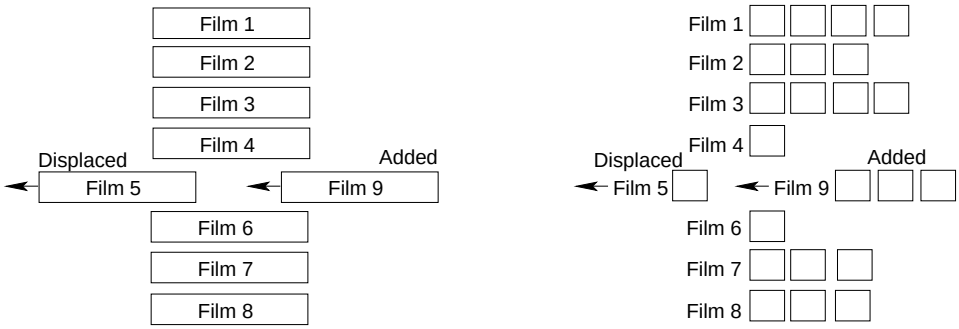

Figure 4.3: Binary vs Partial

4.2 Partial Caching Analysis

Partial caching of content was defined in section 2.1.7. In this section we will look closer on how this method of caching can meet the requirements we have set. We aim at consistent caching of content. Caching is not a necessity in itself, rather a means to reduce resources and boost the all important QoS for experiential presentations. This means that a caching scheme must not introduce more overhead than the resource gain it yields. Partial caching is a very effective way to keep only the very hottest objects in close reach of the end users. By close reach we refer to both server hops and latency. Reducing server hops reduce network load, and reducing latency increase the potential for a successfull playback of the presentation from the users' point of view. There are several ways to cut down the size

of the content, which in short is what partial caching is. They are listed here as three different domains. The first two domains was defined by [59], and the third is the novel proposal of this thesis. The idea presented in QBIX that a cache can also serve as a gateway that adapts the content for different sets of specifications still apply. We will not consider this in detail however.

4.2.1 Time Domain

The time domain caching is presentated in 2.1.7. Caching interactive content in the time domain means that each and every branch that makes up the presentation is cut into segments. The usual modus is then to keep the first segment, called the prefix, and later add the remainder, called the suffix if the content is popular enough. For this to work with interactive content, the algorithm in the caching scheme must know how the objects make up the timeline of the presentation. In a branching video this could implemented by adding a index number to each object that identifies when in the timeline it is used. However, since we are talking about interactive content here, objects may be used in a random sequence according to the intent of the author and requests of the users. This adds an element of inefficiency to this approach. Since the algorithm is already considering index numbers on individual objects in the interior of the presentation's namescope, we can expand upon this. The proxy can consider each presentation and try to identify the objects it is made of. More on this in subsection 4.2.3. If we are already applying a scheme such as the one presented in 4.2.3, the proxy can use time domain caching on individual objects, particularly if they are long.

4.2.2 Quality Domain

There are several ways to use qualtity domain caching, we can use an adaptive algorithm, or we can drop dependent elementary streams. Either way there are no direct problems associated with employing quality domain caching on interactive content. Quality domain caching is done on an even footing throughout a presentation, and it matters little where in the timeline an object occurs. It is interesting to note however, that if a presentation is split by the author of the content into dependant streams, one being the base level, and the remaining being temporal enhancements. This is similar to the way interactive content split a narrative presentation into constituent objects. If a presentation is split like this, in both timeline specific branches, and quality specific dependant streams, it is easy to use quality domain caching in unison with extent domain caching.

4.2.3 Extent Domain

Extent domain caching is simply keeping only the branches that the users request often. Rather than bothering with prefix and suffix, and internal graph relationships, the caching scheme is solely conserned with the popularity of the individual caching candidate. This reduces the problem of caching interactive multimedia content to that of recognizing the individual caching candidates. More on that in section 4.3.1. As mentioned above extent domain caching is perpendicular to the other two domains, and can be utilized in the same scheme if more powerful compression of the content is wanted. There is however, one similarity between extent domain caching and time domain caching. It is possible to argue that extent domain caching is simply an advanced form of time domain caching. The point we make here is that, extent domain caching is done on the scope of the entire presentation, whilst the time domain caching can be done on the scope of the individual branches that presentation is made of. In time domain caching it is completely random whether the prefix of the presentation will prove to contain the most popular segments or not. As opposed to extent domain caching which considers this as the primary task.

4.3 Knowledge needed

What is the knowledge needed by the proxy cache server to cache the interactive content. As introduced in the previous section, we will argue that extent domain caching reduces the problem to that of identifying the individual caching candidates, and that they are in the interior of the presenations' namescope. The interactive multimedia presentations are made up of a graph of multimedia objects, which in turn can be an aggregation of several objects. Therefore this is not a trivial task. There are two primary problems, identifying the objects as unique and reusable objects in the context of the proxy cache. And finding the correct boundary of the objects, since we are working with an arbitrarily complex graph of constituent objects. The problem of identifying the objects are subject to the problem of finding their boundaries. Furthermore, for different types of content, different styles of author work, and different ways the users want to navigate through the content, the frequency of context shifts will vary. What is seen as a single object in one style might be regarded as a set of objects by a new set of users. This is a non trivial problem, and we leave solving this as future work. For this thesis we will assume, that the content defined as branching video in 2.3.3 has a branch as the building block object. The task is to keep a score of popularity for the each and every branch, and to keep tabs of which branch is in which presentations when the users request them.

4.3.1 Candidate Recognition

To recognize objects identifiers are needed, which must be unique in the entire system, and remain so for the duration of the source objects lifetime. The identifier of the source object must corespond with the identifier for the cache object. For large object of non-interactive content this is fairly easy. For interactive objects, an easy approach where the URL of the source server together with the filename comprises the identifier is not sufficient. Since such an approach would imply caching the file in its entirety enabling the proxy to fully replicate the interactivity. In this thesis we will suggest fine grained object recognition as an approach. Fine grained object recognition enables caching of the objects that are reused most often, whilst still refraining from caching very large source objects in their entirety. An interactive file may be several times larger than the content any single user actually accesses.

4.3.2 MPEG-4

In the MPEG-4 standard several sub objects are natural candidates for caching. Starting at the top with the largest object, we have the Elementary Streams (ES) which can be identified within a file with their 16 bit ES_ID. Some ES are quite large whilst others can be very small. The ES are comprised of Access Units (AU), which lies back to back within the stream. These are also of varying size, still significantly smaller than the ES. The AUs can be identified through the ES_ID and random access information in the ES headers, [24, 25]. The AU may have a sequence number, which is a modulo counter for every ES. AUs begin with accessUnitStartFlag and end with accessUnitEndFlag. Only AUs that are in ES that have all random accessible AUs or AUs that have the randomAccessPointFlag set may be accessed directly.

Hierarchy

Due to the hierarchical architecture of MPEG-4 caching decisions may be on several different layers of complexity even if we only cache AUs or ES. The Scene Description Stream (BIFS stream) contain commandFrames within AUs, one commandFrame for every AU. The special commandFrame SceneReplace is the only random access point in the BIFS ES, and describe an Audio Visual Scene which in turn describe one or several Audio Visual Objects, which in turn may consist of one or several ES that contain media data. The caching may be on BIFS AUs, or on media data ES, or on media data AUs. Caching the BIFS ES would be the same as caching the entirety of the file, since the BIFS ES is one of the elements that enable the interactivity.

Keeping cache objects on the BIFS AU level of complexity, means

keeping track of structured sets of sub objects. The BIFS AU only hold meta data, a cache object comprises of several ES indirectly described through the Object Descriptors pointed to by the BIFS commandFrame. To save a copy of the cache object data must be collected from the Scene Description Stream and then the Object Descriptor Stream before the media data held in the media data ES can be accessed.

Keeping cache objects on the media data ES level of complexity, means keeping track of the source server, the file accessed, and the ES_ID. The media data will be accessed by BIFS AU as above, but will remain open to other scenes then the one they originally occurred in, increasing the chance of reuse. Some object descriptors point to new ES outside the ES_ID scope of the original multimedia file. The ES may be kept in its entirety, or partially by segments of several AUs.

Hybrid

Keeping entire media data ES or just media data AUs is mainly a decision of the size of the cache objects. Entire media data ES may be very large, whilst keeping track of all the AUs may be to much work. A compromise could be to keep segments of ES as series of AUs, even if this will aid in optimizing the space usage of the caching algorithm, the amount of bookkeeping is equally large as just keeping the AUs. For some content, depending on the author, reuse of AUs may also be interesting, not just entire ES. Keeping AUs will potentially create most cache hits, but at the expense of more computation in the proxy. It is not certain that this will be a more efficient scheme however, since the interactivity provided by the author might use larger structures as the building blocks.

4.4 Minimum knowledge needed

In this section we assume that the proxy has the same knowledge about the source content, its users and the network as when only non-interactive content is cached. The additional knowledge needed to provide a consistent caching service will constitute the answer to the premise question in this thesis.

4.4.1 Identity

A caching scheme seeks to keep objects in its cache which are likely to be reused. These objects must be maintained in such a way that they are still usable, they should be accessible for the users, and they should not become incorrect or outdated. These issues are resolved in caching schemes for non-interactive content by constantly updating the cache with popular

objects. In a scheme that caches content that has an internal structure of interactivity, this is no longer sufficient. An interactive presentation that is hosted on a source server offer several redundant renditions of the content. Whether this is for narrative or technical purposes is less important. A redundant object will be provided in place of another object, in the presentation for the user this is fine. However, when the content pass through the proxy cache server, the stream of media data the proxy sees is determined by the first user to request that presentation. The stream seen is a result of the path taken by that first user. When the next user requests the presentation, it is not certain that this user will navigate the exact same path through the content. The identity of the presentation no longer points to a unique and atomic object which the proxy can perform caching algorithms upon as for non-interactive content. The presentation must be regarded as an aggregation of several objects, and it is necessary for the proxy to able to identify these. If the proxy can identify the building blocks of the presentation, then the cache can provide these to the user, enabling the user to navigate through the content in the same manner as if it was stored on the source server. This enables the proxy cache server to remain transient to the user.

4.4.2 Boundaries

The objects that appear in an interactive presentation might be easy to identify if using the same type of object recognition as the source content. Still, the users will probably have the possiblity of only using part of that object in the presentation. Similarly the author of the presentation might use only a subset of an object. If the boundary of the subset of the object that is used is not provided in the mechanism for object recognition, it is difficult for the proxy to find them. If the users request part of an object from a source server through a channel that does not leave information in the proxy, there is no way for the proxy to know that the transmitted part of that object does not constitute the entire object. When later requested, by means of its identity, the proxy might offer an incomplete object to the end user. In fact, the same argument that applies to the identity of the objects that are aggregated together to form an interactive presentation also recursively apply downwards in the internal structure of the presentation if it is rendered by subsets of aggregated objects. The granularity of the objects used as atomic building blocks in the presentation will be the ideal caching candidate. However, it might be inefficient to cache an enourmous amount of very small objects, particularly if they only rarely appear on their own. Each independent caching candidate warrants its meta data in the caching algorithm, and the lookup functions must consider every one. There is no non-trivial way of avoiding this problem. Chapter 7.2 talks about a meta algorithm to determine the granularity of the candidates.

In this thesis we assume that the granularity of the caching candidates is on the same level as in the presenations. Any inefficiency introduced by users stopping or seeking within an object is accepted, and refered to future work. Figure 4.4 shows a cache object containing three AUs, the boundary of the cache object is an important attribute. The proxy risk serving the wrong subobjects to the users if the boundary of cache objects are undefined.

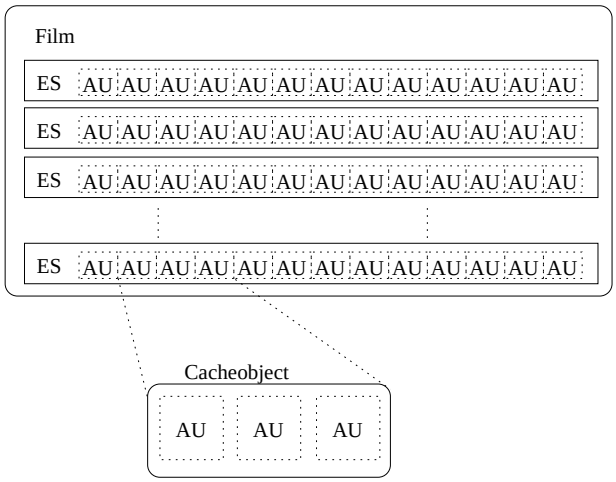

Figure 4.4: Boundaries

4.5 Media Types

In the previous section it seems that different types of content would need different strategies for caching, additionally it is easy to envision content that is pointless to cache at all. This thesis will not consider three dimensional games using shared state or other such content that generate highly detailed information in real-time, but never reused. Another such type are teleconferences. The caching scheme outlined will consider films with interactive storylines, interactive TV shows, and limited interactive games that has branching private state content. The term common for these types are branches in the storyline. For now, lets assume that only applicable media are used. Note that OCI descriptors could be a natural way to tag content that should not be cached with such a scheme, however we will not elaborate further on this in this thesis.

4.5.1 Branching

In media where there is only branching of the storyline the discussion in the previous section would favor caching of ES. One would assume that

the content is broken into sizable segments between the branching points, which would simplify the bookkeeping in the caching scheme. However some types might reuse smaller units very often, for instance in a TV show, the logo and jingle of the show would reoccur often. This also favors caching of smaller units like the media data AUs. The system could of course also consider all types of AUs for caching, keeping statistics for every one. If the BIFS AUs are reused often in one file, they are also cached as such. But if only some media data AUs that are a subset of the content indirectly held by the BIFS AU are popular, only those AUs will remain in the cache. This would need a more sophisticated system to keep track of the objects on the varying levels of composition.

Minimum

Since the thesis shall investigate the minimum amount of knowledge needed to consistently cache interactive multimedia scenes, keeping track of varying levels of composition is probably not a necessity. Note that the natural objects in the MPEG-4 standard are all described by Object Descriptors (OD), these are not candidates for caching, since multiple ODs may reference the same ES. The knowledge the proxy has about the BIFS AU is clearly a smaller amount than about media data AU. Still caching on that level of complexity might limit the consistency with which the system can be said to cache interactive scenes. An assumption, that authors have created content in such a way that an echelon of BIFS AUs, can be identified, and cached in persistent state, without limiting the interactivity, is needed. For instance, an example interactive movie might have a 'top level' BIFS AU that describe the primary objects the end user interacts with. Beneath this 'level' is a new BIFS AU reached through ODs within the first, that describe the various branches the end user might choose between. Obviously, if the system cache the 'top level' BIFS AU the movie will be locked down in a persistent state that has no interactivity at all, or indeed reference the entire movie with all possible branches, both approaches clearly disrupt the proxy's efficiency. However, making decisions based on the second level of BIFS AUs would keep sets of objects that allow some amount of interactivity. But the type of interactivity where a user can jump out of a branch and switch to some other point entirely will undermine the efficiency of this approach. The system would in fact be somewhat sophisticated since it needs to discern between top level scene descriptors and second level scene descriptors. The user terminals must at be able to identify the ES at playback time, hence the proxy should be able to identify the ES as well. The ES are refered to from ESD in ODs and may be reused several places in a presentation, or only used in part. If an ES is reused several times, it only adds to the proxy's efficiency to cache it. If it is used only in part, the proxy's efficiency is decreased. By keeping the ES

small, that is; on the same level of size as the interactive choices in the presentation, the author of the content can make sure that the presenation is easy to cache. If all interactive objects are stored in the interior of one monolithic ES, the proxy will have to utilize additional algorithms to see which segments of the ES that are requested, and cut these out as caching candidates. This is not the approach assumed in chapter 6, rather we assume that each branch is stored in its own set of ES, easily identifiable. Breaking monolithic ES into smaller segments are much investigated in time domain caching, 2.1.7, and is not a central topic for this thesis.

4.6 Storage

In this section it is assumed that the caching algorithm considers the media data AUs as atomic cache objects, and stores them sequentially in a segment if they are from the same ES. The natural way to store these objects is in the mp4 file format. However, a complete mp4 file format storage system is not required for the thesis. The MPEG-4 standard has support for external URLs as a part of the file system. The test proxy cache server receives MPEG-4 content and delivers MPEG-4 content, but does not have to store MPEG-4 compliant mp4 files. The mp4 file format does not frame the media data, rather the meta data include the media data by reference. This enables the media data to be stored in its natural state, and the same data can be used for disparate protocols. The test server will only use one protocol but the media data will still be stored in the same manner, in its most natural state as access units (AU), a range of contiguous bytes for each AU. The AUs are transmitted in the stream on byte boundaries, constructed by the SL Packet headers. The intermediate storage system needs to save the AUs in the same manner, so the correct AUs can be reconstructed when requested by new clients. Every ES needs some meta information about its source and what AUs it consists of. An ES needs not be stored in its entirety, since AUs are the chosen atom, and the length of the ES might therefore be of a different size than what the clients request. The remaining parts of the ES must be streamed from the source server, and the AUs of the proxy cache and the AUs from the source server must of course stream 'back to back' to avoid disrupting the clients playback.

To reference the set of media data the client request the proxy searches its cache for meta data in the following order: source server, mp4 file, ES Descriptor, AU. Since the AUs are the smallest building block the cache will contain an ES consisting of one or more AUs. The decoding and composition times are kept in the SL Packet header, and not within the AU itself. The payload of a SL Packet is either a whole AU or a fragment of an AU, all meta information needed in the decoder is kept in the SL Packet header. In the mp4 track structure the SL Packet headers are stored

interleaved with the AUs they are headers for. The media data atoms are really storage of an SPS and not ES. However the SL layer is configurable and its syntax may be configured to be empty. The figure shows the meta information kept in the SL packet header. The payload following the header on the next byte boundary is the AU to which the header applies. Storing pure AUs back to back would result in loss of timing information as well as loss of the AUs boundaries.

4.7 Interactivity

In this section we will investigate how the proxy tell the difference between two different choices for a branch. When a user interacts with the source server and chose a particular path through the tree of branches, only a contiguous set of branches that make a whole storyline pass through the proxy. It is important to identify the disparate ES that make up the set of choices for the storyline. This is already supported in the scheme outlined in the previous section. When the user reaches a branching point, that point is either identified by an option to jump to another ES, or with a set of new ES to chooses between. At a merging point the storyline might either jump into another ES (similar to branching point, but with no option), or begin at the beginning of a new ES. However this gives rise to a weakness. If one ES is used in several branches but retains the same ID all the time, the proxy cannot distinguish between the popular subsections and the unpopular subsections of that ES. Significant work must be done to resume storage of new AUs into a partially cached ES that might have gaps, and even lack its beginning AUs. All these problems are solved if the authors of the media content comply with the following limition. All interactivity that give rise to a change in the ES transmitted should change the ID of all the ES involved at the time of the branching point. That is, there is no jumping out of or into an ES, see figure 4.5 for an illustration of the difference. All the ES are transmitted from beginning to the end. Of course the end user might opt to cancel transmission and restart somewhere else.

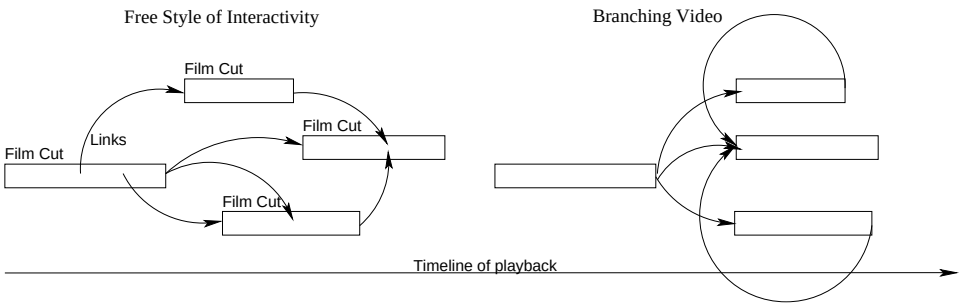

Figure 4.5: Branching Points

This means that at both branching and merging points the ES ID change. This might be a limitation in flexibility for the content author, but it allows for easy cache replacement decisions. The granularity of the cache replacement algorithm must be the same as that of the interactivity. For content that has much smaller scenes, the ES segments will be shorter. The interactivity should be very high and fast paced to warrant the caching algorithm to consider single AUs.

4.7.1 ES Identifier

The encoding for interactivity is stored in the BIFS which are conveyed in their own ES, which means that the next user will not lose the interactive option. The BIFS ES is usually very light and there is little need for caching it. As long as the encoding for interactivity identifies ES in the same manner as the proxy, branching will not disrupt the proxy from correctly identifying the ES. It is not guaranteed in the MPEG4 standard that this is the case. In my thesis test we shall assume that all interactivity is preserved in a correct manner in its ES, which is not cached. Furthermore that the ESes used in branching and merging are identified in the same manner and name scope as in the proxy. The MPEG4 standard specifies conditions for when the name scope of the ES ID change. Any content with storylines patched together from material with different original name scopes must rename the ES IDs for consistency within the file. The MPEG4 standard specifies that any branching point can reference content outside the current name scope. In my thesis we will assume that such content is not suitable for caching, and will not look further into this issue. That is, we assume that only content eligible for caching uses the current name scope.

4.8 Scenario

In this example scenario the communication between the end user's terminal, the proxy and the source server are assumed correct and working. The cache is assumed to be empty before the first movie is transmitted. The transmission of ESes can begin after the end user terminal receives the initialObjectDescriptor (IOD). The IOD specifies the name scope and the main ES that starts the multimedia content presentation. The timeline, the scene descriptions, object descriptions, the visual and the audio ES are a set of such ESes. The timeline, scene description and object description ESes are not eligible for caching as these are small. They could be cached, but could just as easily be transmitted since they need few resources.

The movie starts with three different branches in the storyline. Since the movie is available in several languages, and the sound effects are on another audio track than the music, this movie has a set of four media

content ES for every branch. The very first scene in the presentation is a branching point. The end user chooses one, and the four ESes that are referenced by that choice are transmitted from the source server, stored in the cache and forwarded to the end user terminal. After some time the three branches reach a merging point in the middle of the movie. The end user doesn't see this, but the proxy stops storing new scenes into the four running ES. After the merging point, four new ES make up the movie, and these are now transmitted, stored and forwarded by the proxy. In the end there are two branches, the end user chooses the happy ending, and once again the proxy stops storage of the four current ESes, and begins again with the four final ESes. After this end user is finished, the proxy cache now contains 12 segments, each being a whole ES. Together they make a whole movie from the beginning to the end. When the next end user starts transmission of the same movie, that user also receives the IOD which will provide the same interactivity as the first user saw. If the second end user chooses branches that is different from the ones chosen by the first user, the ESes that make up these branches must be retrieved from the source server.

After several users have made their choices through the branch structure, the movie is cached with the most popular path. Of course if the movie is very popular all the branches that are available might be cached. The cache replacement algorithm works on the granularity of segments. Some branches might be represented with all the ESes they consist of, whilst other might have only the beginning half of the visual ES, according to popularity and storage space.

4.9 Taxonomy of Objects

In this section we will discuss the taxonomy of aggregated objects. Any interactive presentation that consists of premade stored content have an internal structure. The various primitives used to make the presentation are included in larger objects which again may be included in yet larger objects.

4.9.1 Non aggregated objects

What constitutes a non aggregated object? In this field, the minimum object could be a single frame. In MPEG-2 coding even a single frame may be decomposed to several objects in the I B and P frames. A set of such frames called a group of pictures (GOP) could be a minimum object. Where this delineation between aggregated and non aggregated objects occur is important for real applications. In this thesis we will assume that the smallest possible non aggregated object is determined by the application the end user chooses to play the presentation in, and that the author of

the content intended the presentation to be played in that application. With respect to the MPEG-4 content descriptors, the smallest possible non aggregated objects in streaming content are the AUs. That is for media content, in the scene description, there might be objects such as nodes that could be refered to as non aggregated objects. Still the MPEG-4 standard specifies the AU as the smallest descriptor to which timeliness information is?attributed. If we need to identify a specific node, that node should be placed in an AU of its own.

4.9.2 Aggregated objects

Aggregated objects consists of several smaller objects, in a linear presentation this is not a problem. A linear presentation starts playback of the next aggregated object when the current is finished playing. Possibly modified by time shifting. In interactive content the end user might choose a path of playback in which every step consists of complete aggregated objects. If the end user is determined to not play a specific part of such an object, but rather moves along the selected path before the aggregated object currently playing has shown all its content. This means that some of the information that was passed on to the end user was not used, and this is detrimental to the efficiency of any delivery network. Any aggregated object of premade stored content intended for use in an interactive presentation must not contain more information than the author of the content believes the end user is interested in. This may be a severe restriction for authors for content to many types of application.

4.9.3 Parsing the tree

A small object contained within a larger object might be accessed by parsing to it in the structure these objects are stored in. For presentations with a broad sort of audience, it would be beneficial for the author to create larger segments of content which end users might break up into user specific sizes as needed. This is possible in MPEG-4 content by parsing the tree of objects that make up the audio visual scenes. But only so far as the author has had this sort of use in mind. In branching video the author has multiple alternative choices for the end users, and expects that each branch between the branching and merging points are played in full. If these branches are to long, end users might grow impatient and shift the playback point.

4.10 Caching Branching Video

Using partial caching on branching video is the method chosen in the next chapter. This means identifying the individual branches, making caching

decisions based on their popularity acording to the chosen algorithm. The identity of the branches is given by the ES_ID they are represented in. Since the design assumes that each branch is represented in a unique ES. The boundary of the object is then naturally the entire ES. This limitation is not large however, if several branches are included in one ES, techinques can be used to make identifiers for the branches in the interior of the ES. Such identifiers could be the AU index number, since an ES is made by AUs back to back. Another technique could be to store the beginning and ending absolute file offset of the ES.

4.11 Segment Caching

The extent caching of branching video is both similar to and different from segment caching of the timeline. In segment caching the linear movies are divided into several segments, not just a prefix and a suffix. There are several ways to recognize the beginning and end of such a segment. A major similarity is the way the timeline of the narrative is broken into roughly equal size bits, and that caching candidate replacements can be made on them. In the segment scheme however, it is assumed that the most popular part of any movie is the beginning and that the likelyhood for reuse decline with the clock as the timeline passes. This may not necessarily be true for an interactive presentation, there might be links from the very start of the presentation to objects late in what would linearily be interpreted as the timeline. Since the composition of the narrative is left to the end user, the objects may appear in for the proxy random positions in the timeline. This means that there are no way the algorithm can calculate the expected chance of reuse for a segment based on the timeline. It is not sufficient to count the number of links to an object, or any other numeric quality such links might have, since the chance of any end user actually navigating that link and requesting the referred object is based upon human aesthetic taste or the informational content of that object, both of which are impossible for any algorithm to calculate.

4.11.1 Segments versus Branches

Segments of an elementary stream may be regarded as similar to branches in a branching video. The main difference is that the segments are determined by the algorithm in the proxy according to some heuristics or mathematically chosen metric. Whilst the branches in the branching video are chosen as being individual objects by merit of narrative content as understood by the author. This means that the chance the author have of guessing the correct boundaries of a branch are rather larger than what the algorithm in the proxy has.

Furthermore, the branches may be jumped into and out of by low level interactivity, time shifting. In this respect the part of the branch actually being played by the end user will in every respect be a good candidate for a segment.

If the algorithm in the proxies chooses the segments based solely upon which seconds of the elementary streams are actually requested, that would be sufficient to cache interactive content in a consistent manner. However, the caching candidate replacement policy would then be on a detail level of singular frames, and this could entail significant overhead.

Alternatively GoPs or several AUs could be used as the minimum candidates, still the overhead of bookkeeping will be significant. If the author of the content creates enough branches, and refrain from putting semantically different content into the same ES, the concept of extent domain caching branching video will probably be the most efficient.

Chapter 5

Design

This chapter presents the proposed design for caching interactive premade stored content in servers, using MPEG-4 like descriptors, and partial caching in the extent domain.

5.1 Intention of the Design

This thesis considers how a system can be set up to cache interactive content, the scope of the thesis is limited to caching of branching video in MPEG-4. The analysis chapter 4 discussed the possibility of partial caching in the *extent* domain, see figure 5.2. This is interesting because partial caching central theme is limiting disk size, and one major characteristic about interactive content is the enlarged file size. Before we can consider partial caching, complete caching is necessarily a stepping stone.

Noninteractive content plays from beginning to end, barring low level user interactivity such as FF and Reverse, and communicates the author's prefabricated story to the viewer. Interactive content may also utilize prefabricated content, or may generate unique content for every user. It is intuitively not interesting to cache an enormous amount of unique content. Caching prefabricated content however may prove efficent, as this is already proven for noninteractive content.

The simple trick is to identify the objects that the users will request on the same level of granularity as the interactivity in the content provides. For instance, the medium level interactivity provided in Branching Video where a user chooses between three different, but prefabricated, endings for a movie. In this example, those three different endings may be regarded as three independent objects by the caching algorithm. If complete movies are cached the proxy must recognize the user feedback and stream the correct interactive object. If these objects are identified, they make fine candidates for partial caching. Since partial caching is said to be either in the time or in the quality domain, this type of partial caching can be called

extent of interactivity, or *extent* domain for short. Extent domain caching is orthogonal to temporal or quality domain caching, because both the other techniques can be applied to the interactive objects independently without regard to whether they are stand alone objects or interactive subparts in a Branching Video.

The primitives used to construct the compound objects must all be available, but the most popular ones should be cached. Whether a subobject is a dependent quality increasing elementary stream or a fully independent ES that provide an alternative narrative storyline, the proxy cache does not consider this as interesting information. Rather the proxy uses simple but powerful statistical algorithms to tag every object with information about their popularity. The drawback is of course that every object must be identified, its boundaries must be defined and the metainformation about its popularity must be stored in some structure in the proxy. For very large systems identification could be done with hashvalues. Alternatively it is left to future work to consider cost function algorithms. Figure 5.1 shows shuch a compound object.

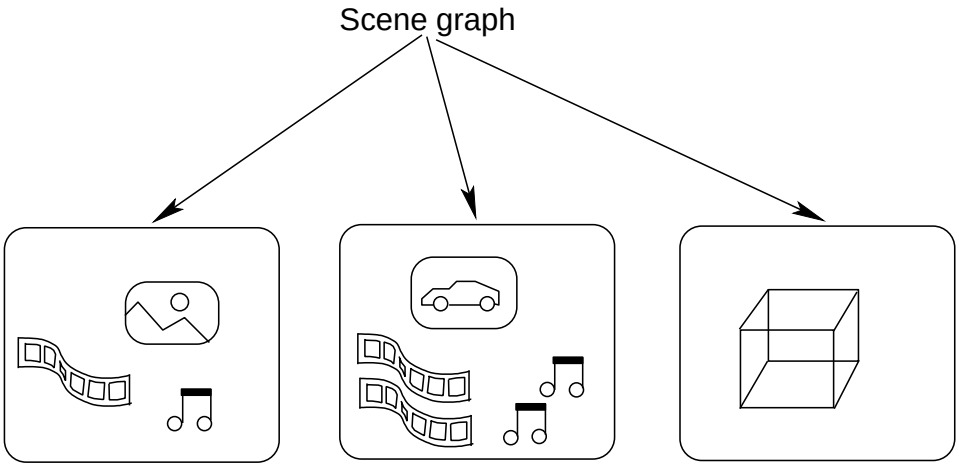

Figure 5.1: Media objects

5.2 Proposed Architecture

The proposed architecture is set within any type of CDN with intermittent proxy cache servers. The proxies may appear within the CDN or on the edges of the WAN close to the Clients. The proposed system has a minor part in the server and the client, and a major part in the proxy. The protocol of choice is MPEG-4. MPEG-4 and source servers in 5.2.6.

5.2.1 Partial Caching

The approach presented in this thesis is one of partial caching. As outlined in section 2.1.7, there are various versions of partial caching. In this thesis it is suggested that limiting the number of alternative elementary streams available for playback, is a way to reduce disk space requirements. One standard for interactive content is MPEG-4 ISO 14496 which defines a set of Descriptors used for referencing Audio Visual Objects. The building blocks of any multimedia presentation are elementary streams, also for MPEG-4. [59] propose to drop alternative encodings for ES as an adaptive way for quality domain partial caching. In MPEG-4 however, it is also possible from a human viewpoint to regard sets of ES as alternative representations of the same narrative content, even if they are disparate from a technical viewpoint. It is these alternative ES this thesis proposes to drop, in order to implement interactivity extent domain partial caching. If further restriction on size is necessary, partial caching in the quality or time domains may be used as well. It is possible to argue that the interactivity extent domain is actually in the *time domain*. However navigation through branching video is not supposed to pass through all the alternative branches. Partial caching of branching video does not limit the play time for beginning to end playback, rather the alternative branches will be slightly more expensive to get hold of. Time and extent caching can be combined for even more powerful file size reduction, see figure 5.3. And lastly adding quality caching on top gives the most powerful reduction, in figure 5.4.

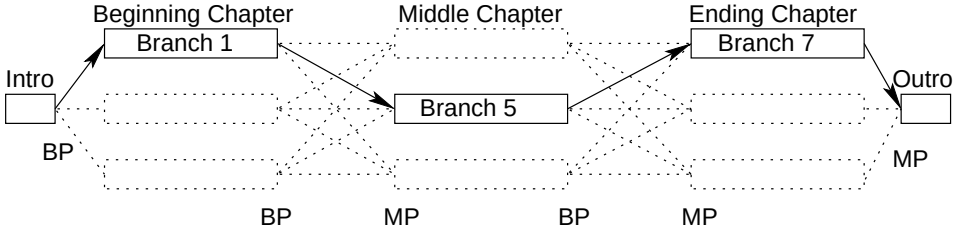

Figure 5.2: Extent caching

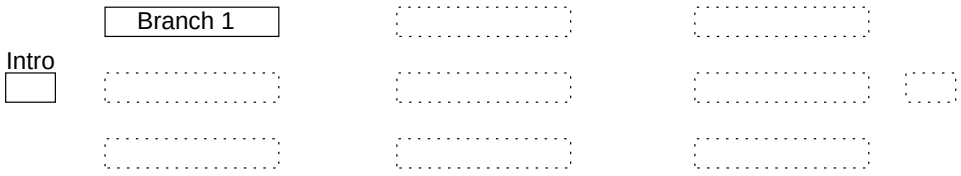

Figure 5.3: Extent and time

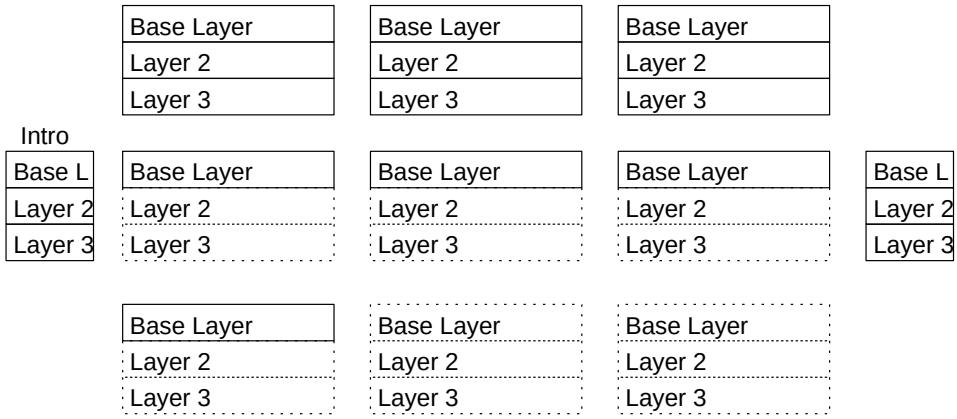

Figure 5.4: Extent, quality and time

5.2.2 Premise

To investigate the minimum amount of knowledge needed to consistently cache interactive multimedia scenes, the test setup described in 6.1 was used. The source server contains all the hinted interactive video content. The proxy is the intermediary between the clients and the source server, it is the proxy which constitutes the major work in this thesis. The clients connect to the proxy in the same way as to the source server since caching should be transparent. The 5.2.4 section describes the proposed architecture for consistent caching of interactive branching video when fully deployed.

5.2.3 Considerations

The central topic of this thesis is the minimum amount of knowledge needed to consistently cache interactive multimedia scenes, specifically in branching video encoded in MPEG-4. A proxy cache server in a CDN is a major system, and is beyond the scope of a cand. scient. thesis. The architecture proposed in the next section is a specific system targeted to show that it is possible to cache interactive branching video at all. The surrounding CDN and the RTP / RTSP communication, the network communication and other noncentral aspects are subdued.

The information necessary to cache each segment is the Branching Points and the Merging Points, if these are recognised the caching objects will be easily identified as the intermittent segments. A stream is played from beginning to end in the client terminal, unless it is halted by low level user interactivity, which is not a central topic. The scenes which provide the step into medium level interactivity are the Branching Point scenes. In these the user is presented with a choice of different paths forward, this

is enabled in MPEG-4 by the Scene Description Nodes that refer to new Elementary Streams, or a new position in a known Elementary Stream. SD Nodes regardless of type refer to ES only through Object Descriptors (OD), which in turn refer to the ES through ES Descriptors. The ES Descriptors (ESD) contain the encoding protocol specific information and the exact position of the ES within the file. One OD can refer to several alternative ESD, encoding the same content in scalable layers, or alternative protocol encodings. This is the alternative ES [59] uses in their partial caching scheme. The caching scheme in this thesis chooses between different ODs that refer to independent ES that are said by the author of the film to be alternative content for narrative purposes.

For a scene to be a random access point in an interactively navigated setting, they must be submitted by the Access Unit Command Frame: BIFS Command Replace Scene. Which is the only AU in the Scene Description Stream (sdsm) with the random access flag set to true. This is already specified in the MPEG-4 standard. Furthermore this thesis assumes that all the segments are identified with unique ES_ID within each film, which in turn should have unique IOD_ID. This assumption is not a severe restriction, because MPEG-4 specifies that all ES_ID in a .mp4 interchange format file are unique, though it does not apply for ES that appear elsewhere. Keeping the IOD_ID unique is up to the CDN administrators, and should not be to difficult. Failing that it is also possible to use the Source Servers URL together with the Films filename as a unique ID.

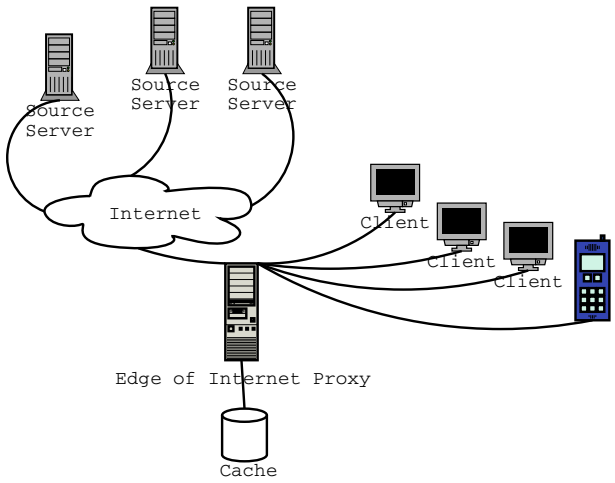

Figure 5.5: Architecture

5.2.4 Architecture

The architecture includes three major elements, the source servers, a proxy and a set of clients, illustration in figure 5.5. This is a simple autonomous caching system, still the ideas should translate well if deployed in hierarchical or cooperating systems. The server and proxy should both be able to accept clients using for instance RTSP/RTP and additionally communicate between themselves using TCP or inband information over RTSP.

In MPEG-4 the media data and other types of data is transported in Elementary Streams, each type of data in a new stream. The 'hint' track contains the framing that allows the streaming servers to serve the stream. The interactivity that MPEG-4 provides is described in the BIFS stream. For the part of Branching Video the interesting information are the sensor nodes, conditional programming nodes and the replace scene nodes. To simplify the system, this thesis assume that all such nodes only appear at the Branching and Merging Points (BP,MP) where the user may choose the next segment of the movie through the click of a mouse.

The BIFS Command Replace Scene is the only random access point in the BIFS stream, within these all the information that composes the interacitvity is stored as fields in objects, ultimately dereferencing the movie segments. The movie segments are stored in natural media Elementary Streams. This thesis assumes that each segment is stored in a complete ES of its own. This eliminates the need for searching through media ES, and, more importantly, eliminates the need to recompose the BIFS scene graph in the proxy to find the random access points that references the beginning and end of movie segments stacked within a large Elementary Stream. Saving those all to precious CPU cycles for other algorithms.

If a movie has its content stored in a more conservative way, the need for preparing the movies arise. Just as movies must be 'hinted' if the Quicktime or MPEG-4 formats are to be used for streaming. Movies where the interactivity is hidden within a large ES, needs to be broken up into its constituent branches. This is done by reading the BIFS stream of the movie, dereferencing the movie segments, and exporting the result into an MPEG-4 interchangeable format mp4 file. It is better to do this once, than having the proxy server do it for every single client! No information is lost in the process, the additional information is minimal, as an ES header is very small.

What work now remains for the proxy is to recognize that the various movie segments are semi independent objects, and to which movie they belong. This is done through bookkeeping and clever use of the namespace. Each movie originates from a server uniquely, this is kept in the namespace, additionally, within each movie, all the segments that make up the parts of the branches appear independently and with unique ES_ID numbers.

Everytime the movie goes through a BP or MP the ID of the ES must change.

This means that rather than treating the streaming movie from the source server as one bundle of streams that runs from beginning to end. The proxy considers the media content to be several objects for one movie, and the notion of a movie is abstracted to a 'set of small interlinked clips.' A server could provide several edits of a noninteractive movie as well. The only difference is that the BP or MP scenes does not provide interactivity but cut straight to the next segment.

5.2.5 Assumptions for the System

Using Branching Video in MPEG-4 is discussed in the analysis chapter, 4. This section sums up some of the assumptions that is necessary for the proposed architecture. Although the thesis tries to be complete, one is never guaranteed to have identified all antecedents.

Authors' responsibility

It is the responsibility of the author or the authoring software to ensure that all movie segments appear in their own ES with unique ID. This limits the use of Media Control Nodes in the SD. Which may point to a segment lying in the interior of an elementary stream. It is still possible to cache these segments, an additional algorithm to find and separate the segment would be needed. Since it is easy for the author to know where the presentation provides a BP, it is easier for the author to keep all segments in separate ES. Developing an algorithm that recognize segments in the interior of other ES is left for future work, this task is similar to enabling low level user interactivity within each branch, see 5.2.5.

Assuming that the interactivity provided in the movie is on the granularity of the movie segments. That is, all the interactivity are tied to BIFS Command Replace Scene. The BIFS Server Command is used to send a message from the user terminal to the server, enabling timedetermined actions or user responsive interaction. Still, the SD must be replaced in some way, and the command for SD replacement is; BIFS Command Replace Scene.

Assumptions

We assume that low level user interactivity, such as Fast Forward, Reverse, Stop and Pause does not give rise to additional complexity, see figure 5.6. This type of interactivity should be restricted to the BP. If users utilize low level interactivity within the segments between the BP, the efficiency of the caching will be reduced. This is of course a major issue, and will deteriorate

the performance of the system. The implementation will assume that the interactivity provided by the authors is sufficient for the users. The way to solve this is simply to add more chapters along the timeline of the films. Since some users are likely to use low level of interactivity regardless of how small the segments are, this sets an upper bound for the efficiency of this type of caching. To investigate exactly how large this restriction is is nontrivial and is left for future work.

Assuming that higher level user interactivity, is either nonexistent, or does not give rise to additional complexity. Higher level interactivity does not use pre-made stored content, and will need CPU time and other resources to be implemented, this will adversely affect the performance of the system. How much this will affect the system is determined by where the higher level of interactivity is implemented. Is it in the source server, and hence also in the proxy, or is it confined to the player terminal using predistributed primitives. These aspects will affect the performance of the system in a nontrivial manner, and investigation of this is left for future work.

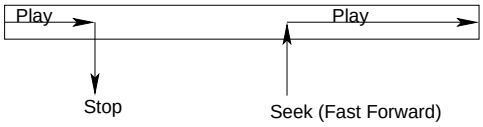

Figure 5.6: Low level interactivity

5.2.6 The Source Server

The source server is a normal source server which may stream interactive multimedia content. As explained in chapter 3, MPEG-4 does not frame the media data. This enables indirection of the streaming protocol, for this reason the server does not need to be MPEG-4 enabled, and MPEG-4 does not need to implement all types of protocols. This indirection is bridged by a hinter, which add all the necessary information in a separate hint track. A new type of hint format is necessary for every type of transport protocol used and for every encoding the MPEG-4 content is in. This means that the raw media data itself is not suitable frames, rather the media data is hinted by a separate hinter program to allow streaming, hence the term 'streamable' as opposed to 'streaming.' MPEG-4 specifies that the streaming server can be MPEG-4 agnostic, due to the indirection provided by the hint tracks. This is not a central topic in this thesis. The hint tracks are created by a hinter which is the bridge between MPEG-4 Systems and the specific encoding protocol used. Enabling MPEG-4 Systems to be protocol agnostic. The scheme in this thesis regards the structures in MPEG-4 Systems, and it is assumed that any ESD have the correct protocol

information and is referred to by a correct hint track.

If any problems arise when alternative ES are dropped, such as the need to rewrite the hint tracks, is beyond the scope of this thesis.

5.2.7 The Proxy

The proxy server accepts requests from the end users, and serves them as a streaming server if the requested content is in cache, exactly like any other proxy cache server. The elements that make up the proxy server are a Streamer, a Retriever, a Parser and a Cache Storage. The three elements Streamer, Retriever and Cache are as in any proxy. It is the Parser element that constitutes the novelty. It navigates the SD and OD structures and identifies the Branching Points and Merging Points. The exact way this is done may be arbitrarily complex, as interactive content can be made modularily into very complex structures. Mixing static and dynamic content, as well as changing the availability of objects temporarily. Even if it may be possible to make a Parser that can work with arbitrarily complex content, this thesis has limited the discussion to that of reasonably wellformed Branching Video. Extending the capabilities of the Parser to such content remains for future work, and would perhaps need extensive meta information, as well as hints from the authors as to whether the content is eligible for caching or not. The proposed Parser merely navigates the OD and SD looking for unique ES. These are then the caching candidates given to the Cache Storage. Since ES may be alternative to each other also in a technical sense, it is important to scan the OD streams to determine whether they appear in the same OD or not, as only those ES that appear in the same OD are technically alternative encodings. If the Parser disregards the SD and OD structures and only searches for unique ES, Quality Partial Caching similar to the one proposed in [59] will also apply, provided of course that such ESes appear at all.

The Replacement Algorithm

Several different approaches are possible, to cache complete Branches, to cache complete Chapters, to cache any ES, to cache all ES that appear in the scope of a single SD and so on. When SD Nodes such as Media Control Node refer to a repositioning in an ES, the segment that lies between the Branching Point and Merging Point does not correspond in a one to one fashion to a single ES. In such cases it is to expect that a caching algorithm that only takes whole ES as caching candidates will suffer inefficiencies. The smaller the segments are with respect to the length of the complete ES, the larger the inefficiencies will be. If there are several segments that appear as a result of Media Control Nodes, SD and OD parsing are more efficient. Still it is necessary to seek to the position in the ES given by the

Media Control Node, there is no field in the OD giving the offset into an ES, though there are several alternative fields that may serve such a purpose if reinterpreted to do so.

Singular AUs may also be considered as caching candidates if small segments and low level user interactivity is rampant. This is similar to Time Domain Partial Caching, and is not a central topic. If Time Domain caching is utilized alongside Extent caching the problem solves itself. This is because a time domain caching algorithm would have the needed capability to find caching candidates that are segments within the interior of antother ES, as explained in 5.2.5.

If all segments are complete ES, and no or little low level user interactivity jumps into or out of a running ES, it is sufficient to only parse the list of ES for candidates.

Whether to try to keep whole popular movies or branches in cache, or just to try to increase the byte hit ratio is a business decision. Particularly for movies that are highly advertised some CDN administrators might feel the need to keep a complete set of either the branches or segments with alternative encodings in the cache. Such considerations are not a central topic to this thesis, and it is assumed that the CDN administrators would want to increase the byte hit ratio.

5.2.8 The Clients

The clients connect to what they believe is the source server, but which really is the proxy. Therefore all interaction should be transparent. The clients may request any branch from any movie they have knowledge about, although it is not likely that they jump from within one movie to the interior of another, some services such as news, education, documentaries or single state games, might still warrant such behavior. The proposed architecture will assume that most users keep within the scope of a single Branching Video for every presentation. That those that don't accepts the increased latency of changing namescope, moreover that such changes are done by the Terminals without needing specific functionality from the Proxy, other than start, stop and reposition.

5.3 Fulfilling Requirements

In this section we will discuss how the proposed design fulfills the requirements put forward in the analysis in chapter 4.

5.3.1 Consistency

The minmum requirement for caching to be useful is that the caching candidates consistently are the most popular ones, and that they are

possible to retreive from the proxy by the clients with the correct id. Our design will split the BV into their consistuent objects, and make caching decisions based on the frequency of request of these objects. This will only work if the end users actually request the same type of objects as the proxy has split the BV into. Likewise the id namespace for the objects in the end users applications must be the same as for the proxy. This last requirement can be circumvented by considering for instance md5 hashsums of the caching candidates. However, in MPEG-4 the object hierarchy has a ready made system of object namespace, and it is somewhat easier to recognize ids.

Transparency

Consistency is preserved if the service is transparent. To test for transparency the service available to the end users should be the same as if the presentations where available directly on the source server.

Testing

To test for consistency, we only need to show that the caching candidates kept in the cache are reused by way of user requests. That is, if we observe cache hits at all, at least some consistency has been shown. Additionally, it is interesting to test whether the objects transmitted from the proxy to the end users, are of the same type that the users wanted. If they are smaller in size, and are in the interior of the object that the users actually requested, then the cache hit is not a good cache hit. Also, if the object transmitted is larger than the object requested by the end users, the actual efficiency of the delivery network will be lower. To test for all these things in one go, we introduce the metric consumed byte hit ratio. More on this metric in 6.3.

Narrative

The proposed architecture is designed to preserve the interactivity, even if the extent of it is limited in the cache, the system can still retrieve cache misses from the source server. It is also important from a feasibility point of view to ensure that the narration of the content is preserved, as the authors of such content will want their work to be the same, whether it passes through a proxy or not. The narration of the content has been preserved if the interactive choices of the end users remain the same even though they request constituent objects from a proxy, rather than from a source server. That is, the rules of how to put together the building blocks into a complete narrative, must remain the same. This is true if the end users play the presentation in the intended application, and if the repository of building blocks is available with respect to the rules of the narrative in the

same manner as on the source server. The repository of building blocks is fully available, possibly with difference in latency for objects of varying popularity. The rules of narrative interactivity are given by the application and the MPEG-4 descriptors. This means that if we guarantee that the MPEG-4 descriptors are preserved, the narrative is also preserved. We can only assume that the end users use the correct application, and if they don't that is beyond the scope of this thesis.

5.3.2 Performance

Proxy caching systems are usually introduced to improve performance, which may be speed of delivery, quality of content of sheer number of simultaneous end users. Additionally they might add functionality. In the proposed design, the proxy uses extent caching of the interactivity, possibly together with other forms of partial caching. Extent caching reduces disk size requirements of the presentations, and also splits presentations into easier to deliver sized objects. Though this thesis will regard the minimum requirements rather than performance issues, it should be a minimum requirement that the caching system does not introduce more overhead than performance gain.

Overhead

The overhead introduced comes from parsing the presentations and recognizing the constituent objects. Once this is done, the objects can be requested by their absolute id from the end users. If it is not possible to leave the job of finding the absolute id to the end users' applications, the cache objects must be requested by relative id. Request by relative id will introduce slightly more overhead, as the proxy will have to reverse parse the presentation in some manner, to find the correct object. That is, either it must have a directory of all the orginal presentations as they appear on the source server, and then find the relative id of the cache object that has been requested. Or it must consider every cache object on disk, and use some method to discern if the object in question is the correct. This can be done, either by asking the source server if the object is in the interior of the presentation that the end user is currently playing, which would be a very time consuming method. Alternatively the proxy could store the caching candidates' relative ids in such a way that they appear as absolute ids. That is, nested namespaces. This thesis will assume that nested namespaces are possible, and that the ids of all the presentation in the work domain of the proxy are unique. Since there is no fully deployed system to compare with, it is difficult to test the size of the introduced overhead, further investigation of introduced overhead is beyond the scope of this thesis.

Latency and QoS

The main performance metric for a proxy cache in the multimedia world is the latency. By increasing byte hit ratio we hope to decrease latency. Since this thesis looks at this problem from a point of view of minimum amount of knowledge needed. Exact measurements of latency improvements are less interesting, additionally it is only a simulated system, and not fully deployed with human end users. Other QoS metrics are likewise not central to the topic of this thesis.

Chapter 6

Implementation

We present an implementation of the extent caching of branching video, with interactivity limited to complete elementary streams. This approach should also work if extended to consider smaller objects, such as partial ES or AUs, if global ids are not available they can be constructed using the unique path to the individual object. The assumption is that statistical algorithms provide efficient enough ways to determine which of the objects that should be kept in the cache. Cost function algorithms will have to consider human constructed interactive structures to determine likeliness for requests, whether this is feasible or not is left for future work.

6.1 Implemented Test Architecture

The primitives used to construct our compound objects are the elementary streams, such as they might appear in MPEG-4 or some similar encoding. Each film in the system is assumed to be divided in the length dimension into several chapters, such as presentation chapter, middle chapter, dramatic ending chapter, or something of this nature. In each chapter the user is thought to be presented with several alternative representations of the narrative in the form of branches. Each branch is then encoded in two dependent elementary streams, one for video and one for audio. Naturally, the branches could contain more ES, but we believe two suits the purpose for now.

MPEG-4 agnostic

In MPEG-4 a server is said to be MPEG-4 agnostic, in this implemenation the proxy and server must be able to identify the requested descriptors inside the Films. This can be done by constructing absolute identifiers from the available relative identifiers. The MPEG-4 standard guarantees that the descriptors have unique ids inside the context of each film. And if each

film can be identified uniquely, the absolute identification of any descriptor is assured. Rather than constructing an elaborate algorithm that combines the server id with the film id, and then constructs an id for each ES with the ES_ID which is unique in each film, we have given each object that appear in the system a global id. The central idea remains that; the objects must be identified.

Parsing

The User requests Branches from the server according to what is available in the scene description that is in the presentation at any given moment. How many branches that constitute a film is an important factor. More branches gives a greater extent, which users like, greater freedom of choice, but also greater file sizes. The extent caching appears as being more efficient with increasing number of branches, as long as there are some bias to which is more popular, as this allows to drop the unpopular ones. Dropping unpopular branches, gives an appearance of the cachesize to be larger in relation to the total number of objects in the system, and hence it is more likely that we have a requested object in cache.

Chapters

An interactive branching video has several chapters, in each chapter, an interactive scene allows the user to choose between the alternative branches for that chapter, an author might reuse branches in several chapters.

The more chapters a film is presented in gives the segments smaller size, and makes the performance progressively more like time domain caching. The more branches each chapter has to choose between increases the extent of the film. It is this extent that interactivity extent caching tries to limit by dropping ES, see figure 6.1.

Example

The film is as in figure 6.1, a branching video. It has an introduction with the opening score and selection scene for the first branching point. In the test runs the Film objects are thought to include the IOD, the odsm and sdsm and any audio ,video or other content for the playback of the introduction and initial selection sequence. Subsequent branches are divided in audio and video ES. The content in the cache pertaining to one Film could look like figure 6.2. Every object that the proxy considers have a metaobject in the cache, and the metaobject tag whether the content is local in the cache or on the remote server. No actual content is being transmitted, only the descriptor objects implemented as C++ classes.

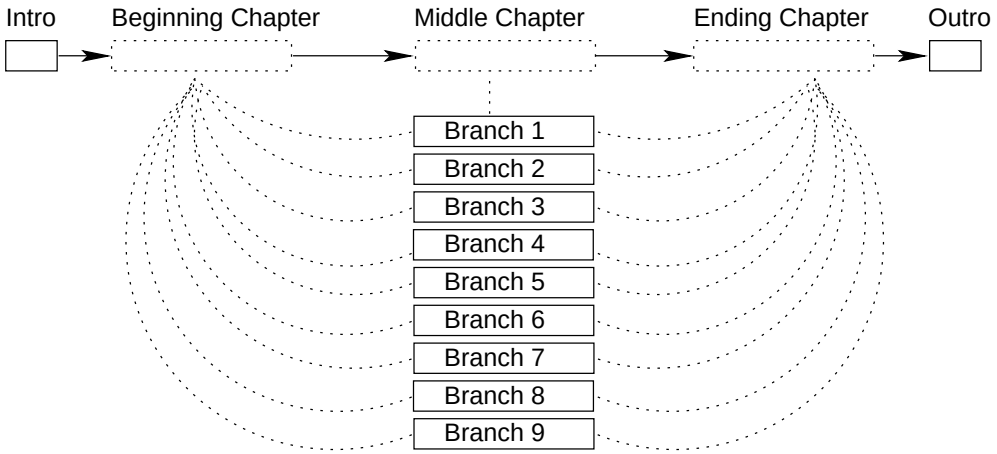

Figure 6.1: Freeform Branching Video

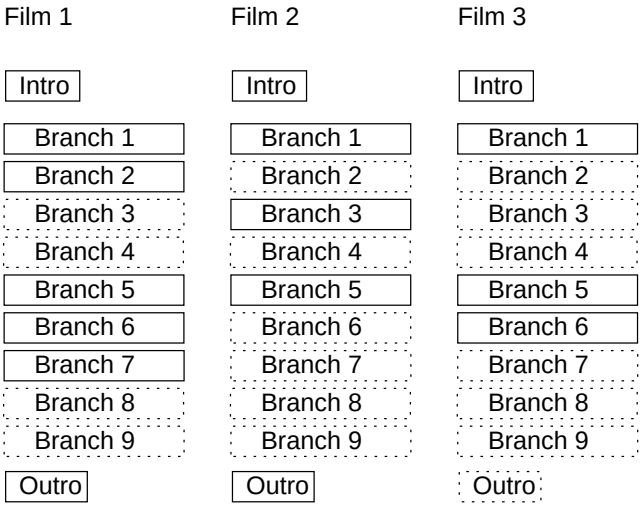

Figure 6.2: Cache Record

6.2 Disk Usage

The Proxy uses the Storage Element as a cache, and has one for the film meta information and one for the actual caching candidates. The meta information storage is not very big, at least in this implementation. The storage of caching candidates use many GB, but this is simulated, each cache candidate is a small meta information object with a field that says how many MB it is supposed to need. Their size would be on the order of 10 to 100 mb. A complete film would need from one to several GB, depending on whether just one path, or the entire extent of the film is cached.

6.3 Parameters

This section will present the parameters by which the caching scheme may be measured. The first issue that has to resolved is whether the caching scheme provides a transparent service into the interactive presentation. If the users are able to request the objects that are internal to the presentation through the cache this is said to be fulfilled. The caching candidates must be reused by the users, and not just stored in the cache and never accessed again. If the candidates score cache hits, this is the minimum requirement for any caching proxy. The amount of network resources used between the source servers and the end users will be reduced in accordance with the byte hit ratio in the proxy. The latency of the data transmission to the end users will be reduced for every object the end users can request from the proxy and not from the source server. Since startup time is important from a human perspective, the initial segment of any presentation should be worth more in the cache. The Film objects are thought to contain the initial selection scene, and since the requests start by requesting a film, this is fulfilled implicitly. The byte hit ratio also denotes the amount of data with a reduced latency. For interactive presentations that use a freeform style of navigation, fig 6.1, there are no guarantees that any object will be used as the initial segment. The BV in the test setup has a small intro that is thought to be the opening parts of the presentation, where the producers list their names, the company has its logo, and so on. This intro ends with the first selection scene followed by one of the alternative branches as per users' choice.

6.3.1 Byte Hit Ratio

Since all the ES are thought to be of the same size, and that includes the initial Film object, the byte hit ratio is identical to the cache hit ratio. In a situation with a mix of smaller and larger ES, the byte hit ratio would of

course drop accordingly.

6.3.2 Consumed Byte Hit Ratio

In this implementation the User consumes all the requested ES in full. This is because we assume that the author of the content has provided a unique ES for every alternative branch. If several optional content descriptors share the same ES, not all of that ES would be consumed by the user. How much the consumed byte hit ratio drops is in relation to how many alternative storylines the author has put into the same ES. If the ES contains two equally large storylines, and the user only play one of them, the consumed byte hit ratio would be cut in half.

6.4 LRU and LFU

The algorithms used in the simulator for cache replacements were LRU and LFU. The first two figures show test runs were the y-axis denote increasing cache hit, and the x-axis either increasing cachesize 6.3 or increasing number of films 6.4. The last figure also has a plotting of Zipf distributed branch selection. The plot with Zipf distributed branches lie above the one with random selections.

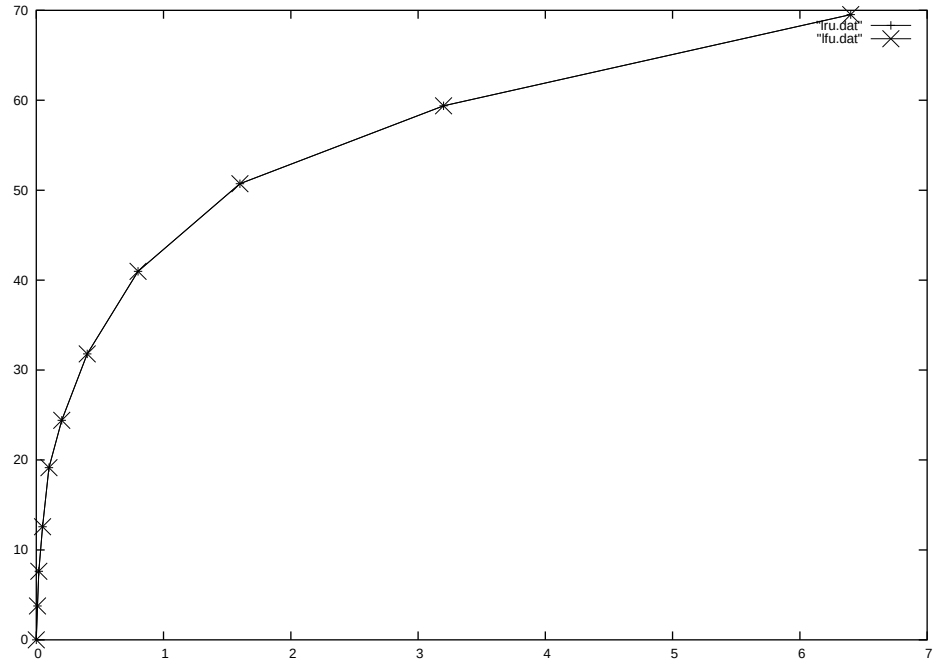

Figure 6.3: LRU and LFU vs cachesize

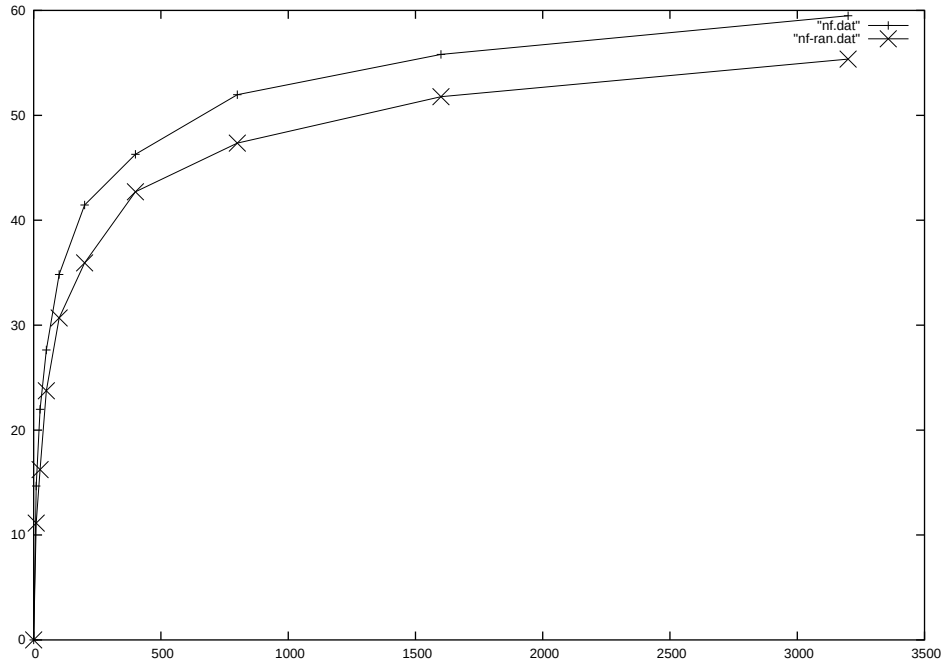

Figure 6.4: LRU vs films

6.5 Branches

Changing the number of branches per chapter, or whether the branches are selected randomly or by some Zipf distributed way, strongly affects the results. Figure 6.5 plot freeform selection of one in ten branches, vs five branches in five chapters and finally three branches in three chapters. As is obvious the more branches that can be selected away, the better performance, since we get a higher cachesize to interesting object ratio. All three series used zipf distributed selection of the branches. In figure 6.6 we plot random vs Zipf distributed selection of the branches. When the selection of the branches is random the performance of the extent caching declines, since no objects emerges as interesting to cache.

6.6 Interactivity inside ES

The MPEG-4 standard gives a set of object descriptors and content is included in these by file offset pointers into the media content files. Not all of these objects are possible to access directly in VoD systems. Particularly since they are hidden inside the scene description and object description stream and the terminals must know MPEG-4 to extract them and gain access to the content they represent. Scene Descriptions include SD-Nodes

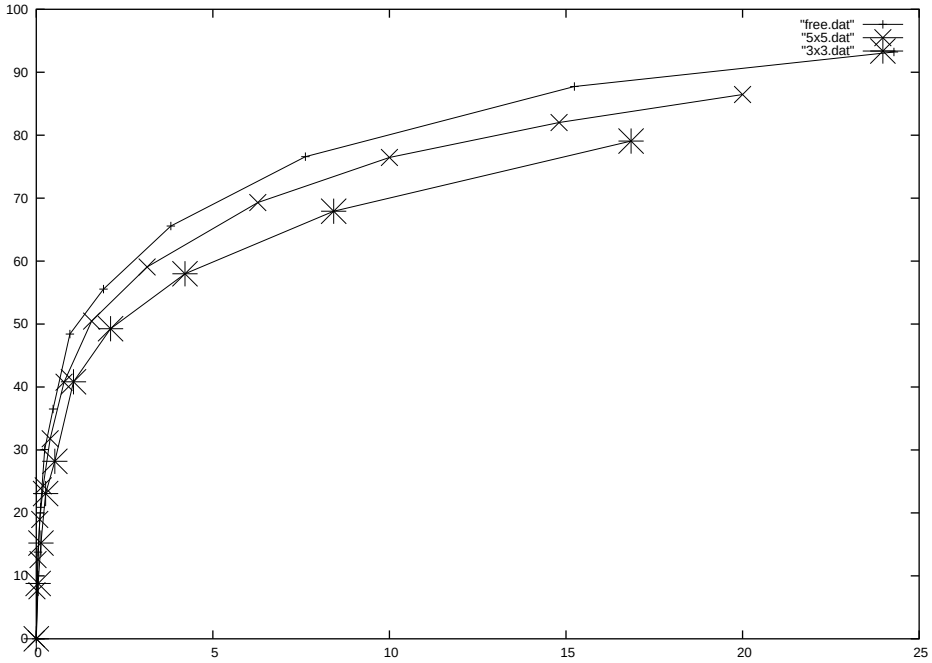

Figure 6.5: freeform, 5x5, 3x3

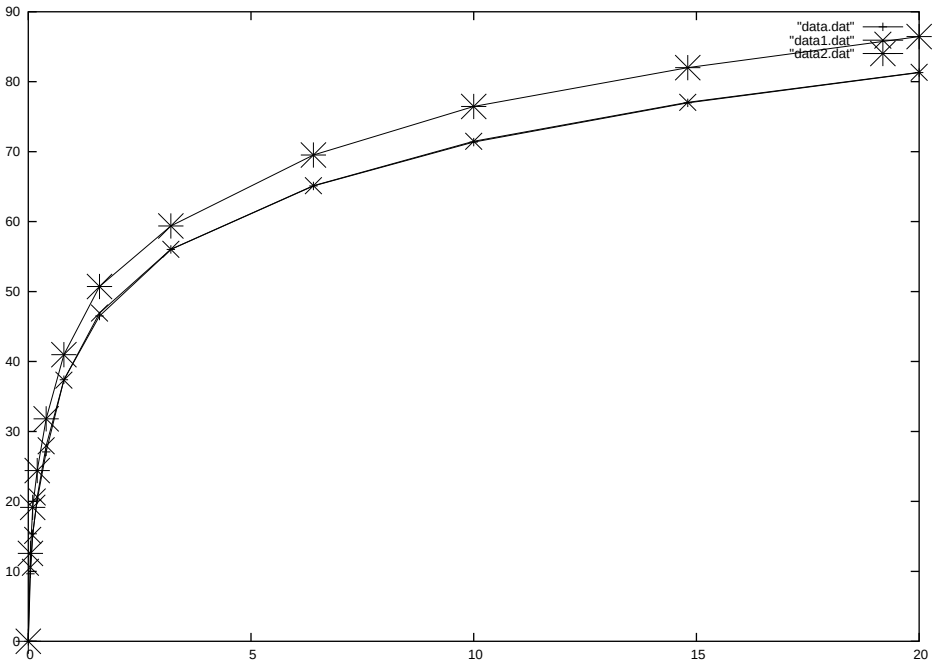

Figure 6.6: random vs Zipf branch

as fields, some of these can represent alternative content, this makes it possible to construct ES in which every scene includes branching points. Since caching can only work if the objects can be identified, this means that either we are forced to cache the entire ES, even if large parts of it remain unconsumed. Alternatively, we can create ad hoc objects that point into the ES by file offset, and record which parts of the ES are popular. The task would be; recognize that the user has shifted to a different offset inside the ES, and recognize when the user shifts away again. The beginning of the offset correspond to the object's id, the shift away point denotes the boundary of the object. Since AUs are the smallest items that the MPEG-4 standard attributes timeliness information too, they are perfect candidates for the smallest segments to recognize. Tracking all the AUs with metaobjects can be onereous though. The implementation in this thesis relays on the author to keep whole ES as single storylines, and put alternatives in ES' of their own.

Chapter 7

Conclusion

This thesis' premise was to investigate the minimum amount of knowledge needed for a proxy caching server to consistently cache interactive branching video encoded in MPEG-4.

As is shown in the analysis in chapter 4, the design in chapter 5 and finally the results in chapter 6 the answer to the premise task is:

The proxy cache server must know the identity of the caching candidates, both absolute and relative to the structure they are a part of. Furthermore the caching candidates must have defined borders, which gives them specific granularity with respect to the complete presentation.

Partial caching in the extent domain has been shown to work in the simulator, in future work we hope that it can be shown to be efficient also with more realistic user request patterns.

We chose only statistical algorithms, LRU and LFU, because no additional complexity was needed to simply demonstrate that the concept worked. For a more realistic implementation it could be interesting to se whether algorithms taking other factors into acount work better.

7.1 Results Discussion

The work presented in this thesis concludes with a test implementation and a proposed design for full implementation.

7.1.1 Test Implementation

The test runs agree on LRU and LFU, and increase cachehit with increasing number of films and increasing cachesize with respect to total number of objects. The effect *extent* caching has on the result is to provide a smaller set of interesting objects. That is, the cache hit ratio, and all other ratios that are have the cache hit ratio as a factor, will be improved in the same

manner as when the cachesize increase with respect to the total number of objects.

7.1.2 Proposed Design

The proposed design to construct cache objects based on the granularity of the interactivity, make sure these objects were identifiable and had defined boundaries, should be feasible. In the test implementation the objects had a global id, constructed upon creation of the content. No such luxury should be expected in a real deployment. Still the necessary ids can be constructed from the relative ids of the films, their constituent ESs and the sequence number of the AUs.

Consumed BHR

In the implementation the ES had only one alternative scene each, and could be consumed in its entirety by the users. For content with several alternative scenes within an ES, the consumed byte hit ratio would be likely to drop. To increase the consumed byte hit ratio again, the proxy would have to recognise the alternative scenes within the ES. This can be done by creating ad hoc objects from the AUs that constitutes the alternative scenes. These ad hoc objects should be identifiable and have defined boundaries. Identities can be constructed from the sequence number of the AUs, available in the SL-PDU header as an optional field, and the ES_ID of the ES. ES_IDs are unique within each film, and we are already able to identify unique films. Defining the boundaries can either be done by considering each random acces AU as a unique entity and log statistics for it. Or from analysing where the users shift the playback point. Analysing shifting playback point allows us to dispense with onerous logging structures for the AUs. The length of an ad hoc objects in number of AUs could be determined by how many AUs a set of users play before shifting playback point.

7.2 Future Work

Caching of primitives used as building blocks for interactive multimedia works. However for the caching to succeed in reducing the amount of data that is transmitted from the source server to the end users, the caching candidates must be on the same scale of granularity as the primitives. The design could be used in conjunction with the adaptive approach for quality domain partial caching in [59].

7.2.1 Meta Algorithm

The amount of data transmitted across the network from source server to proxy, and from proxy to end users can be reduced if it is limited to the data the end users actually want. This can be done with an adaptive policy for choosing the caching candidates, according to the pattern of reuse and the granularity of the actual primitves. In figure 7.1 we see a small branching video and a freeform set of very small clips.

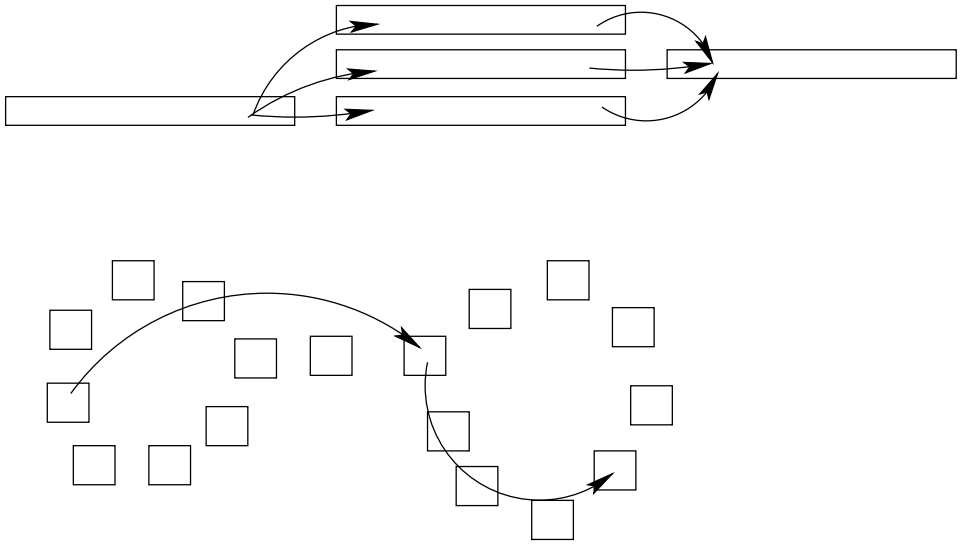

Figure 7.1: Type of interactivity

Cache object

Which type of cache object to choose as the primitive would depend on the type of interactivity. In Manovich principle of modularity, [35], any type of object can be used to build more complex compound objects, which in turn can be used to support the branching nature of the interactivity for the user, in the principle of variability. The meta algorithm would choose the same type of object as its caching object, ideally, as the author of the content intend to use as building blocks for compound objects. How to detect this using an efficient algorithm is quite interesting and not answered here.

Pattern of Use

Recognising the current pattern of use, such as branching video, freeform hypermedia, learning on demand or single state games, could be a source for optimisation. The meta algorithm could then choose the best fitting

replacement and admission policies based on heuristics. Such a futuristic proxy could be as in figure 7.2.

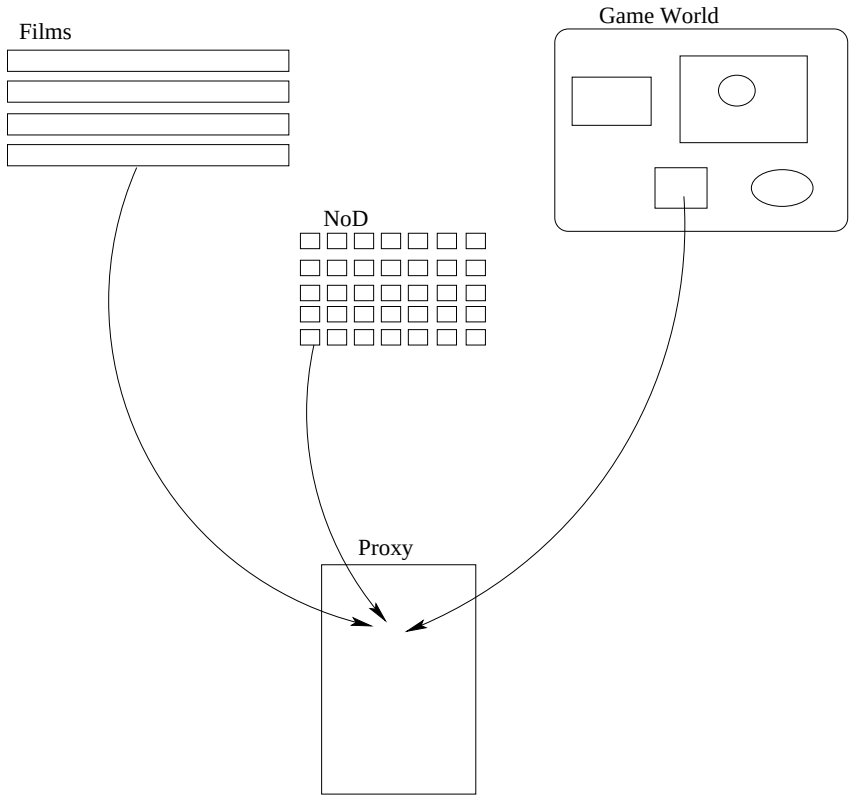

Figure 7.2: Proxy recoginses pattern

Motivation

Caching of interactive multimedia is likely to increase in importantance as all the multimedia services add new features; and as the existing services with a high degree of interactivity add multimedia content. If only the dedicated content delivery networks employ caching the services available in the www is likely to either become slower, or avoid adding interactive multimedia content.

Bibliography

[1] Espen J Aarseth. *Cybertext, perspectives on ergodic literature.* the john hopkins university press baltimore and london.

[2] S. Acharya and B. Smith. Middleman: A video caching proxy server. *Proceedings of the 10th international workshop on network and operating system support for digital audio and video*, jun 2000.

[3] E. Balafoutis, A. Panagakis, N. Laoutaris, and I. Stavrakis. The impact of replacement granularity on video caching. *Networking*, pages 214–225, may 2002.

[4] OpenGL Architecture Review Board. OpenGL. *http://www.opengl.org/about/overview.html*.

[5] E. Bommaiah, K. Gou, M. Hofmann, and S. Paul. Design and implementation of a caching system for streaming media over the internet. *IEEE, Real-time technology and application symposium (RTAS)*, 2000.

[6] S. H. Butcher. The Poetics by Aristotle, part XXIII plot in narrative. *http://www.scholars.nus.edu.sg/resources/poetics/23.html*.

[7] M. Cavazza, F. Charles, and S. J. Mead. Acm international conference proceeding series archive. *Proceedings of the second international conference on Entertainment computing table of contents*, pages 1 – 8, 2003.

[8] Chuck Clanton, Harry Marks, Janet Murray, Mary Flanagan, and Francine Arble. Interactive narrative: stepping into our own stories. pages 88–89, 1998.

[9] Coulouris, Dollimore, and Kindberg. *Distributed Systems Concepts and Designs*. Addison Wesley, third edition, 2001.

[10] Marc Davis. Theoretical foundations for experiential systems design. pages 45–52, 2003.

[11] H. Fahmi, M. Latif, S. Sedigh-Ali, A. Ghafoor, P.Liu, and L.H. Hsu. Proxy servers for scalable interactive video support. *IEEE Computer, 43(9):54-60*, sep 2001.

[12] Macromedia Flash. http://www.macromedia.com/software/flash.

[13] Davenport G and Murtaugh M. Automatist storyteller systems and the shifting sands of story. *IBM Systems Journal, v. 36 no. 3, 1997, p. 446 - 456.*, 1997.

[14] Kim Gee. The ergonomics of hypertext narrative: usability testing as a tool for evaluation and redesign. *ACM J. Comput. Doc.*, 25(1):3–16, 2001.

[15] Natalio Pincever Glorianna Davenport, Thomas Aguierre Smith. Cinematic primitives for multimedia. *IEEE Computer Graphics and Applications, vol. 11 issue 4, pg. 67 - 74.*, 1997.

[16] Y. Gou, S. Sen, and D. Townsley. Prefix caching assisted periodic broadcast for streaming popular videos. *Proceedings of ICC (International Conference on Communication)*, apr 2002.

[17] Griwodz, Bar, and Wolf. Long-term movie popularity models in video-on-demand systems. *ACM Multimedia*, 9-13, 1997.

[18] Carsten Griwodz. chapter 2, Doctor Thesis. *http://elib.tu-darmstadt.de/diss/000081/*, 2000.

[19] Extensible 3D (X3D) Task Group. http://www.web3d.org/x3d.html.

[20] Network Working Group. Rfc 3016 - rtp payload format for mpeg-4 audio/visual streams. *http://www.faqs.org/rfcs/rfc3016.html*.

[21] Stephane Gruber, Jennifer Rexford, and Andrea Basso. Design considerations for an rtsp-based prefix-caching proxy for multimedia streams. *Tech. Rep. 990907-01, AT&T Labs Research*, 1999.

[22] Pål Halvorsen, Carsten Griwodz, Ketil Lund, Vera Goebel, and Thomas Plagemann. Storage system support for multimedia applications. *Research Report No. 307*, jun 2003.

[23] ISO / IEC. Iso / iec 14496-1, mpeg-4. pages 15–16, 2001.

[24] ISO / IEC. Iso / iec 14496-1, mpeg-4. pages 72–73, 2001.

[25] ISO / IEC. Iso / iec 14496-1, mpeg-4. pages 229–230, 2001.

[26] IETF. Rtp real time transport protocol rfc 1889. *http://www.faqs.org/rfcs/rfc1889.html*.

[27] ECMA international. Standard ecma-262 ecmascript language specification. *http://www.ecma-international.org/publications/standards/ECMA-262.HTM*.

[28] ISMA. Internet streaming media alliance. *http://www.isma.tv*.

[29] ISO/IEC. Hytime standard, iso/iec 10744:1992. *http://www.ornl.gov/sgml/wg8/docs/n1920/*.

[30] ISO/IEC. Virtual reality markup language iso/iec 14772. *http://www.web3d.org/x3d/specifications /vrml/ISO_IEC_14772-All/part1/concepts.html*.

[31] Apple iTunes. http://www.apple.com/itunes/.

[32] Jens F Jensen. Interactivity. *www.nordicom.gu.se/reviewcontents/ ncom-review/ncomreview198/jensen.pdf*, 1998.

[33] J. Kangasharu, F. Hartano, M. Reisslein, and K. W. Ross. Distributing layered encoded video through caches. *Proceedings of IEEE INFOCOM*, apr 2001.

[34] Charles Krasic and Jonathan Walpole. Priority-progress streaming for quality-adaptive multimedia. *ACM Multimedia*, 2001.

[35] Manovich L. *The language of the new media*. The MIT press Cambridge Massachusetts, 2001.

[36] H. Latchman, C. Salzmann, D. Gillet, and J. Kim. Learning On Demand - A Hybrid Synchronous/Asynchronous Approach. *IEEE,http://www.ewh.ieee.org/soc/es/May2001/10/Begin.htm*.

[37] Francis C. Li, Anoop Gupta, Elizabeth Sanocki, Li wei He, and Yong Rui. Browsing digital video. pages 169–176, 2000.

[38] Chris Marin, Rob Myers, Jim Kent, Peter Broadwell, and SONY. Steerable media: Interactive television vis video synthesis. *ACM Multimedia 1-58113-339-1/01/01*, 2001.

[39] Z. Miao and A. Ortega. Proxy caching for efficient video services over the internet. *9th international packet video workshop*, apr 1999.

[40] Microsoft. Windows media player. *http://www.microsoft.com/windows/windowsmedia/download/default.asp*.

[41] Gene Miller, Greg Baber, and Mark Gilliland. News On-Demand for Multimedia Networks.

[42] Real Network. http://www.realnetworks.com/.

[43] nrk. http://www.nrk.no/.

[44] S. Paknikar, M. Kankanhalli, K.R. Ramakrishnan, S.H Srinivasan, and L.H Ngoh. A caching and streaming framework for multimedia. *Proceedings of ACM Multimedia*, pages 13–20, nov 2000.

[45] Dr. Pandian, A Rajesh, and Ram Mishra. Digital tv: Myth or reality. white paper. *http://www.wipro.com/insights/whitepaperdigitalTV.htm*.

[46] L L Peterson and B S Davie. *Computer Networks*. Morgan Kaufmann, 2000.

[47] Javier Pinto. Occurrences and narratives as constraints in the branching structure of the situation calculus. *Journal of Logic and Computation*, 8(6):777–808, 1998.

[48] Lydia Plowman, Rosemary Luckin, Diana Laurillard, Matthew Stratfold, and Josie Taylor. Designing multimedia for learning: narrative guidance and narrative construction. pages 310–317, 1999.

[49] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, 2003.

[50] Glidden R. Broadcastcl: Broadcasting compositing language. *http://www.broadcastcl.org*.

[51] M. Rabinovich and O. Spatscheck. *WEB Caching and Replication*.

[52] A. Rao, R. Lanphier, and H. Schulzrinne. Rtsp real time transport protocol rfc 1889. *http://www.cs.columbia.edu/ hgs/rtsp/*.

[53] R. Rejaie and J. Kangasharju. A quality adaptive multimedia proxy cache for internet streaming. *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, jun 2001.

[54] Reza Rejaie, Mark Handley, Haobo Yu, and Deborah Estrin. Proxy caching mechanism for multimedia playback streams in the internet. *Technical Report for Dept. of Comp. Sci., Univ. of Southern California*, 99-693, 1999.

[55] I S O Rob Koenen. Overview of the mpeg-4 standard. *http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm*, 2002.

[56] Stefan Saroiu, Krishna P. Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. Network behavior: An analysis of internet content delivery systems. *ACM SIGOPS Operating Systems Review*, 36 Issue SI, dec 2002.

[57] M. Sasabe, N. Wakamiya, M. Murata, and H. Miyahara. Proxy caching mechanisms with video quality adjustment. *Proceedings of SPIE Conference on Internet Multimedia Management Systems*, pages 276–284, aug 2001.

[58] Nitin Sawhney, David Balcom, and Ian Smith. Hypercafe. *Seventh ACM Conference on Hypertext*, 1996.

[59] Peter Schojer, Laszlo Boszormenyi, Hermann Hellwagner, Bernhard Penz, and Stefan Podlipnig. Architecture of a quality based intelligent proxy (QBIX) for mpeg-4 videos. *ACM 1-58113-680-3/03/0005*, mar 2003.

[60] S. Sen, J. Rexford, and D. Townsley. Proxy prefix caching for multimedia streams. *Proceedings of IEEE INFOCOM'99*, pages 1310–1319, mar 1999.

[61] Frank Shipman, Elli Mylonas, and Kaj Groenback. http://www.eastgate.com/patterns. *Proceedings of Hypertext '98*, 2002.

[62] In Interactive Storytelling. Planning formalisms and authoring. *citeseer.ist.psu.edu/571128.html*.

[63] Andrew Tanenbaum and Maarten van Steen. Distributed systems. pages 15–16, 2002.

[64] Guy Vardi. Navigation scheme for interactive movies with linear narrative. In *Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots*, pages 131–132. ACM Press, 1999.

[65] W3C. Extensible markup language xml. *http://www.w3.org/TR/REC-xml*.

[66] W3C. Hyper Text Markup Language. *http://www.w3.org/*.

[67] W3C. Scalable graphics markup language. *http://www.w3.org*.

[68] W3C. Scalable vector graphics (svg). *http://www.w3.org/TR/SVG/*.

[69] W3C. Synchronized multimedia integration language (smil). *http://www.w3.org/TR/REC-smil/*.

[70] B. Wang, S. Sen, M. Adler, and D. Townsley. Optimal proxy cache allocation for efficient streaming media distribution. *IEEE INFOCOM*, jun 2002.

[71] Jia Wang. A survey of Web caching schemes for the Internet. *ACM Computer Communication Review*, 25(9):36–46, 1999.

[72] WebTV. http://www.webtv.com.

[73] WorldDAB. http://www.worlddab.org/.

[74] K.-L Wu, P.S.Yu, and J.L. Wolf. Segment based proxy caching of multimedia streams. *Proceedings of the tenth international www conference*, may 2001.

[75] www.nextgentel.no.

[76] R. Young. Creating interactive narrative structures: The potential for ai approaches. *citeseer.ist.psu.edu/young00creating.html*, 2000.

[77] Z.-L. Zhang, Y. Wang, D.H.C. Du, and D. Shu. Video staging: A proxy-server based approach to end to end video delivery over the wide area networks. *IEEE/ACM Transactions on networking, 8(4):429-442*, 2000.

[78] Michael Zink, Carsten Griwodz, Jens Schmitt, and Ralf Steinmetz. Exploiting the fair share to smoothly transport layered encoded video into proxy caches. *Multimedia Computing and Networking 2002*, 2002.

List of Figures

Abbreviations

AU Access Unit
AV Audio-visual
BIFS Binary Format for Scene
CM Composition Memory
CTS Composition Time Stamp
CU Composition Unit
DAI DMIF Application Interface (see ISO/IEC 14496-6)
DB Decoding Buffer
DTS Decoding Time Stamp
ES Elementary Stream
ESI Elementary Stream Interface
ESID Elementary Stream Identifier
FAP Facial Animation Parameters
FAPU FAP Units
FDP Facial Definition Parameters
FIG FAP Interpolation Graph
FIT FAP Interpolation Table
FMC FlexMux Channel
FMOD The floating point modulo (remainder) operator
sgn(fmod(x/y)) = sgn(x), and
abs(fmod(x/y)) < abs(y)
IP Intellectual Property
IPI Intellectual Property Identification
IPMP Intellectual Property Management and Protection
NCT Node Coding Tables
NDT Node Data Type
NINT Nearest INTeger value
OCI Object Content Information
OCR Object Clock Reference
OD Object Descriptor
ODID Object Descriptor Identifier
OTB Object Time Base
PLL Phase Locked Loop
QoS Quality of Service
SAOL Structured Audio Orchestra Language
SASL Structured Audio Score Language
SDL Syntactic Description Language
SDM Systems Decoder Model
SL Synchronization Layer
SL-Packet Synchronization Layer Packet
SPS SL-Packetized Stream
STB System Time Base

TTS Text-To-Speech
URL Universal Resource Locator
VOP Video Object Plane
VRML Virtual Reality Modeling Language

Appendix A

Source Code

A.1 Header Files

```
/***************************************************************************
                          lab.h  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Wed Jun 23 2004
    copyright            : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; version 2 of the License.               *
 *                                                                         *
 ***************************************************************************/

#ifndef LAB_H
#define LAB_H


/**Contains the simulation lab and offers an interface
  *for the main driver.
  *@author Simen Rekkedal
  */
#include "user.h"
#include "server.h"
#include "proxy.h"
#include "terminal.h"
#include <iostream>
#include <stdlib.h>
#include <string>
#include <vector>
```

```
using namespace std;

namespace Lab {
void setProxyAlg(Terminal::Algorithm iproxyAlg);

/* seeds the random number generator */
void initRandom();

void setWarming(int w);

/* divides the films into several sequential chapters */
void setChapters(int in);

/* divides the chapters into several alternative branches */
void setBranches(int in);

/* determines how many users request media concurrently */
void setUsers(int in);

/* how many films are on the menu */
void setFilms(int in);

/* determines the size of the proxy in simulated mb */
void setProxySize(int ps);

/* the length of the simulation in simulated days */
void setSimulationDays(int in);

/* determines the number of AUs in the ESs */
void setESsize(int in);

/* creates the source server with the complete films */
void makeServer();

/* creates "numFilms" films, with branches and chapters as set */
void makeFilms();

/* creates "numUsers" user objects */
void makeUsers();

/* creates the simulated proxy object */
void makeProxy();

/* must be complete setup before start */
void startSimulation();

};

#endif
/*************************************************************************
                        media.h - Will investigate the consistency
                        of the caching of interactive multimedia objects
                        in a simulated proxy server, using films with
                        MPEG-4 like descriptors of the media objects.
```

```
                                                                        90
    begin                : Mon Jun 28 2004
    copyright    : © 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

#ifndef MEDIA_H
#define MEDIA_H
#include <string>
#include <stdlib.h>
#include <vector>                                                       100
#include <iterator>
#include <iostream>

using namespace std;

/**This namespace defines the objects that symbolize media content, or handles
 * to such objects. All necessary interfaces are defined here.
 */
namespace Media {
                                                                        110
class Media_Base{
public:
   virtual int getSize()=0;
   virtual string media_type()=0;
   virtual void parse()=0;
   virtual ~Media_Base();

};

/**Handle for Media_Base objects, used to pass around media. AS IF :(     120
 *
 */
class Media_Handle{
public:
   void parse();
   int getID();
   Media_Base* getRep();
   void setID(int i);
   int init(Media_Base* inrep);
   Media_Handle* lookup(int i);                                         130
   Media_Handle(Media_Base* inrep);
   Media_Handle();
   ~Media_Handle();

private:

   int id;
   Media_Base* rep;
};
                                                                        140

/**Determines the various types of Elementary Streams
 *
```

```
 */
enum stream_type {video,audio,odsm,sdsm};


/**Access Units
  *are of the type OD, SD, video data or audio data
  */

class AU : public Media_Base {
public:
  int getSize();
  /* used when building generic names, not in type checking !*/
  string media_type();
  void addMH(Media_Handle* mh);
  void parse();
  void init(int i,stream_type st);
  Media_Handle* getMH(int i);
  ~AU();
private:
  int size;
  int id;
  stream_type myType;
  vector<Media_Handle*>*branches;

};


/**Elementary Streams
  *@author Simen Rekkedal
  */

class ES : public Media_Base {
public:
  int getSize();
  /* used when building generic names, not in type checking !*/
  string media_type();
  stream_type get_type();
  AU* addSD(int i);
  AU* getSD(int i);
  void init(int insize,stream_type it);
  void parse();
  ~ES();
private:
  int size;
  /* the es may contain, od streams, sd streams, video or audio */
  stream_type myType;
  vector<Media_Handle*>* content;

};


/**The 'interactive branched video'
  *@author Simen Rekkedal
```

```
   */

class Film : public Media_Base  {                                    200
public:
   int getSize();

   /* used when building generic names, not in type checking !*/
   string media_type();
   void parse();
   int getNumChap();
   int getNumBran();
   void init(int numch,int numb,int ess);
   Media::ES* getSdsm();                                             210
   ~Film();

private:
   int size;
   int numChap;
   int numBranch;
   /* each film consists of a set of identifiable mediacontent */
   vector<Media_Handle*>* content;

};                                                                   220

/**Tags whether the object is brand new and not in cache, used to be in the
 * cache and was deleted, or if it is there now.
 */
enum CacheState {remote_new,remote_old,cached};

/**Handle class for the information needed to implement algorithms.
   *@author Simen Rekkedal
   */
class Cacheobject {                                                  230

public:
   int getSize();
   int getID();
   Media_Handle* getRep();
   Media_ID getMID();
   void setState(CacheState st);
   void setAlgc(int c);
   int getAlgc();
   CacheState getState();                                            240
   void addRep(Media_Handle* mh);

   Cacheobject(int i);
   Cacheobject(Media_Handle* inrep);
   Cacheobject();
         ~Cacheobject();

private:
   Media_ID mid;
   Media_Handle* rep;                                                250
   int accesses;
```

```
  int algc;
  int hits;
  int size;
  int id;
  CacheState myState;

};

};

#endif

/***************************************************************************
                          proxy.h  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Thu Jun 24 2004
    copyright            : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

#ifndef PROXY_H
#define PROXY_H

#include "terminal.h"
#include <vector>
#include <string>

namespace Terminal {
enum Algorithm {lru,lfu,fill};

/**The simulated proxy cache server.
  *@author Simen Rekkedal
  */

class Proxy : public Terminal_Base  {
public:
  void dumpList();
  void dumpPlot();

  /* incoming request for some media object */
  Request* serveRequest(Request* rin);

  /* create the proxy, keep virtual functions out of constructors */
  void init(int iwarm,int cachesize,int inumfilms,int ifilmsize,int numO,Algorithm a,Terminal_Base* is);

  virtual ~Proxy();

private:
  /* the link to the source server, which has the complete media */
  Terminal_Base* theServer;
```

```
/* use reserve() to set sizes */
vector<Media::Cacheobject*>* cache;

/* holds the index to the cacheobjects that are local */
vector<int>* cacheindex;

/* updates the cache */
void algAddCache(int i);

/* one of the algs selected in algaddcache */
void fillCache(int in);

/* one of the algs selected in algaddcache */
void lruCache(int in);

/* one of the algs selected in algaddcache */
void lfuCache(int in);

/* one of the algs selected in algaddcache */
void sizeCache(Request* rin);

/* enumeration */
Algorithm alg;

/* first time added to meta */
void addMetaCache(Request* rin);

/* gets noncached content, and asks if the caching alg wants it too */
Request* getMedia(Request* rin);

/* looks just in the cache, the request comes back with state set accordingly */
Request* findCache(Request* rin);

void algHit(int in);

void lruAccess(int in);

void lfuAccess(int in);

int warming;
int cacheSize;
int metaSize;
int numHits;
int numAccess;
int numFilms;
int filmSize;

/* counts how many objects have status Media::cached in the cache */
int count;

/* used as timestamp by lru */
int timestamp;
```

```
};

};

#endif
/***************************************************************************
                          request.h  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Mon Jun 28 2004
    copyright            : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; version 2 of the License.               *
 *                                                                         *
 ***************************************************************************/

#ifndef REQUEST_H
#define REQUEST_H

#include "media.h"

namespace Terminal {
enum ReqState{noop,id_only,found,sub};

/**An action class that represents the necessary information for a request
  *between terminals conserning media.
  *@author Simen Rekkedal
  */

class Request {
public:
  int getID();

  const ReqState getState();

  Media::Media_Handle* getRep();

  void addRep(Media::Media_Handle*);

  void setFound();

  void cacheMe();

  bool getTag();

  void setState(ReqState rs);
```

```
  bool foundState();

  Media::Media_ID* getMID();

  /* used as dummy, eg when User answers other terminals*/
  Request();

  /* used by user to make the first request using just the int id */
  Request(int i);

  /* used when the object can be refered to directly by some means */
  Request(Media::Media_ID* mid);

  Request(Media::Media_Handle* irep);

  Request(const Request&);

  Request::Request& operator=(const Request& old);

  ~Request();


private:
  /* when the request has got hold of the wanted media */
  Media::Media_Handle* reqMediaH;

  /*used to get hold of the media by name */
  Media::Media_ID* reqMediaID;

  /* holds the current status of this request, used in consistency checking */
  ReqState myState;

  /* holds the global id for the media_handle we want */
  int id;

  /* tags whether this requests object is interesting to cache or not */
  bool cacheTag;
};

};
#endif

/***************************************************************************
                          server.h  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Mon Jun 28 2004
    copyright            : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/
```

```
/*************************************************************************
 *                                                                       *
 *   This program is free software; you can redistribute it and/or modify *
 *   it under the terms of the GNU General Public License as published by *
 *   the Free Software Foundation; version 2 of the License.             *
 *                                                                       *
 *************************************************************************/

#ifndef SERVER_H
#define SERVER_H

#include "terminal.h"
#include <vector>
#include <iostream>

namespace Terminal {
/**The simulated source server, that has the films in complete versions.
  *@author Simen Rekkedal
  */

class Server : public Terminal_Base {
public:
  /* creates nf films, with nb branches and nc chapters as set */
  vector<int> makeFilms(int nf,int nc,int nb,int ess);

  int getNumObj();

  void dumpList();

  Request* serveRequest(Request* rin);

  Server();

  ~Server();
private:
/* stores the created film objects that this simulation uses */
vector<Media::Media_Handle>* filmList;

/* finds the media in the filmlist */
Request* lookup(Request* rin);

/* counts how many objects there are in total, which is identical to the ids */
int numObj;

};

};
#endif
/*************************************************************************
                          terminal.h  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
```

```
                          ------------------
    begin                : Wed Jun 23 2004
    copyright  : © 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; version 2 of the License.               *
 *                                                                         *
 ***************************************************************************/

#ifndef TERMINAL_H
#define TERMINAL_H

#include "media.h"
#include "request.h"

namespace Terminal {

/**Base class for Proxy, Server and User.
  *@author Simen Rekkedal
  */

class Terminal_Base {
public:
  virtual Request* serveRequest(Request* rin)=0;
        virtual ~Terminal_Base();
};

};
#endif
/***************************************************************************
                          user.h  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                          ------------------
    begin                : Mon Jun 28 2004
    copyright  : © 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

#ifndef USER_H
#define USER_H

#include <terminal.h>

namespace Terminal {

/**The simulated user that will request interactive media content from the servers.
  *@author Simen Rekkedal
```

```
  */

class User : public Terminal_Base {
public:
  /* not used    */                                                    580
  Request* serveRequest(Request* rin);

  /* called once, to get the user requests started */
  void start(int nu,vector<int>imenu,Terminal_Base* ip,int numDays,int nb);

  void printResults();

  User();
        ~User();
private:                                                               590
  /* going out by name, returning with object */
  Request* getMedia(Request* rin);

/* uses Terminal::zipfnums to store the pop distribution, 0.9 and 1.1, longtailed */
  void zipfdistribute();

  /* the user 'play' one film with this function */
  void play();

  /* parses the subobjects in mimic of user interactive play */        600
  void parser(Request* r);

  void consume(Request* rin);

  int chooseLength(int mint);

  int chooseBranch(int max);

  int numComplete;
                                                                       610
  int numBranch;
  int numDays;
};

};
#endif
#ifndef ZIPF_H
#define ZIPF_H

#include <sys/types.h>                                                 620

extern double* zipftable;
extern double* branchtable;

void init_zipf_branch( size_t moviect);
void init_zipftable_probability( size_t moviect );
void init_zipftable_density( size_t moviect );
void uninit_zipftable( );
```

```
#endif                                                                  630
```

A.2 Source Code Files

```
/***************************************************************************
                          au.cpp  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Tue Jun 29 2004
    copyright            : © 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

#include "media.h"

using namespace Media;
void AU::parse(){
}

int AU::getSize(){
  /* an au will usually be optimised to some particular size,
   * we use this as default*/
  return 1;
}

void AU::init(int i,stream_type st){
  id = i;
  myType = st;
  branches = new vector<Media_Handle*>;
  branches->reserve(5);//dynamically resized if need more space

}

void AU::addMH(Media_Handle* mh){
  branches->push_back(mh);
}

Media_Handle* AU::getMH(int i){
  return branches->at(i);
}

string AU::media_type(){
  return string("au");
}


AU::~AU(){

}


/***************************************************************************
```

```
                        cacheobject.cpp  -  Will investigate the consistency
                        of the caching of interactive multimedia objects
                        in a simulated proxy server, using films with
                        MPEG-4 like descriptors of the media objects.
                        ---------------
    begin                : Mon Jun 28 2004
    copyright      : © 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; version 2 of the License.               *
 *                                                                         *
 ***************************************************************************/

#include "media.h"
using namespace Media;
/* enum CacheState {remote_new,remote_old,cached}; in media.h */

int Cacheobject::getSize(){
  return size;
}

void Cacheobject::setState(CacheState st){
  myState = st;
}

CacheState Cacheobject::getState(){
  return myState;
}

void Cacheobject::addRep(Media_Handle* mh){
  rep = mh;
}

int Cacheobject::getID(){
  return id;
}

int Cacheobject::getAlgc(){
  return algc;
}

Media_ID Cacheobject::getMID(){
  return mid;
}

void Cacheobject::setAlgc(int c){
  algc = c;
}
```

```
/* !! */
Media_Handle* Cacheobject::getRep(){
  return rep;
}

Cacheobject::Cacheobject(int i){
  id = i;
  myState = remote_new;
}


Cacheobject::Cacheobject(Media_Handle* inrep){
  id = inrep->getID();
  rep = inrep;
  myState = remote_new;
}

Cacheobject::Cacheobject(){
  static int cid=0;
  cid++;
  id = cid;
  /* counts dummys only */
}

Cacheobject::~Cacheobject(){
}
/***************************************************************************
                          es.cpp  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Tue Jun 29 2004
    copyright      : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; version 2 of the License.               *
 *                                                                         *
 ***************************************************************************/


#include "media.h"
using namespace Media;
void ES::parse(){
  for(unsigned int i = 0 ; i<content->size();i++){
    content->at(i)->parse();
  }
}
```

```
int ES::getSize(){
  return size;
}

string ES::media_type(){
  return string("es");
}

/*sdsm es add au that is sd */
AU* ES::addSD(int i){
  AU* sd = new AU();
  sd->init(i,sdsm);
  content->push_back(new Media_Handle(sd));
  return sd;
}

AU* ES::getSD(int i){
  AU* sd = dynamic_cast<AU*>(content->at(i)->getRep());
  return sd;
}

void ES::init(int insize, stream_type it){
  myType = it;
  content = new vector<Media_Handle*>;
  content->reserve(insize);
  size = insize;

  if(it != sdsm){
    for(int i = 0;i<size;i++){
      AU* temp = new AU();
      content->push_back(new Media_Handle(temp));
    }
  }

}

stream_type ES::get_type(){
  return myType;
}

ES::~ES(){
}


/***************************************************************************
                          film.cpp  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Mon Jun 28 2004
    copyright            : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
```

```
 ***********************************************************************/

#include "media.h"
using namespace Media;
void Film::parse(){
  for(unsigned int i = 0 ; i<content->size();i++){
    content->at(i)->parse();
  }
}

int Film::getSize(){
  return size;
}

string Film::media_type(){
  return string("film");
}

int Film::getNumChap(){
  return numChap;
}

int Film::getNumBran(){
  return numBranch;
}


void Film::init(int numch,int numb,int ess){
  numChap = numch;
  numBranch = numb;
  size = numch*numb;
  content = new vector<Media_Handle*>;
  content->reserve(size+1);//all media + sdsm

  ES* sdsmPtr = new ES();
  sdsmPtr->init(numChap,sdsm);
  content->push_back(new Media_Handle(sdsmPtr));
  //debug cout << "sdsm with " << numChap << " chapters." << endl;

  for(int i=0;i<numChap;i++){
    /*Several alternative branches for each chapter*/
    AU* sd = sdsmPtr->addSD(i);
    //debug cout << "sdsm add new chapter " << i << endl;

    for(int j=0;j<numBranch;j++){

      /*the streams that interdepend,make a complete branch*/
      ES* tempV = new ES();
      tempV->init(ess,video);
      Media_Handle* pV = new Media_Handle(tempV);
      content->push_back(pV);
      sd->addMH(pV);

      ES* tempA = new ES();
```

```
        tempA->init(ess,audio);
        Media_Handle* pA = new Media_Handle(tempA);
        content->push_back(pA);
        sd->addMH(pA);

      }
    }
}

ES* Film::getSdsm(){
  /* we trust the init function to place the sdsm in position 0 */
  ES* sdsmPtr = dynamic_cast<ES*>(content->at(0)->getRep());
  return sdsmPtr;
}

Film::~Film(){

}
/***************************************************************************
                          lab.cpp  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Wed Jun 23 2004
    copyright            : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; version 2 of the License.               *
 *                                                                         *
 ***************************************************************************/

#include "lab.h"
using namespace Terminal;
using namespace Media;

namespace Lab{
/* the length of the simulation in simulated days */
int numDays=0;

/* how many films are on the menu */
int numFilms=0;

/* determines how many users request media concurrently */
int numUsers=0;

/* divides the films into several sequential chapters */
int numChapters=0;
```

```
/* divides the chapters into several alternative branches */
int numBranches=0;

/* determines the size of the proxy in simulated mb */
int proxySize=0;

/* determines the number of AUs in the ESs */
int esSize=0;

/* result of numbran and numchap *2 +1 */
int filmSize=0;

/* total number of objects in this simulation */
int numObjects =0;

/* how many request shall we warm the cache with before logging % */
int warming =0;

/* determines the algorithm used by the proxy in cache */
Algorithm proxyAlg;

Server* theServer;

Proxy* theProxy;

User* theUser;

/* holds the id for the films on the menu */
vector<int>menu;

/* how many days shall we warm the cache with before logging % */
void setWarming(int w){
  warming = w*numUsers;
}

/* determines the algorithm used by the proxy in cache */
void setProxyAlg(Algorithm iproxyAlg){
  proxyAlg = iproxyAlg;
}

/* divides the films into several sequential chapters */
void setChapters(int in){
  numChapters = in;
}

/* divides the chapters into several alternative branches */
void setBranches(int in){
  numBranches = in;
}

/* determines how many users request media concurrently */
void setUsers(int in){
  numUsers=in;
}
```

```
/* how many films are on the menu */
void setFilms(int in){
  numFilms=in;
}

/* determines the size of the proxy in simulated mb */
void setProxySize(int ps){
  proxySize=ps;
}

/* the length of the simulation in simulated days */
void setSimulationDays(int in){
  numDays = in;
}

/* determines the number of AUs in the ESs */
void setESsize(int in){
  esSize = in;
}

/* seeds the random number generator */
void initRandom(){
  /** seeding the dice.*/
  srand(static_cast<unsigned>(time(0)));
}

/* creates the source server with the complete films */
void makeServer(){
  cerr << "  Lab::makeServer()" << endl;
  theServer = new Server();
}

/* creates "numFilms" films, with branches and chapters */
void makeFilms(){
  cerr << "  Lab::makeFilms() " << numChapters << " " << numBranches << endl;
  menu = theServer->makeFilms(numFilms,numChapters,numBranches,esSize);
  filmSize = numChapters*numBranches*2;//we dont count the sdsm, sd's and films
  numObjects = theServer->getNumObj();
}

/* creates the simulated proxy object */
void makeProxy(){
  cerr << "  Lab::makeProxy()" << endl;
  theProxy = new Proxy();//careful with polymorphic constructors
  theProxy->init(warming,proxySize,numFilms,filmSize,numObjects,proxyAlg,theServer);
}

/* creates "numUsers" user objects */
void makeUsers(){
  cerr << "  Lab::makeUsers()" << endl;
  theUser = new User();
}
```

```
/* must be complete setup before start */
void startSimulation(){
  cerr << "  Lab::startSimulation()" << endl;
  theUser->start(numUsers,menu,theProxy,numDays,numBranches);
  theUser->printResults();
  //theServer->dumpList();
  theProxy->dumpPlot();
  theProxy->dumpList();

}

};// end of namespace lab.c
```

```
/***************************************************************************
                          main.cpp  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Wed Jun 23 2004
    copyright            : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; version 2 of the License.               *
 *                                                                         *
 ***************************************************************************/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <iostream>
#include <stdlib.h>
#include "lab.h"
using namespace std;
using namespace Lab;

int main(int argc, char *argv[])
{

  cerr << "Initializing the Lab" << endl;
  initRandom();
  if(argc == 1){
    setUsers(1000);//sequential requests only, not different user objects
    setSimulationDays(100);
    setProxySize(50);
    setProxyAlg(Terminal::lfu);
```

```
    setFilms(50);
    setChapters(5);
    setBranches(5);
    setESsize(0);
    setWarming(10);
  }else {
    setUsers(atoi(argv[1]));
    setSimulationDays(atoi(argv[2]));
    setProxySize(atoi(argv[3]));
    setProxyAlg(Terminal::Algorithm(atoi(argv[4])));
    setFilms(atoi(argv[5]));
    setChapters(atoi(argv[6]));
    setBranches(atoi(argv[7]));
    setESsize(atoi(argv[8]));
    setWarming(atoi(argv[9]));
  }
  makeServer();
  cerr << "server initialized" << endl;
  makeFilms();
  cerr << "films initialized" << endl;
  makeProxy();
  cerr << "proxy initialized" << endl;
  makeUsers();
  cerr << "Lab initialized" << endl;
  cerr << "Starting simulation" << endl;
  startSimulation();
  cerr << "Simulation finished!" << endl;
};
```

```
/***************************************************************************
                          media.cpp  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Wed Jun 23 2004
    copyright            : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; version 2 of the License.               *
 *                                                                         *
 ***************************************************************************/

#include "media.h"
using namespace Media;
```

```
Media_Base::~Media_Base(){
}
                                                                        540
/***************************************************************************
                          media_handle.cpp  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Thu Jul 1 2004
    copyright            : © 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/ 550

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; version 2 of the License.               *
 *                                                                         *
 ***************************************************************************/

#include <sstream>                                                      560
#include <iostream>
#include <string>
#include "media.h"

using namespace Media;
//public:
void Media_Handle::parse(){
  cout << "This is " << rep->media_type() << " id: " << id << endl;
  if(rep) rep->parse();
}                                                                       570

Media_Base* Media_Handle::getRep(){
  return rep;
}

int Media_Handle::getID(){
  return id;
}

int Media_Handle::init(Media_Base* inrep){                              580
  static int numMH=0;
  rep = inrep;
  id = numMH;
  /* debug cout << "Media_Handle::init:     " << numMH << " type "<<rep->media_type() << endl; */
  numMH++;
  return id;
}

Media_Handle* Media_Handle::lookup(int i){
  //debug cout << "Media_Handle::lookup () : for i= " << i << " This = " << id <<endl;  590
  if(i==id) return this;
```

```
  else return 0;
}

void Media_Handle::setID(int i){
  id = i;
}

Media_Handle::Media_Handle(Media_Base* inrep){
  init(inrep);
}

Media_Handle::Media_Handle(){
}

Media_Handle::~Media_Handle(){

}
/***************************************************************************
                          proxy.cpp  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Thu Jun 24 2004
    copyright            : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; version 2 of the License.               *
 *                                                                         *
 ***************************************************************************/

#include "proxy.h"
using namespace Terminal;
void Proxy::dumpList(){
  cerr << "Cachehits :" << numHits << endl;
  cerr << "Accesses : " << numAccess << endl;
  cerr << "Percentage hits: " << ((double(numHits)/double(numAccess)) * 100.0) << " %" << endl;
  //for(unsigned int i=0;i<cache->size();i++){
    //cout << cache->at(i)->getRep()->getRep()->media_type() << " ";
    //cout << cache->at(i)->getID();
    //cout << " " << cache->at(i)->getState() << endl;
  //}
  cerr << "Cachesize is : " << cacheSize << " number of objects." << endl;
  cerr << "Total number of objects is: " << numFilms*filmSize+numFilms << endl;
  cerr << "Percentage cachesize: " << ((double(cacheSize)/double(numFilms*filmSize+numFilms)) * 100.0) << "
  cerr << "The caching algorithm was: ";
  switch(alg){
    case lru: cerr << "lru " << endl; break;
    case lfu: cerr << "lfu " << endl; break;
```

```
    case fill: cerr << "fill " << endl; break;
  }
  cerr << "Accesses : " << numAccess << endl;
  cerr << "The 'count' variable is: " << count << endl;
}

void Proxy::dumpPlot(){
  cout << ((double(cacheSize)/double(numFilms*filmSize+numFilms)) * 100.0) << " ";
  cout << ((double(numHits)/double(numAccess)) * 100.0) << endl;
}

/* */
void Proxy::init(int iwarm,int icachesize,int inumfilms,int ifilmsize,int numO,Algorithm a,Terminal_Base* is){
  warming = iwarm;
  metaSize = numO;
  cache = new vector<Media::Cacheobject*>(metaSize);
  //cache->reserve(metaSize);

  cacheSize = icachesize;
  cacheindex = new vector<int>;
  cacheindex->reserve(cacheSize);

  alg = a;
  numHits=0;
  numAccess=0;
  theServer = is;
  numFilms = inumfilms;
  filmSize = ifilmsize;
  count = 0;
}

Request* Proxy::serveRequest(Request* rin){
  numAccess++;
  if(numAccess==warming){
    warming = 0;
    numAccess = 1;
    numHits=0;
  }
  rin = findCache(rin);
  if(!rin->foundState()) getMedia(rin);
  return rin;
}

Proxy::~Proxy(){
}

Request* Proxy::getMedia(Request* rin){
  rin = theServer->serveRequest(rin);
  addMetaCache(rin);//first timers are only added as metaobjects
  return rin;
}

//remote_new,remote_old,cached
Request* Proxy::findCache(Request* rin){
```

```
  int i = rin->getID();

  if(cache->at(i)){
    rin->addRep(cache->at(i)->getRep());

    switch(cache->at(i)->getState()){
      case Media::remote_new : algAddCache(i); break;//cache entry point !
      case Media::remote_old : algAddCache(i); break;//cache entry point !
      case Media::cached     : algHit(i);      break;//update popularity or timestamp
    }
  }
  return rin;
}

void Proxy::addMetaCache(Request* rin){
  cache->at(rin->getID()) = new Media::Cacheobject(rin->getRep());
  cache->at(rin->getID())->setState(Media::remote_new);

}

/* enum CacheState {remote_new,remote_old,cached}; in media.h */
void Proxy::algAddCache(int i){

  switch(alg) {
    case lru : lruCache(i);     break;
    case lfu : lfuCache(i);     break;
    case fill: fillCache(i);    break;
  }

}

void Proxy::fillCache(int in){
  if(count < cacheSize){
    cache->at(in)->setState(Media::cached);
    count++;
    cacheindex->push_back(in);



  }
}

void Proxy::lruCache(int in){
  if(count < cacheSize){
    cache->at(in)->setState(Media::cached);
    count++;
    cacheindex->push_back(in);

  }else{
    int oldestPtr = 0;
    int oldest = 0;
    for(int j=0;j<count;j++){
      int temp = cache->at(cacheindex->at(j))->getAlgc();
        if(temp < oldest){
```

```
            oldestPtr = j;
            oldest = temp;
          }
        }

      cache->at(cacheindex->at(oldestPtr))->setState(Media::remote_old);
      cacheindex->at(oldestPtr) = in;//switching which object is cached
      cache->at(in)->setState(Media::cached);
      lruAccess(in);
    }
}

void Proxy::lfuCache(int in){
  if(count < cacheSize){
    cache->at(in)->setState(Media::cached);
    count++;
    cacheindex->push_back(in);

  }else{
    int leastPtr = 0;
    int least = 0;
    for(int j=0;j<count;j++){
      int temp = cache->at(cacheindex->at(j))->getAlgc();
        if(temp < least){
          leastPtr = j;
          least = temp;
        }
      }

    cache->at(cacheindex->at(leastPtr))->setState(Media::remote_old);
    cacheindex->at(leastPtr) = in;//switching which object is cached
    cache->at(in)->setState(Media::cached);
    lfuAccess(in);
  }
}

void Proxy::algHit(int i){
  numHits++;
  switch(alg) {
    case lru : lruAccess(i);      break;
    case lfu : lfuAccess(i);      break;
    case fill: break;
  }
}

void Proxy::lruAccess(int i){
  cache->at(i)->setAlgc(timestamp);
}

void Proxy::lfuAccess(int i){
  cache->at(i)->setAlgc(cache->at(i)->getAlgc()+1);
}
```

```
/***************************************************************************
                          request.cpp  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Mon Jun 28 2004
    copyright            : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; version 2 of the License.               *
 *                                                                         *
 ***************************************************************************/

#include "request.h"
using namespace Terminal;
int Request::getID(){
  return id;
}

Media::Media_ID* Request::getMID(){
  return reqMediaID;
}

Media::Media_Handle* Request::getRep(){
  return reqMediaH;
}

void Request::addRep(Media::Media_Handle* mh){
  reqMediaH = mh;
  if(mh) myState = found;
}

void Request::setFound(){
  myState = found;
}

void Request::cacheMe(){
  cacheTag = true;
}

bool Request::getTag(){
  return cacheTag;
}

/* copy constructor */
Request::Request(const Request& old){
  cout << "Request copy constructor" << endl;
```

```
  myState = old.myState;
  reqMediaH = old.reqMediaH;
  reqMediaID = old.reqMediaID;
  id = old.id;
  cacheTag = old.cacheTag;
}

/* operator = */
Request& Request::operator=(const Request& old){
  cout << "Request operator = " << endl;
  if(&old != this){
  myState = old.myState;
  reqMediaH = old.reqMediaH;
  reqMediaID = old.reqMediaID;
  id = old.id;
  cacheTag = old.cacheTag;
  }
  return *this;
}

/* used only as dummy */
Request::Request(){
  myState = noop;
  reqMediaID = new Media::Media_ID();
  cacheTag = false;
}

/* outgoing request for media object */
Request::Request(int i){
  id = i;
  reqMediaID = new Media::Media_ID(i);
  myState = id_only;
  cacheTag = false;
}

/* wait a little with this one */
Request::Request(Media::Media_ID* mid){
  reqMediaID = mid;
  myState = found;
  cacheTag = false;
}


/* these are returned when the proxy or server has found the media */
Request::Request(Media::Media_Handle* irep){
  id = irep->getID();
  //cout << "request constructor with id: " << id << endl;
  reqMediaH = irep;
  myState = id_only;
  cacheTag = false;
}

Request::~Request(){
  cacheTag = false;
```

```
}

const ReqState Request::getState(){
  return myState;
}

/* used to determine if the request has been satisfied */
bool Request::foundState(){
  if(myState == found) return true;
  return false;
}

void Request::setState(ReqState rs){
  myState = rs;
}



/***************************************************************************
                          server.cpp  -  Will investigate the consistency
                          of the caching of interactive multimedia objects
                          in a simulated proxy server, using films with
                          MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Mon Jun 28 2004
    copyright            : (C) 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

#include "server.h"
using namespace Terminal;
vector<int> Server::makeFilms(int nf,int nc,int nb,int ess){
  /* id menu */
  vector<int>menu(nf);

  /* holds the complete films permanently */
  filmList = new vector<Media::Media_Handle>(nf);

  for(int i=0;i<nf;i++){
    Media::Film* temp = new Media::Film();
    temp->init(nc,nb,ess);
    filmList->at(i) = Media::Media_Handle();
    filmList->at(i).init(temp);
    menu[i]=filmList->at(i).getID();
    //cout << "film: " << filmList->at(i).getID() << endl;
  }

  numObj = filmList->at(nf-1).getID()+1;
  cerr << "Server::numObj : " << numObj << endl;
  return menu;
}

void Server::dumpList(){
  cout << "Server::dumpList() : list size =" << filmList->size() << endl;
```

```
  for(unsigned int i =0;i<filmList->size();i++){
    cout << filmList->at(i).getID() << endl;
  }
}

int Server::getNumObj(){
  return numObj;
}

Request* Server::serveRequest(Request* rin){
  //cout << "server::serveRequest()" << endl;
  return lookup(rin);
}

/* finds the media in the filmlist (contains media_handles) */
Request* Server::lookup(Request* rin){
  //debug cout << "Server::lookup() : " << rin->getID() << endl;
  Media::Media_Handle* temp;

  /* if this is a request for an ES, we have "cheated" a little and the
   * correct object is already at hand.
   */
  if(rin->getTag()){
    rin->setFound();
    //debug cout << "Server::lookup() Tag = cache " << endl;
    return rin;
  }

  for(unsigned int i=0;i<filmList->capacity();i++){

    temp = filmList->at(i).lookup(rin->getID());//reassigned to right pointer

    if(temp){
      //cout << "Server::lookup() found:" << temp->getID() << endl;
      //cout << "Server::lookup() found:" << temp << endl;
      rin->addRep(temp);
      rin->setFound();
      return rin;
    }
  }

  /* serious error, request for menu object not on the menu !! */
  cout << "Server::lookup() request for menu object not on the menu !!" << endl;
  exit(0);
  return rin;
}

Server::Server(){
}
Server::~Server(){
}
/*************************************************************************
                        terminal.cpp  -  Will investigate the consistency
```

```
                        of the caching of interactive multimedia objects
                        in a simulated proxy server, using films with
                        MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Wed Jun 23 2004
    copyright      : © 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no                                1030
 ***************************************************************************/

#include "terminal.h"
using namespace Terminal;

Terminal_Base::~Terminal_Base(){
}
/***************************************************************************
                        user.cpp   -  Will investigate the consistency
                        of the caching of interactive multimedia objects    1040
                        in a simulated proxy server, using films with
                        MPEG-4 like descriptors of the media objects.
                             -------------------
    begin                : Mon Jun 28 2004
    copyright      : © 2004 by Simen Rekkedal
    email                : simenre@ifi.uio.no
 ***************************************************************************/

/***************************************************************************
 *                                                                         *  1050
 *    This program is free software; you can redistribute it and/or modify *
 *    it under the terms of the GNU General Public License as published by *
 *    the Free Software Foundation; version 2 of the License.              *
 *                                                                         *
 ***************************************************************************/

#include "user.h"
#include "zipf.h"

namespace Terminal {                                                        1060
/* a few semi global variabels */
/*holds the zipf lottery numbers for the films according to pop distribution*/
  vector<int>zipfnums;

  /* holds the zipf numbers for the branches inside any film */
  vector<int>zipfbran;

  /* the films by id */
  vector<int>menu;
                                                                            1070
  int numUsers;
  int max_zipf;
  int numFilms;
  int consumed;

  Terminal_Base* theProxy;
```

```
};

using namespace Terminal;
//public:

/* not good for much in user, we don't use the user's inheritance to terminal much */
Request* User::serveRequest(Request* rin){
  return rin;
}

void User::printResults(){
  cerr << "Number of requests for films in total: " << numDays*numUsers << endl;
  cerr << "Number of requests for entire films: " << numComplete << endl;
  cerr << "Number of ES that were available for consumption : " << consumed << endl;

}

/* called once, to get the user requests started */
void User::start(int nu,vector<int>imenu,Terminal_Base* ip,int nd,int nb){
  numUsers = nu;
  numDays = nd;
  menu = imenu;
  numFilms = imenu.size();
  numComplete = 0;
  consumed = 0;
  zipfnums.reserve(numFilms);
  numBranch= nb;

  theProxy = ip;

  zipfdistribute();

  for(int j = 0;j<numDays;j++){
      for(int i = 0;i<numUsers;i++){
        play();
      }
    /* need to see this on screen, not in logfile */
    cerr << "Day : " << j << " of :" << numDays<<endl;//debug << ". NumComplete: " << numCo

  }
}

User::User(){
}

User::~User(){
}

//private:

/* going out by name, returning with object, requests Proxy for media */
Request* User::getMedia(Request* rin){
  return theProxy->serveRequest(rin);
}
```

```
/* the user play one film with this function */
void User::play(){

  double* p = std::lower_bound( zipftable,
                                &zipftable[numFilms],
                                drand48() );//supposed to exchange drand48 ! with a better random gen
  int zipf_distributed_entry = ( p - zipftable );

  parser(getMedia(new Request(menu[zipf_distributed_entry])));

}



/* parses the subobjects in mimic of user interactive play */
void User::parser(Request* rin){

  Media::Media_Handle* filmHandle = rin->getRep();

  Media::Film* filmPtr = dynamic_cast<Media::Film*>(filmHandle->getRep());
  if(filmPtr) {

    int numChap=filmPtr->getNumChap();
    int numBra=filmPtr->getNumBran();

    /* get hold of the Scene Description stream */
    Media::ES* sdsmPtr = filmPtr->getSdsm();

    for(int i = 0; i < numChap ; i++){
      /* Getting the Scene Description for this chapter */
      Media::AU* sd = sdsmPtr->getSD(i);

      int j = chooseBranch(numBra);

      Media::Media_Handle* branchVideo = sd->getMH(j);
      /* we already "have" the object, but will now ask the proxy for it */
      Request* reqV = new Request(branchVideo);
      reqV->cacheMe();

      consume(getMedia(reqV));

      Media::Media_Handle* branchAudio = sd->getMH(j+1);
      Request* reqA = new Request(branchAudio);
      reqA->cacheMe();

      consume(getMedia(reqA));

    }

  } else {cerr << "Not a Film in the Film ptr !" << endl;}
}
```

```
void User::consume(Request* rin){
  /* accepts Requests that contain the audio video ES, log consumption */
  Media::stream_type st = dynamic_cast<Media::ES*>(rin->getRep()->getRep())->get_type();
  switch(st){
    case Media::video:   consumed++; break;                                    1190
    case Media::audio:   consumed++; break;
    case Media::odsm:    cerr << "Wrong ES type!" <<endl; break;
    case Media::sdsm:    cerr << "Wrong ES type!" <<endl; break;
    default: cerr << "Casting error ! st == 0 !" <<endl;
  }
}

/* select random length */
int User::chooseLength(int mint){
                                                                               1200
  return mint;
}


/* the users will probably single out some branches as cooler than others */
int User::chooseBranch(int max){
  double* p = std::lower_bound( branchtable,
                                &branchtable[max],
                                drand48() );//supposed to exchange drand48 ! with a better random gen
  int zipf_distributed_entry = ( p - branchtable );                            1210

  return zipf_distributed_entry;
  //return int(drand48()*max);
}


/* longtailed */
void User::zipfdistribute(){
  init_zipftable_density( numFilms );
  init_zipf_branch( numBranch );                                               1220
}
```