uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Razib Iqbal

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

An Architecture for MPEG-21 Based Live Adaptation of H.264 Video

TITRE DE LA THÈSE / TITLE OF THESIS

Shervin Shirmohammadi

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Jiying Zhao

Dorina Petriu

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

An Architecture for MPEG-21 Based Live Adaptation of H.264 Video

by

Razib Iqbal

A thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements for the degree
Master in Computer Science

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-18428-8
Our file Notre référence
ISBN: 978-0-494-18428-8

An Architecture for MPEG-21 Based Live Adaptation of H.264 Video

Master of Computer Science Thesis
Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University of Ottawa

by

Razib Iqbal

April 2006

Abstract

Diversity of devices in both wired and wireless networks through which multimedia contents are desired to be accessed and interacted with has grown significantly. The driving concept of Universal Multimedia Access (UMA) depicts that instead of having multiple types of the same content, it is preferred to adapt the original content accordingly. Conventional transcoders adapt digital contents by decompression, adaptation, and recompression. An easy solution to adaptation is thus achieved by allocating some trusted intermediary nodes in between the media resource provider and the consumer to perform manifold decoding and encoding operations. But if cascaded transcoding can be avoided and adaptation can be performed in a compressed domain without decompression and re-compression, it will be surely a significant improvement. The objective of this thesis is to show that knowledge of the bitstream along with content structure in the form of MPEG-21 generic Bitstream Syntax Description (gBSD) enables the adaptation process to be easily deployable avoiding cascaded transcoding. A comprehensive study and an implementation technique of a dynamic schema for temporal adaptation of the H.264 pre-recorded and live video, conforming ISO/IEC MPEG-21 Digital Item Adaptation (DIA), is presented in this work. Adaptation is performed on demand directly from the bitstream and its gBSD. As a result, any MPEG-21 compliant host can adapt the stream without requiring the video codec.

Acknowledgements

On the eve of this succession, I express my gratitude to Almighty Allah who has bestowed upon me the strength and patience to successfully complete my research.

I am highly indebted to my supervisor Dr. Shervin Shirmohammadi. Dr. Shervin has provided me with an extraordinary opportunity to undertake research in the emerging field of multimedia adaptation. His mentorship during the course of this research has directed me to the way of innovation and accomplishment. I thank my supervisor for providing me with all necessary inputs, the facilities and, last but not least, his financial support. His encouragements were the driving force behind getting this project ready well ahead of time. I truly believe that without his support and guidance, this project would not have come into fruition at any point throughout my degree.

Appreciation also goes to Dr. Christopher Joslin, now Assistant Professor at Carleton University, for taking the time to clarify my understanding of concepts, and providing relevant suggestions during the implementation phase. His supportive role was also a motivation for this venture.

I gladly remember all my colleagues and well-wishers from the DISCOVER Lab, University of Ottawa - to whom I shall remain thankful always.

I must acknowledge Rizwana Alamgir, Ph.D. candidate, University of Ottawa for her support and editing assistance.

And finally, there is no way I can thank my parents and sister in the entirety of eternity. These are the people who have poured me a bounty of affection, and I have no words to express my gratitude towards them in any way.

Thank you.

Contents

List of Figures

List of Tables

List of Acronyms

AVC	-	Advanced Video Coding
BSDL	-	Bitstream Syntax Description Language
BSD	-	Bitstream Syntax Description
CIF	-	Common Intermediate Format
DI	-	Digital Item
DIA	-	Digital Item Adaptation
gBSD	-	generic Bitstream Syntax Description
ISO	-	International Standards Organization
IEC	-	International Engineering Consortium
IETF	-	Internet Engineering Task Force
ITU	-	International Telecommunication Union
MPEG	-	Motion Picture Expert Group
QCIF	-	Quarter Common Intermediate Format
SQCIF	-	Sub-Quarter Common Intermediate Format
STX	-	Streaming Transformations for XML
UMA	-	Universal Multimedia Access
URI	-	Uniform Resource Identifier
XML	-	Extensible Markup Language
XSL	-	Extensible Stylesheet Language
XSLT	-	Extensible Stylesheet Language Transformation

Chapter 1

Introduction

1.1 Motivation

An exploding usage of video information is a reality in current times, since we are broadcasting over cable, satellite and the internet, while storing in different media every moment. Consequently, different digital applications such as video-on-demand and multimedia streaming have become common buzz words. From the viewpoint of contents, multimedia contents are now available everywhere. Revolution of digital media introduces various types of representation format. Another phenomenon is the growth of the communication networks. The heterogeneous network provides a very convenient tool for content transmission, but its topmost limitation is the variation of available bandwidths. However, with advances in wireless communication, multimedia content transmission now becomes possible through the 3G and beyond 3G networks. As a whole, the diversity of devices in both wired and wireless networks through which multimedia contents are desired to be accessed and interacted with has grown significantly.

In the last decade, multimedia communications have experienced rapid growth and commercial success stimulated research interest in digital techniques for adapting and transmitting multimedia information. Seamless adaptation and transcoding techniques to adapt the digital content have achieved significant focus to serve consumers with the desired content in a feasible way. Universal Multimedia Access (UMA), which refers to universal access to multimedia content, is dominating the multimedia community in its access to multimedia contents in both wireless and wired networks. Figure 1 reveals the concept of Universal Multimedia Access. Pressure to satisfy user preferences and device requirements seamlessly is raising the need for content to be customized providing the best possible experience. UMA thus demands a generic adaptation technique to access

multimedia contents seamlessly. One way to solve this problem is to adapt the content to fit with the playback environment and another way is to adapt the playback environment to accommodate the existing input contents. But for the playback environment, different types of hardware that are already in service cannot be replaced by one single device. Ubiquitous and pervasive computing concepts are against this solution. Thus, the feasible way to accomplish the goal of UMA is to adapt the contents according to the playback environment's need.

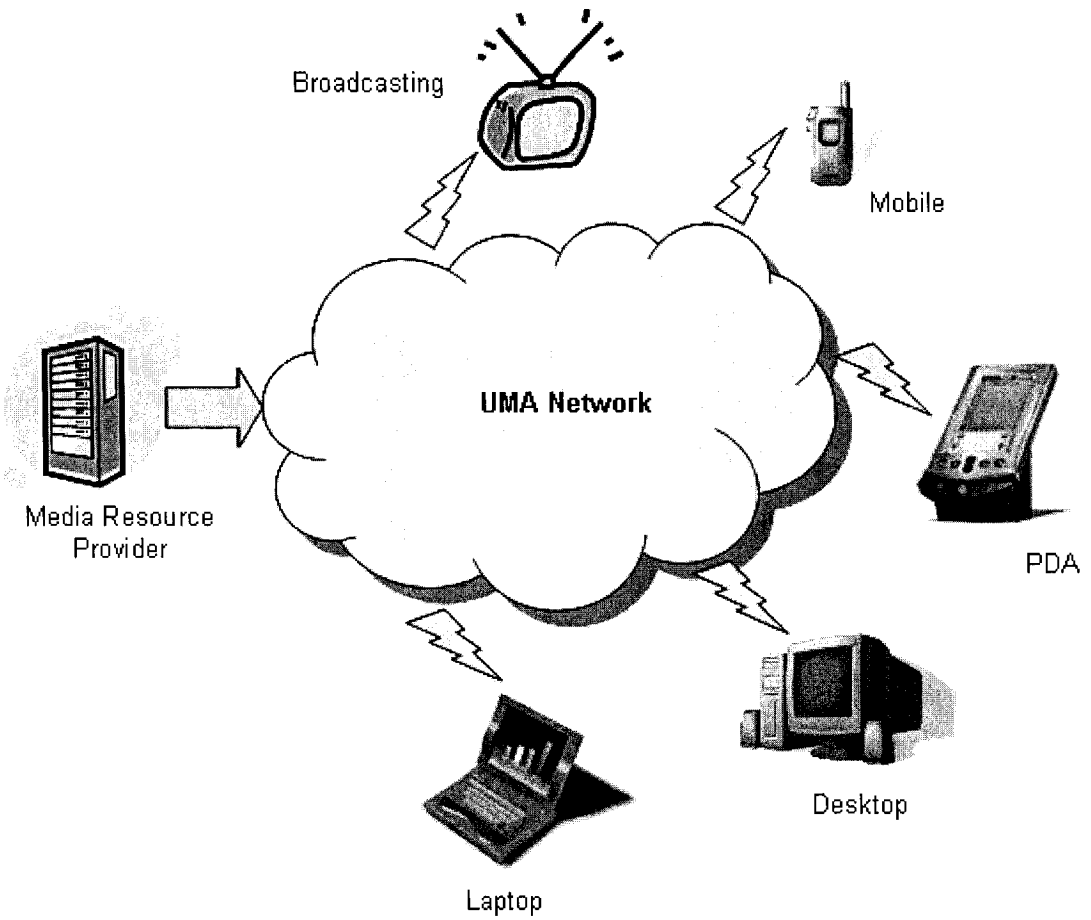


Figure 1. Universal Multimedia Access concept

It is thus realized that a successful adaptation framework for multimedia delivery and consumption, to be used by all players in the delivery and consumption chain, will provide content creators and service providers with equal opportunities and as well as

feasibility in the digital open market. This will also benefit the content consumers, providing them access to a large variety of content in an interoperable manner. The motivation behind this thesis lies here, with the need to produce architecture for video adaptation that will address the concept of UMA. The developed architecture considers that, rather than having multiple types of the same content, it is preferred to have the original content adapted accordingly, as depicted by UMA. As a very initial, but novel step, a specific challenge in the temporal adaptation of H.264 video bit-streams is portrayed in this paper. Adaptation is performed on the implemented adaptation module, which is in compressed domain and considers dynamic live streaming scenario.

1.2 Research Problem

Video adaptation differs from video coding in its scope and intended application locations. When performing video adaptation for frame rate conversion of an incoming or pre-coded compressed video bitstream for a low bandwidth outgoing channel, such as a wireless network, a high adaptation ratio is required. In addition, frame rate conversion is also needed when an end system supports only a lower frame rate. For example, some handheld devices can only decode and display at most 10 video frames per second. Frame-rate conversion or frame-skipping is often used as an efficient scheme to allocate more bits to the remaining frames, so that an acceptable quality can be maintained for each frame. Transcoding is the frequently applied technique in this regard for digital content customization. Scalable or non-scalable transcoding approaches to adapt digital video refers to adapting various coding parameters such as frame rate, spatial resolution or color. To improve multimedia content accessibility in terms of UMA, it is essential to adapt multimedia content in a feasible way, possibly in the delivery path to end consumers, with minimum processing delay. An alternative to instantaneous transcoding is to produce and store the content in several formats taking into account a wide variety of possible user preferences and finally making an appropriate selection at the delivery time which will surely require high storage in the resource server. In the conventional cascaded decoding/re-encoding scheme, a transcoder performs the full decoding of the

input bit stream and then carries out resizing and/or reordering of the decoded sequence before fully re-encoding it. Content customization is usually performed at the media resource server or at some intermediary nodes. End nodes are avoided for customization due to their lack of processing power and memory resources, especially when they are small handheld devices.

Since this thesis mainly focuses on the temporal adaptation, frame rate conversion can be simply accomplished by arbitrary frame dropping. For instance, dropping every other frame in a sequential order leads to a half rate reduction in the adapted sequence. However, when frames are skipped, the motion vectors cannot directly be reused because the motion vectors from the incoming video frame are pointed to the immediately previous frame. If the previous frame is dropped in the transcoder, the link between two frames is broken and the end decoder will not be able to reconstruct the picture using these motion vectors. Figure 2 shows a group of pictures consisting of 1 I-frame followed by 4 P-frames, where each P-frame is coded based on its previous frame. Now, if frame 2 is dropped, frame 3 cannot be regenerated and eventually all successive frames will be discarded.

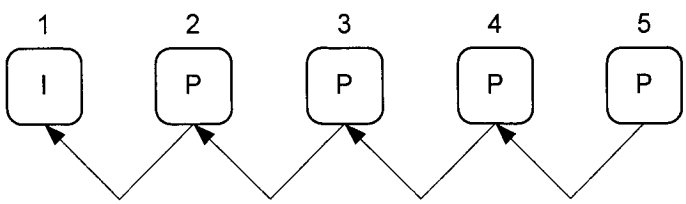


Figure 2. Inter Frame Dependency

Clearly, a crucial research problem can be identified from the above. There is need to perform all the adaptation operations in compressed domain complying explicit format specification avoiding the conventional aforesaid adaptation approaches. Moreover, once a suitable adaptation decision has been reached, deploying it in the existing network requires a generic and widely accepted standard applied to the process. Recent

developments of such standards have been significant in today's research in order to make multimedia contents available ubiquitously.

1.3 Research Contributions

The goal of this work is to come up with an architecture that is more efficient than that of the existing adaptation systems. Many techniques exist for adapting videos to satisfy heterogeneous resource conditions or user preferences. However, ad-hoc selections of bitstreams among various pre-compiled choices are used most often. To provide a systematic solution to pre-compiled processing, this thesis presents architecture for temporal video adaptation. The architecture excludes the conventional cascaded encoding/decoding and/or precompiled bit stream processing. The architecture is applied to a practical case of live adaptation applying dynamic frame rate. The challenge to conform the architecture to ISO/IEC MPEG-21 Part-7 Digital Item Adaptation specification is successfully achieved, which we shall see in the coming chapters.

In a nutshell, this work introduces an innovative adaptation technique, taking into consideration the existing tools and standards, and includes:

- The procedures to adapt the compressed ITU-T H.264 bitstream directly in compressed domain
- An implementation technique for creating a compressed video item along with its structure in XML format conforming MPEG-21
- A novel pseudo random frame skip algorithm for frame rate adaptation
- An implementation technique for creating adapted XML description that applies XSLT
- A complete adaptation engine to perform temporal video adaptation dynamically
- An implementation of a Client-Server architecture based adaptation system to illustrate the performance of the implemented adaptation module

- Publication is a peer-refereed conference: Razib Iqbal, Shervin Shirmohammadi, and Abdulmotaleb El Saddik, "Secured MPEG-21 Digital Item Adaptation for H.264 Video", *Proc. IEEE International Conference on Multimedia and Expo*, Toronto, ON, Canada, July 2006.

1.4 Thesis Outline

The rest of this thesis is organized as follows: **Chapter 2** familiarizes the reader with some background on video adaptation, an overview of MPEG-21 Digital Item Adaptation and H.264 video coding standard, and also outlines a literature review and critical analysis of existing techniques and tools for adaptation. **Chapter 3** depicts the design and implementation of the adaptation engine. **Chapter 4** presents the prototype of an adaptation system, and a performance check of the overall system. Finally **Chapter 5** concludes this thesis indicating a direction for future work.

Chapter 2

Background

2.1 Video Adaptation

Video adaptation is the process of transforming an input video to an output video, manipulating multiple bitstreams in order to meet diverse resource constraints and user preferences, while optimizing the overall utility of the video. Figure 3 shows the role of video adaptation in pervasive media environments. Usually adaptations tools take into account the content characteristics, device requirements, usage environments, and user preferences to adapt a video content.

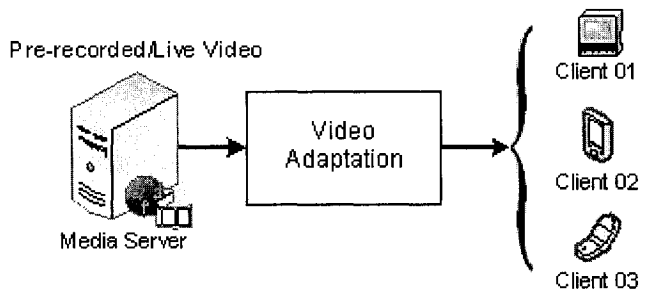


Figure 3. Adaptation to support heterogeneous terminals and devices

Adaptation engines are typically deployed in the intermediate locations between server and client, such as proxy servers, although they may be included in the servers or clients in some applications. Despite the burgeoning interest and advancement, video adaptation is still a relatively less defined field. There has not been a coherent set of concepts, terminologies, or issues defined over well-formulated problems. Section 2.4 reviews some of the literature on video adaptation.

Transcoding is the frequently applied technique for video customization. Video transcoding is defined as the conversion of digital video content from one format to

another. It involves decoding/decompressing the original data to a raw intermediate format (i.e. YUV) first and then re-encoding this into the target format. Scalable or non-scalable transcoding approaches to adapt video refers to adapting various coding parameters such as frame rate, spatial resolution or color. While homogeneous transcoding is done at the same coding standard, heterogeneous transcoding converts from one standard format to another standard. So, video adaptation can be denoted as homogenous video transcoding, which aims to reduce bit rate, frame rate and/or the resolution of the pre-encoded video stream. It does not involve any kind of syntax modifications to the coded video data. Therefore, the incoming compressed video stream preserves its format and compression characteristics after adaptation operation.

2.1.1 Video Adaptation Taxonomy

In video adaptation, "entity" is defined as a basic unit of video content that undergoes the adaptation process [1]. Entities are of different types and can be identified in different levels such as pixel, frame, group of pictures, etc. Different adaptation engines can be designed depending on the variation of these entities. For example, a video frame can be reduced in resolution, spatial quality, or skipped in order to reduce the overall bandwidth. It is important to mention that each of these entities are associated with certain resource requirements like transmission bandwidth, display capabilities, processor speed etc. An adaptation operation transforms the entity into a new one and thus changes the associated resources and utility values. In this subsection we present a simple taxonomy based on the type of manipulations performed.

Format Transcoding:

A basic adaptation process already mentioned above is to transcode video from one format to another. This approach is applied to make the video content compatible with the new usage environment. As we know, for different applications and devices, there are different formats prevailing, one basic implementation is to concatenate the decoder of one format with the encoder of the new format.

Adjusting Video Entity:

In a constrained environment, a popular adaptation approach is to trade-off some components of the entity against some resources (such as machine power). Such schemes are usually implemented by selection and reduction of some elements in a video entity, for example frames in a video clip. Uniform reduction sometimes is sufficient, while sophisticated methods help to improve the user perception of the video content.

Substitution:

This type of adaptation substitute selected elements in a video entity with less expensive counterparts, preserving the overall perceived functionality. For instance, a video sequence may be replaced with still frames, such as key frames and associated narratives to produce a slide show presentation, which eventually helps to reduce the bandwidth usage noticeably.

In the literature, it is revealed that international standardization bodies have recently developed standards to facilitate the deployment and interoperability of adaptation technologies. MPEG-21, a normative open framework for multimedia delivery and consumption for all the stakeholders, is one of them. In the following section, emphasis is placed on MPEG-21 standard, and it illustrates how this standard could be used in an adaptation architecture that is consistent with the effort put forward in this thesis.

2.2 MPEG-21 [2]

Due to the rapid growth and commercial success in the multimedia industry, solutions with advanced multimedia operability are becoming increasingly important as individuals as well as commercial providers are producing more and more digital content for multipurpose use. Lack of interoperability among advanced multimedia packaging and distribution applications motivated the initiatives of the ISO/IEC Motion Picture Expert Group (MPEG) towards working on the definition of enabling normative technology for

the multimedia contents tagged as ISO/IEC 21000 Multimedia Framework. The vision is to define a multimedia framework – "to enable transparent and augmented use of multimedia resources across a wide range of networks and devices used by different communities" [3]. That means MPEG-21 is to describe how various digital contents in different infrastructure fit together. Moreover, it is becoming increasingly difficult to deal with the different intellectual property rights associated with the multimedia contents. As a way out to this obscurity, ISO/IEC 21000 Multimedia Framework aims to enable transparent and augmented use of multimedia resources across a wide range of networks and devices[1].

Currently, ISO/IEC 21000 consists of the following parts, under the general title Information technology — Multimedia framework (MPEG-21):

Part 1: Vision, Technologies and Strategy
Part 2: Digital Item Declaration
Part 3: Digital Item Identification
Part 4: Intellectual Property Management and Protection
Part 5: Rights Expression Language
Part 6: Rights Data Dictionary
Part 7: Digital Item Adaptation
Part 8: Reference Software
Part 9: File Format
Part 10: Digital Item Processing
Part 11: Evaluation Methods for Persistent Association Technologies
Part 12: Test Bed for MPEG-21 Resource Delivery
Part 13: Scalable Video Coding
Part 14: Conformance Testing
Part 15: Event Reporting
Part 16: Binary Format
Part 17: Fragment Identification for MPEG Media

[1] Throughout this thesis, a common use of the term "framework" will be used to refer to ISO/IEC 21000 Multimedia Framework, unless otherwise mentioned.

2.2.1 Digital Item Adaptation (DIA)

The goal of the Terminals and Networks element described in ISO/IEC 21000 Part 1 was to achieve interoperable transparent access to distributed multimedia content by shielding users from network and terminal installation, management and implementation issues. To achieve this goal, adaptation of the multimedia content is required. Part 7 of the framework specifies the syntax and semantics of tools that may be used to assist the adaptation of Digital Items. In this framework, a bitstream and all its relevant descriptions are together denoted as Digital Item. This concept is illustrated in Figure 4:

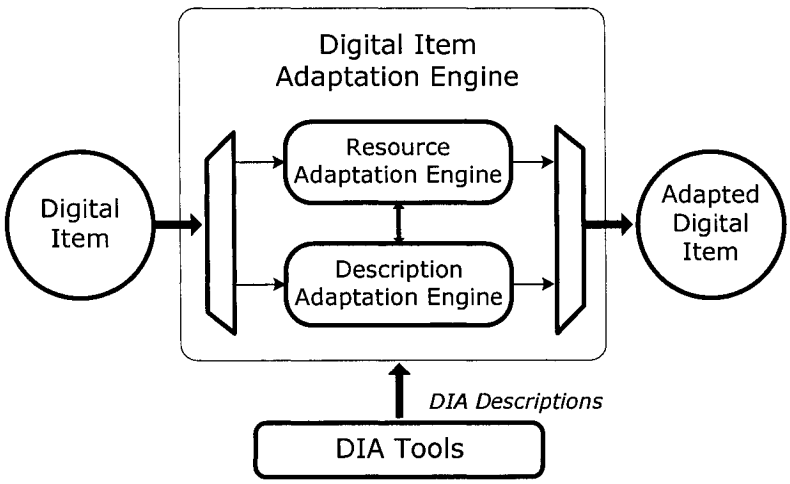


Figure 4. MPEG-21 DIA

It is important to emphasize that the adaptation engines themselves are non-normative tools of this part of ISO/IEC 21000. However, descriptions and format-independent mechanisms that provide support for Digital Item Adaptation in terms of resource adaptation, description adaptation, and/or Quality of Service management are within the scope of the standardization, and are collectively referred to as DIA Tools. From the above figure we see that, DIA engine consists of two sub-engines: Resource Adaptation Engine, and Description Adaptation Engine. A Digital Item is thus subject to resource adaptation and description adaptation, which together produce the Adapted Digital Item.

The Digital Item Adaptation tools in the framework are clustered into eight major categories as illustrated in Figure 5.

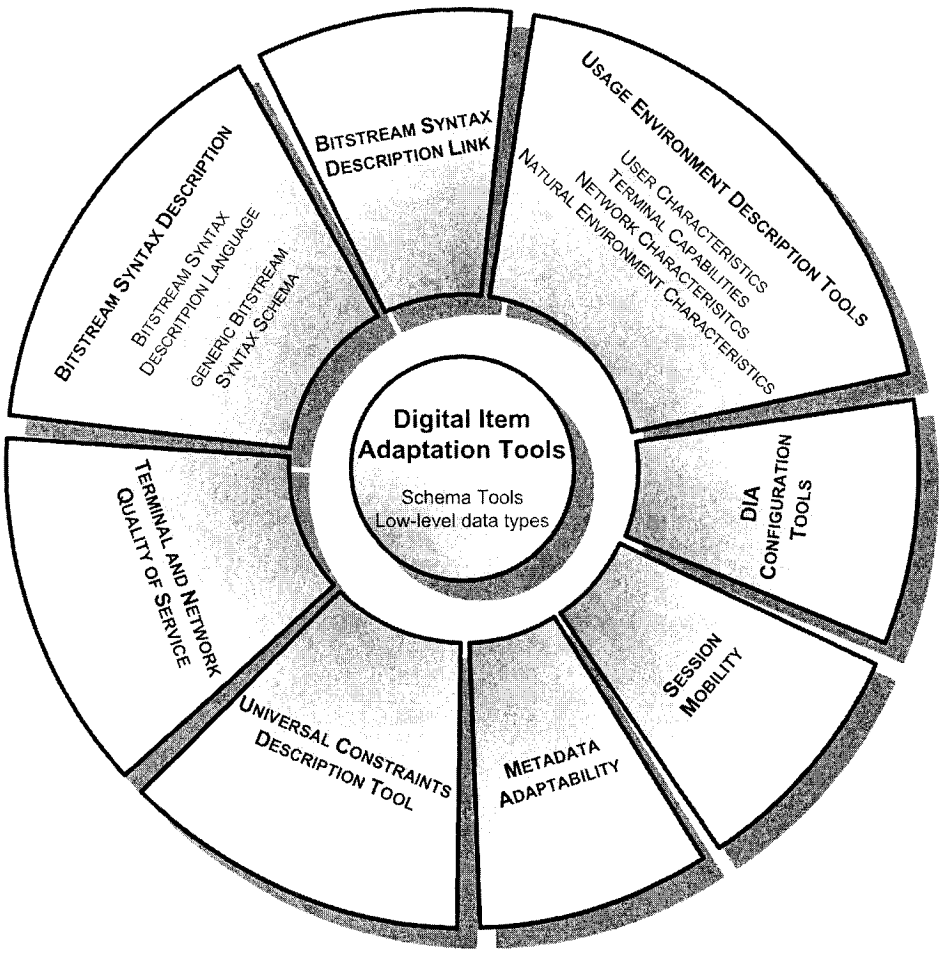


Figure 5. Overview and organization of Digital Item Adaptation tools

Each of these categories is described below:

The first major category is the Usage Environment Description Tools, which include User characteristics, terminal capabilities, network characteristics and natural environment characteristics. These tools provide descriptive information about the various properties of the usage environment, which originate from Users, to accommodate, for example, the adaptation of Digital Items for transmission, storage and consumption.

The second category is referred to as Bitstream Syntax Description Link (BSDLink), which provides the facilities to create a rich variety of adaptation architectures based on tools specified within the relevant parts of ISO/IEC 21000, ISO/IEC 21000-2, and ISO/IEC 15398 among others.

Bitstream Syntax Description (BSD) tools comprise the third major category of Digital Item Adaptation tools. A BSD describes the syntax, which in most cases is the high level structure of a binary media resource. Using such a description, a Digital Item resource adaptation engine can transform the bitstream and the corresponding description using simple operations such as data truncation.

Terminal and Network Quality of Service tools specified in this category describe the relationship between Quality of Service (QoS) constraints (such as network bandwidth or a terminal's computational capabilities), feasible adaptation operations satisfying these constraints and associated media resource qualities that result from adaptation. This set of tools therefore provides the means to trade-off these parameters with respect to quality so that an adaptation strategy can be formulated and optimal adaptation decisions can be made in constrained environments.

The Universal Constraints Description Tools encompass the fifth category of tools which describe the limitations and optimization constraints on adaptations.

The sixth category is referred to as Metadata Adaptability. This tool specifies hint information that can be used to reduce the complexity of adapting the metadata contained in a Digital Item. On the one hand, they are used for filtering and scaling, and on the other hand, for integrating XML instances.

The seventh category of tools, which is for Session Mobility, describes the configuration state information that pertains to the consumption of a Digital Item on one device that is transferred to a second device. This enables the Digital Item to be consumed on the second device in an adapted way.

Finally, the eighth category of tools is referred to as DIA Configuration Tools, which provides information required for the configuration of a Digital Item Adaptation Engine.

2.2.2 BSD based adaptation

The previous section illustrates that Digital Item Adaptation tools in the framework are clustered into eight major categories. Bitstream Syntax Description (BSD) tool of the specification comprises a major category of tools for DIA. In this sub section BSD based adaptation will be discussed in brief.

A binary media resource consists of a coding format specific structured sequence of binary symbols. A bitstream consists of a sequence of these binary symbols representing the resource. It is possible to retrieve a variety of versions from such a resource by simple editing style operations. But to ensure interoperability, it is advantageous to have an adaptation engine that is not aware of any specific codec. For this reason, in BSD based adaptation, an XML based generic approach is referred for manipulating bitstreams. XML is used to describe the high level structure of the bitstream, and the resulting XML document is called a Bitstream Syntax Description (BS Description, BSD). This BS Description is not to replace the original binary format of the resource. It only acts as a metadata. It describes how the bitstream is organized in layers or packets of data. With such a description, it is then possible for a resource adaptation engine [see Figure 4] to transform the BS Description, and then generate an adapted bitstream.

Figure 6 depicts the BSD based adaptation architecture. The architecture comprises the original Bitstream, its Bitstream Syntax Description, Transformed Bitstream Syntax Description, the Adapted Bitstream and two processors - a BS Description generator and a bitstream generator. The output produced in one adaptation step is an adapted bitstream and an updated BS Description that correctly references the new bitstream. This output enables the application of multiple successive adaptations.

In this architecture, the BS Description generator parses the bitstream and generates its BS Description. The bitstream and its BS Description are subject to the adaptation. An adaptation engine is assumed to determine the optimal adaptation for the media resource given the constraints provided by the DIA descriptions. For multi step adaptations, at the end of each adaptation step, the output description must correctly describe syntactical

elements of the adapted bitstream. This requirement allows the adapted bitstream and its description to be directly input to the next resource adaptation step, which might take place on a different device.

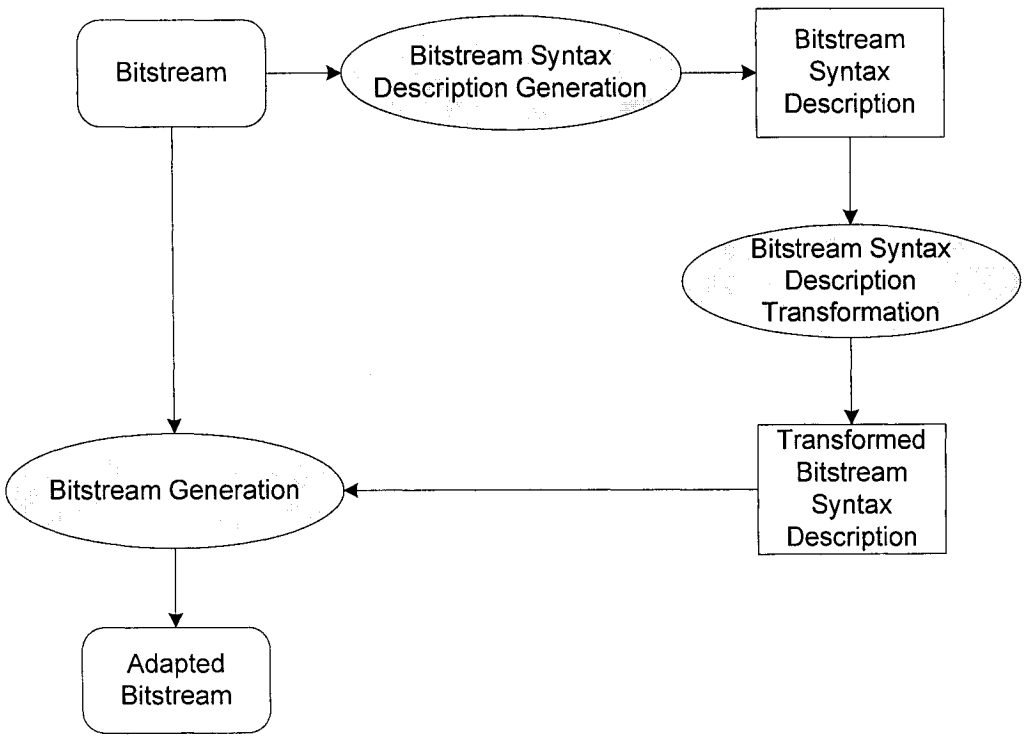


Figure 6. BSD based Adaptation Architecture

The Bitstream Syntax Description Transformation is not normatively defined in the framework. As shown in Figure 7, a possible solution is to apply an XSLT [4] or STX [5] style sheet to the BS Description. In order to ensure that a bitstream can be re-generated by the bitstream generation process, it is required that the transformed BS Description still conforms to the gBS Schema or the bitstream specific BS Schema.

In order to provide full interoperability, the adaptation engine should not be aware of the specific coding format. For this, a new language based on W3C XML Schema, called Bitstream Syntax Description Language (BSDL), is specified in the framework. With this language, it is possible to design specific Bitstream Syntax Schemas (BS Schemas) describing the syntax of a particular coding format. These schemas can then be used by a

generic processor to automatically parse a bitstream and generate its description and vice-versa.

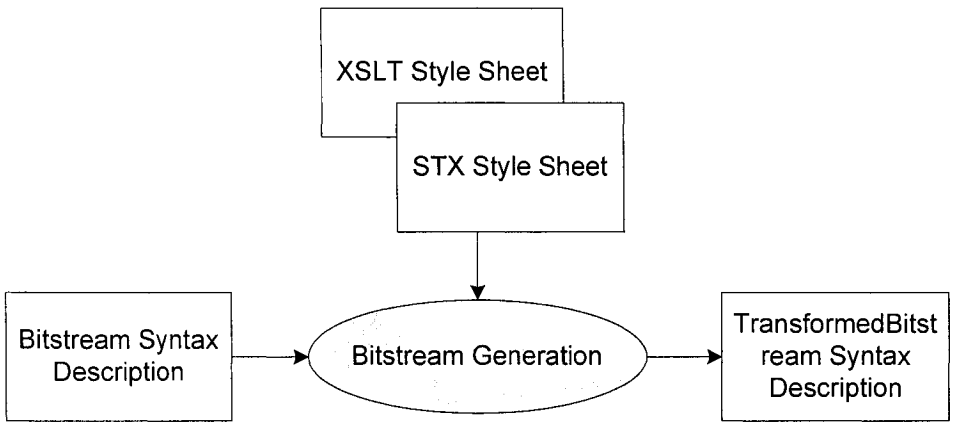


Figure 7. Architecture of BSD transformation

Since, BSDL provides a way for describing bitstream syntax with a codec specific BS Schema, an adaptation engine consequently requires knowing the specific schema. But if the adaptation takes place on some intermediary nodes like gateways and proxies, then a codec independent schema is more appropriate. Therefore, a generic Bitstream Syntax Schema (gBS Schema) is specified in the framework.

2.2.3 generic Bitstream Syntax Schema (gBS Schema)

The normative generic Bitstream Syntax Schema introduces means to describe hierarchies of syntactical units and addressing means for efficient bitstream access. This sub-section gives an overview of the most important elements of the gBS Schema. It is important to mention that a description conforming to this schema is called a generic Bit Stream Description (gBS Description, gBSD).

The gBS Description provides an abstract view on the structure of the bitstream that can be used in particular when the availability of a specific BS Schema is not ensured. However, for transformations on gBS Descriptions, it is important to include coding format specific information in attributes of the gBS Description. For the BSDL case on the other hand, all coding format specific information can be provided by the BS Schema,

which is common to all BS Descriptions following this schema. The framework specifies the technologies required for the bitstream generation process, namely BSDL-1 and the gBS Schema. The bitstream generation process is specified for both technologies as two normative processors, named BSDtoBin and gBSDtoBin respectively. This research is linked to the gBS Schema which is for a flexible binary resource adaptation providing the following functionalities:

- *Codec independence* – it can be used to describe any binary resource in a codec independent manner and no codec specific schema is required to re-generate the bitstream.
- *Semantic marking* – it provides means for semantically meaningful marking of syntactical elements, by use of the "marker" handle. Such "markers" can be used for semantically meaningful, efficient adaptations of binary media resources.
- *Hierarchical descriptions* - it contains elements for the description of a bitstream in a hierarchical fashion that allows grouping of bitstream elements for efficient, hierarchical adaptations.

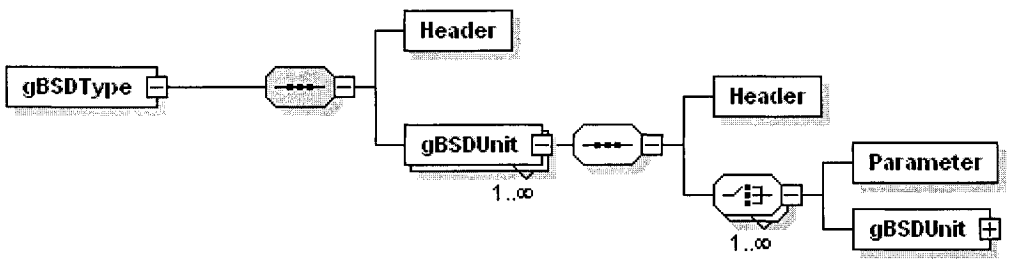


Figure 8. gBS Schema Elements

Detail semantics of gBS Schema is defined in [2]. The gBS Schema defines three types of gBSD elements (see Figure 8):

- Header
- gBSDUnit
- Parameter

Header - Header element comprises the alias and URI of a classification scheme that provides semantics for the names of gBSDUnits and Parameters. These names allow a codec aware adaptation engine to perform bitstream adaptations in an interoperable and extensible way because classification schemes are standardized and can be easily extended by simply adding new terms. The Header element also defines default address values such as the URI of the bitstream location, address mode (absolute, consecutive, or offset) and address units (bit or byte) to be used. The scope of these default values is the complete gBSD but these can be overwritten by additional Header elements within a gBSDUnit element or by address attributes within a gBSDUnit or Parameter element.

gBSDUnit - The gBSDunit element represents a segment of the bitstream but does not include the actual values of that segment. Therefore, it can be used for bitstream segments which might be removed during adaptation or for carrying further gBSDUnit or Parameter elements, which then results in a hierarchical structure for efficient bitstream manipulations. It can also be used to represent blocks of data in the bitstream that do not play any role in the adaptation process. Each gBSDUnit carries several possible attributes:

- `start` and `length` - attributes conveying addressing information depending on the address mode.
- `syntacticalLabel` - attribute for including coding-format specific information identified via classification scheme terms.
- `marker` - attribute which provides a tool for application-specific (e.g., semantic) information to be used for performing complex bitstream adaptations in an efficient way.

In addition, each gBSDUnit may also contain its own bitstream segment location, address mode and address unit, which allows to locate parts of the bitstream on different devices within a distributed multimedia environment.

Parameter - The Parameter element describes a syntactical element of the bitstream, the value of which might need to be changed during the adaptation process. Therefore, it provides the actual numerical value and datatype of the bitstream segment. The datatype

is required for correctly encoding the syntactical element in the bitstream during the gBSDtoBin process. Due to the multitude of possible datatypes they are specified by using the xsi:type attribute in the instance instead of within the gBS Schema. Nevertheless, frequently used basic datatypes are specified within an additional XML schema. Similar to the gBSDUnit element, the Parameter element may also contain its own address information.

2.2.4 Bitstream generation with gBSDtoBin

gBSDtoBin process shown in Figure 9 takes as input the original bitstream and a transformed gBSD and edits the bitstream according to the gBSD. The resulting bitstream is the adapted version of original bitstream.

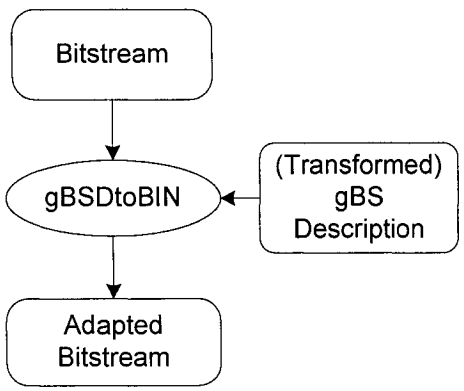


Figure 9. Architecture of Bitstream generation using gBSDtoBin

In the framework, the gBSDtoBin process is specified as follows:

1. First the gBS Description is parsed as an XML document.
2. gBSDtoBin then sequentially starts to step through the top-level gBSDUnit elements. For each gBSDUnit element, the gBSDtoBin process will verify whether this gBSDUnit element has further gBSDUnit or any Parameter child elements. If there are such child elements, then gBSDtoBin will recursively parse these child elements.

This depth-first recursion will continue until an element without gBSDUnit or Parameter child elements is reached.

3. When an element without gBSDUnit or Parameter child elements is encountered, the gBSDtoBin will proceed as follows, depending on whether this element is a gBSDUnit or a Parameter element:

   gBSDUnit: gBSDUnit elements describe segments of data of the original bitstream that do not need to be altered. Therefore, gBSDtoBin will simply locate this segment in the bitstream and append it to the new bitstream. To locate this segment in the bitstream, the addressing scheme of the gBS Schema must be used. The information needed are: bitstream location as specified by the `bs1:bitstreamURI` attribute, the unit used (bit or byte) as defined in the `addressUnit` attribute, the addressing mode as defined in the `addressMode` attribute and the start address as well as the length of the segment (in "units") as specified by the start and length attributes. The values of the `bs1:bitstreamURI`, `addressUnit` and `addressMode` attributes are determined by the scope semantics of these attributes. For each of the three address attributes that a gBSDUnit element does not specify itself, it inherits its value from the first ancestor of the gBSDUnit element that does specify it, otherwise it assumes the default value specified in the Description element with `xsi:type="gBSDType"`. There are three possible address modes. Depending on the mode, the gBSDtoBin shall proceed as follows in order to locate the position of the gBSDUnit in the original bitstream:

   I. Absolute: Both start and length attributes must be specified. The start attribute of gBSDUnit gives the exact (absolute) position in the original bitstream (offset from the beginning of the bitstream) at which the segment begins. The bitstream is assumed to start at 0 units.

   II. Consecutive: The length attribute is mandatory whereas the start attribute is optional. The segment represented by this gBSDUnit starts at the next unit (bit or byte) after the end of the preceding element (previous sibling or parent, if the element is the first child). The length of the segment is given by the value of the

length attribute. If a start attribute is present, its value (n) provides the offset from the previous sibling (or parent). Therefore the beginning of the data represented by this gBSDUnit element can be located n "units" (bit or byte) after the end of the previous element.

III. Offset: Both start and length attributes must be specified. The value of the start attribute is the offset of this gBSDUnit from the absolute start position of its parent gBSDUnit.

Parameter: Parameter elements represent syntactical elements, the values of which may have been changed during the description transformation. Therefore, gBSDtoBin needs to correctly encode the value of the Parameter element and append it to the new bitstream. The value of the Parameter element is given by its Value child element and the encoding format will be determined from the xsi:type attribute specified in the instantiation of the Value element. For gBSDtoBin processes that directly encode the new Parameter value in the original bitstream, it is possible to locate the position of the Parameter in the bitstream following the same process as that for the gBSDUnit element, as described above.

2.3 H.264

H.264, the latest coding and compression standard, by ITU-T and ISO/IEC as International Standard MPEG-4 part 10 Advanced Video Coding (AVC), is currently dominating the field by offering a flexible architecture and compression gain of up to 50% [6]. H.264 delivers the same quality as MPEG-2 at a third to half the data rate. When compared to MPEG-4 Part 2, H.264 provides up to four times the frame size at a given data rate [7]. One of the magnificence of H.264 coding efficiency is its better image quality. Once reaching its limits, H.264 encoded frames, rather than breaking down into distinct blocks, softens the image as compression increases. An example of H.264 image quality compared to that of MPEG-4 can be seen in Figure 10.

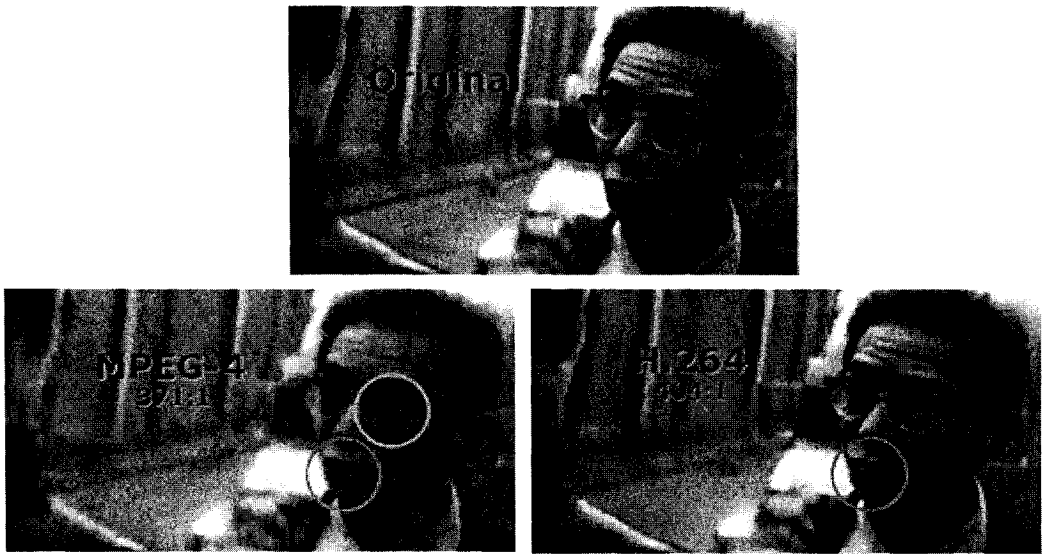


Figure 10. Example of H.264 image quality

In H.264, resolution ranges across a wide operating range from 3G to HD and everything in between. It provides solutions for broadcast, video-on-demand (VOD) or multimedia streaming over cable, satellite, Cable Modem, DSL, LAN, terrestrial, wireless networks etc. H.264 provides exceptional video quality at impressively low data rates as depicted in **Error! Reference source not found.**.

Table 1. H.264 image quality irrespective of size

Scenario	Resolution	Frame Rate
Mobile (3G)	176 x 144	10 – 24 fps
Standard Definition	640 x 480	24 fps
High Definition	1280 x 720	24 p
Full High Definition	1920 x 1080	24 p

fps = frame rate per second, p = progressive[2]

Compared to other existing video coding standards like H.263 [8], MPEG-2 [9] etc., the selective features of H.264, among many [10], compels the choice of H.264 for this particular thesis are highlighted below:

[2] Interlaced scanning divides a picture into odd and even lines and then alternately refreshes them at 30 frames per second. In progressive scanning scans the entire picture line by line where captured images are not split into separate fields like in interlaced scanning.

Multiple reference pictures for motion compensation

Motion compensation is a way of describing the difference between consecutive frames in terms of where each section (e.g. macroblock) of the former frame has moved to.

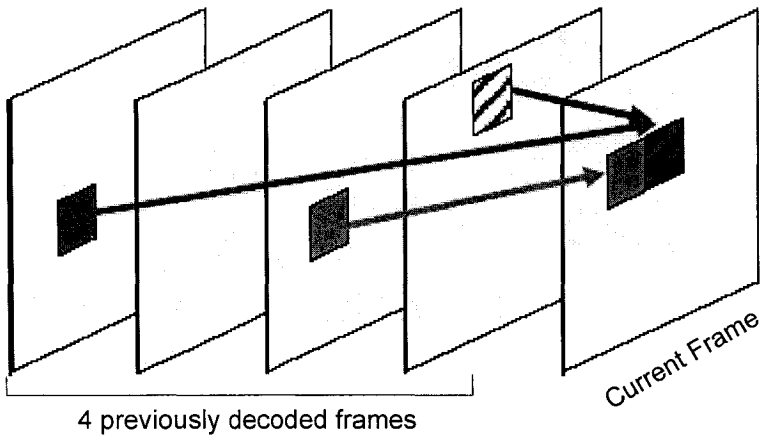


Figure 11. Multi frame motion compensation in H.264

In MPEG-2 and its predecessors, usually only one reference picture (previous picture) is used to predict the values in an incoming picture for P-frames. But H.264 coding standard extends the enhanced reference picture selection technique of H.263++ (a.k.a. H.263v3 or H.263 2000) which allows selecting among a larger number of pictures that have already been decoded. Motion compensated bi-prediction in MPEG-2 uses only two specific pictures - the previous I or P frame and the next I or P picture, in display order.

Decoupling of referencing order from display order

H.264 coding standard permits the encoder to decide the ordering of pictures for referencing and display purposes with a high degree of flexibility constrained only by a total memory capacity bound imposed to ensure decoding ability.

Decoupling of picture representation methods from picture referencing capability

In earlier video coding standards, bi-predictive-encoded pictures could not be used as references for prediction of other pictures in the video; H.264 provides the encoder more flexibility and ability to use a picture for referencing that is a closer approximation to the picture being encoded.

Redundant pictures

H.264 design enables an encoder to send redundant representations of regions of pictures, enabling a degraded representation of regions of pictures for which the primary representation has been lost during data transmission.

SP/SI synchronization/switching pictures

The H.264/AVC design includes a new feature consisting of picture types that allow exact synchronization of the decoding process of some decoders with an ongoing video stream produced by other decoders without penalizing all decoders with the loss of efficiency resulting from sending an I-frame. This enables switching a decoder between representations of the video content that used different data rates, recovery from data losses or errors, as well as enabling trick modes such as fast-forward, fast-reverse, etc.

Motion vectors over picture boundaries

While motion vectors in MPEG-2 and its predecessors were required to point only to areas within the previously-decoded reference picture, the picture boundary extrapolation technique first found as an optional feature in H.263 is included in H.264/AVC.

Parameter set structure

In H.264 design, parameter set design provides robust and efficient conveyance of header information. This key information is separated for handling, since the loss of a few key bits of information in sequence header or picture header could have a severe negative impact on the decoding process.

NAL unit syntax structure

In H.264, each syntax structure is placed into a logical data packet called a NAL unit, which allows greater customization of the mode for carrying the video content in a manner appropriate for each specific network.

In summary, advanced compression technique, improved perceptual quality, network friendliness and versatility of the codec [11] [12] drives H.264 to outperform all other pre-existing video coding standards.

2.4 Literature Review

In terms of dynamic resource adaptation and content negotiation, many approaches have been proposed as solutions to problems in the field. This chapter summarizes dynamic resource adaptation especially video adaptation, as a means of facilitating Universal Multimedia Access (UMA). Most recent works in this area are highlighted here.

Video adaptation is a promising meadow for challenging pervasive media applications. There have been many research activities and advances in this field in the past decade. Earlier work for instance [13] and [14] explored some interesting aspects of adaptation such as bandwidth reduction, format conversion, and modality replacement for Web browsing applications. In [13], it was realized that much of the rich multimedia content cannot be easily handled by the client devices with limited communication, processing, storage and display capabilities. In order to improve content access, authors outlined a system for scalable delivery of multimedia which uses an InfoPyramid for managing and manipulating multimedia content composed of video, images, audio and text. Recently, international standards such as MPEG-7 [15], MPEG-21, W3C [16], and TV-Anytime [17], have developed related tools and protocols to support development and deployment of video adaptation applications. Different standards are targeted at different applications. For example, TV-Anytime focuses on adaptation of content consumption into high-volume digital storage in consumer platforms such as personal video recorders. W3C and IETF focus on facilitating server/proxy to make decisions on content adaptation and delivery.

In the paper by Xin et al. [18], a comprehensive overview of digital video transcoding can be found. The paper provides a more in-depth view of the architecture and techniques, and it covers topics including quality optimization, complexity reduction techniques, and related applications such as logo and watermark insertion.

The big picture of pervasive multimedia application environment where rich multimedia content is accessible anytime, anywhere on nearly any kind of device and content providers aspire to offer multimedia-based information of the best possible quality from

the consumers' perspective, without neglecting economic principles is considered in [19]. Authors focused on device and coding format independent multimedia content adaptation and publishing applying tools specified within MPEG-21 for interoperable multimedia communication.

In the article "Optimal quality adaptation for scalable encoded video" [20], authors investigated quality adaptation algorithms for scalable encoded variable bit rate video over the Internet. The goal was to develop a quality adaptation scheme that maximizes perceptual video quality by minimizing quality variation, while at the same time increasing the usage of available bandwidth.

Overview of DIA, its use in multimedia applications, and report on some of the ongoing activities in MPEG on extending DIA for use in rights governed environments is available in literature [21] [22]. Terminal and Network Quality of Service (QoS) constraints, feasible adaptation operations satisfying these constraints, associated media resource qualities that result from the adaptation, tools that enable low-complexity adaptation of metadata, and tools for session mobility are discussed by A. Vetro and C. Timmerer [21]. Authors suggest that modality conversion can provide a better solution where the scaling of video content is not sufficient to meet terminal or network constraints. But it will highly depend on the scalability issue and processing capability of the device. Devillers et al. have proposed BSD based adaptation in streaming and constrained environments [23]. In their framework, authors have emphasized on BSD based adaptation applying BS Schema and BSDtoBin processor.

Bonuccellit et al. [24] have introduced buffer-based strategies for temporal video transcoding adding a fixed transmission delay for buffer occupancy in frame skipping. A frame is skipped if the buffer occupancy is greater than some upper value, and it is always transcoded if the buffer occupancy is lower than some lower value, provided the first frame which is I-frame, is always transcoded.

Group of Pictures (GOP) level rate adaptation scheme is introduced in [25] for a single stream, variable target bitrate H.264 encoder, which allows each group of pictures to be

encoded at a specified bitrate, using a dynamically updated table to select the starting quantization parameter for each GOP. Block Adaptive Motion Vector Resampling (BAMVR) method is proposed in [26] to estimate motion vector for frame rate reduction in H.264. But the transcoder follows straightforward cascading architecture of the decoder and encoder.

Even though the literature contains research on MPEG-21 and its contribution to multimedia, relatively little research applying H.264, such as [26], has been explored. Explicit adaptation approaches conforming MPEG-21 are not present. Moreover, the approaches mentioned above are aimed towards adaptation for pre-recorded bitstream but not applicable for live video streaming.

Chapter 3

Design & Implementation of Adaptation Engine

Ubiquitous computing environment concept permits end users to have access to the multimedia and digital content anywhere, anytime and any how they want. The inevitable outcome is the presence of numerous devices each with their own set of distinct features. As a result importance of resource customization and dynamic update of user preferences set the primary challenge towards seamless access. This is especially important in a mobile environment where available resources like bandwidth, device capability are of importance.

The main concept of digital item adaptation for video resources has been portrayed in Figure 12. On the delivery path, starting from media resource server to end client, the desired video content is being adapted depending on various device requirements and user preferences.

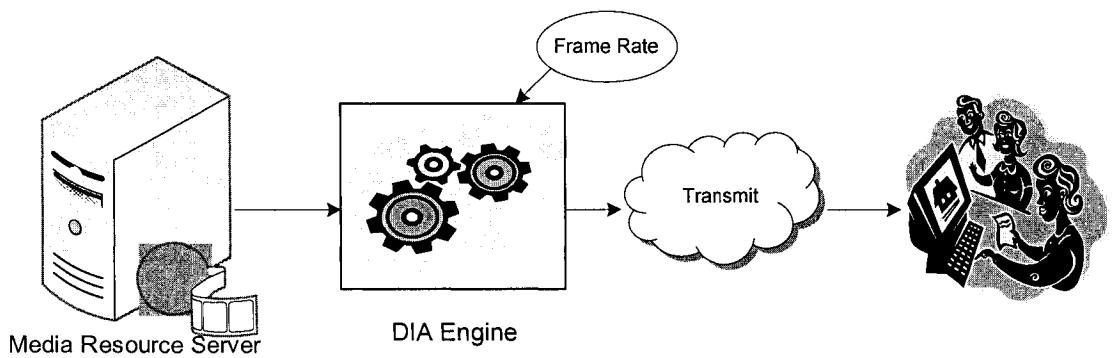


Figure 12. Adaptation on demand

A Systematic Procedure for Designing Video Adaptation Architecture involves following activities:

- Identify adequate entities for adaptation, for example: pixel, frame, group of pictures
- Identify feasible adaptation technique, for example: re-quantization, frame dropping
- Develop method for measuring and estimating resource and utility values associated with the video entities undergoing adaptation
- Given user preferences and constraints on resource or utility, develop strategies to find the optimal adaptation operator(s) satisfying the constraints

Focus in this thesis is placed on temporal adaptation for transmitting both pre-recorded and live H.264 bitstreams, where video content is adapted directly from the bitstream straight away. This chapter provides detailed design and implementation of such an adaptation module.

3.1 Design of the Adaptation Architecture using gBSD

3.1.1 Generation of the Digital Item

As we have already learnt that for the MPEG-21 DIA, generation of Digital Item is one of the important tasks at initial stage. This Digital Item performs as original content for resource server or content provider on the delivery path. One of the prime initial goals was to design and create the Digital Item conforming to the standards that would also be feasible to apply in real time applications. In MPEG-21 framework, generation of original Bitstream Syntax Description from binary data (BintogBSD) is not normatively specified. In our approach, gBSD is generated during the encoding process of the bitstream because the encoder best knows the structure of the bitstream. Based on adaptation possibilities, (in our case, temporal adaptation), gBSD is organized in the XML document.
Figure 13 shows how it is mapped to generate the Digital Item. Uncompressed video stream is the essential content that needs to be processed and delivered to the end user. Uncompressed video, YUV 4:2:0 formats, is the input to the H.264 encoder.

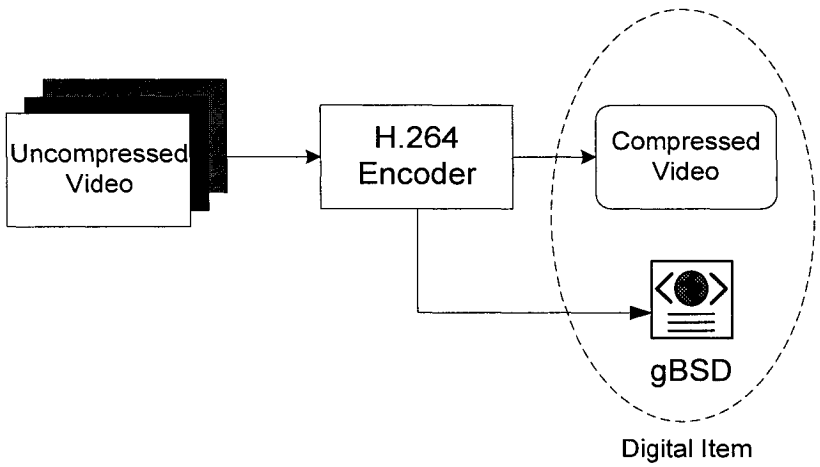


Figure 13. Generation of Digital Item

The H.264 encoder encodes raw video input to compressed bitstream conforming to ITU-T specification. In the design, generic bit stream description (gBSD) of each compressed video is generated while it is being encoded inside the encoder in the form of XML. Complying with gBS Schema (see Figure 8), this gBS Description consists of frame number, frame start, frame type and length of each frame for the entire bitstream. Both the compressed video and gBSD shape the Digital Item together for adaptation.

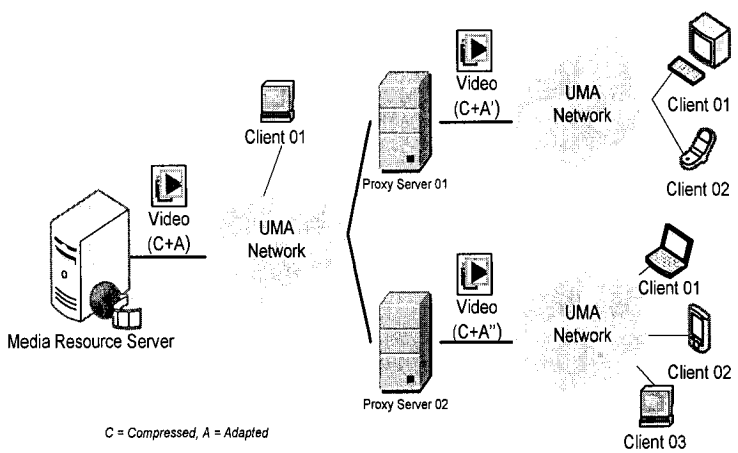


Figure 14. Adaptation on the delivery path

Once the Digital Item is generated, the content can be adapted in some trusted intermediary node in the delivery chain as shown in Figure 14.

3.1.2 Design of Adaptation Engine

The core of the implemented modules in this thesis is the adaptation module to perform adaptation dynamically. Designing the adaptation module was the second milestone for the entire project. Figure 15 shows the structural design of the transcoder to generate adapted video stream from Digital Item. The transcoder is comprised of:

- An XSLT processor, and
- An Adaptation engine

For adaptation, the adaptation module first fabricates the original XML description of the compressed video to output the adapted XML. Based on this adapted XML, compressed bitstream is engineered to produce adapted compressed bitstream.

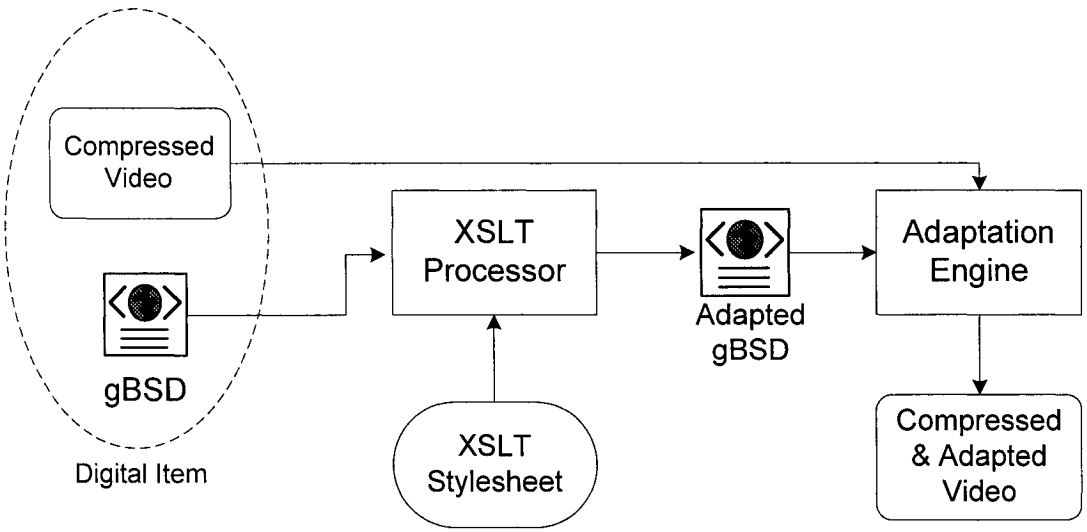


Figure 15. Structure of Adaptation module

Original gBSD is transformed to adapted gBSD by means of XSLT. The XSLT processor takes a tree structure as its input by parsing the gBSD and generates another tree structure as its output into Adapted gBSD.

An XSL style sheet defines the template rules and describes how to display a resulting document. So, an XSLT style sheet is designed to characterize the transformation constraints of the original XML. Template rules assembled in the style sheet are

predisposed to filter the original XML based on the actual encoding frame rate and target frame rate. For this purpose, the user preference or device requirement, in terms of bitrate, is considered as an external input to the transcoder. XSLT processor engineered inside the transcoder finally performs the XML transformation to generate the adapted XML ensuring constraints defined in XSLT style sheet. Figure 16 illustrates an example of the original and adapted gBSD transformation.

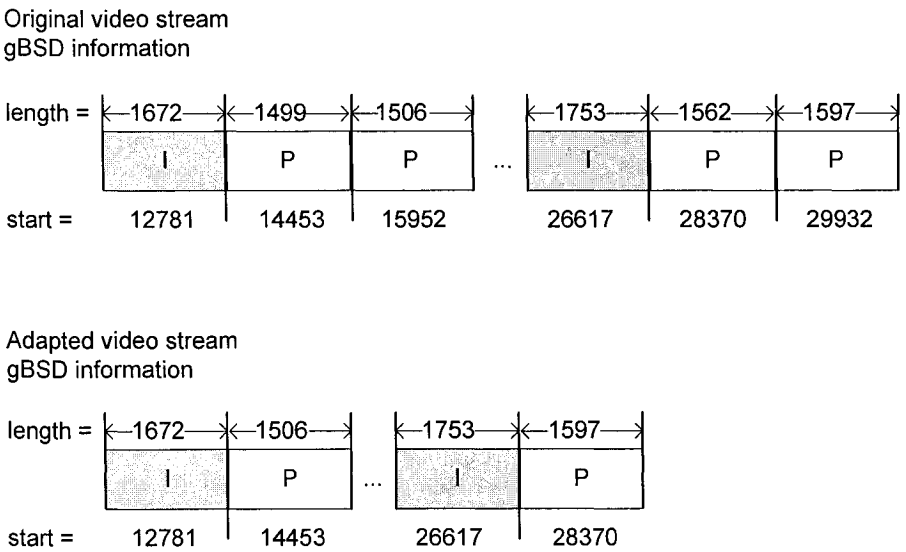


Figure 16. Example of Original and adapted gBSD transformation for video stream

Finally, an adaptation engine in the module generates transformed/adapted H.264 compliant video by discarding gBSD portions corresponding to specific frames in adapted XML from the original compressed video. The final output from the system is an H.264 format compliant adapted compressed bitstream.

3.1.3 Decoding and playing compressed adapted video

Decoding the adapted bitstream is featured to be completed at the client side. The test implementation (see section 4.1) embeds the features to connect to server, send preferences, receiving adapted video file and lastly playing after decoding the received file.

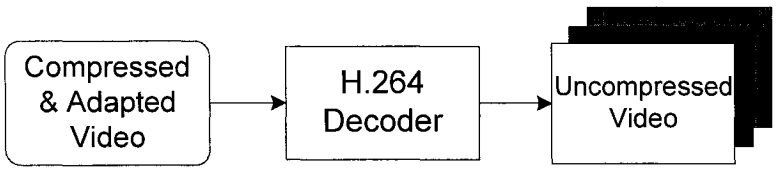


Figure 17. Decoding Compressed and Adapted Video

Compressed and adapted video generated in previous steps conforms to the normative definition of H.264 bitstream syntax description, so a standard H.264 decoder is placed to uncompress the video file. Figure 17 shows this simple step to decode the adapted bitstream. Once the uncompressed video is available, it is played in a YUV player also designed and implanted in the client system. In practical context, the decoder will be embedded in the player to view the decoded frames on the fly.

The system architecture and design presented in this section was implemented to show the MPEG-21 based digital item adaptation's applicability and performance. Implementation milestones and programming details of the adaptation module and other experimental add-ons are illustrated in the next section.

3.2 Implementation of the Adaptation Architecture

3.2.1 Functional Requirements

The 2 cornerstones that this research produces in terms of implementation are:

1. Producing the Digital Item
2. Implementing the Transcoder

C++ programming language in conjunction with C was used to write the codes for Digital Item generation. Microsoft Foundation Class (MFC) application framework was applied for implementing the test application embedding the above two milestones. Another intrinsic reason to choose C/C++ is the other reference software's [27] implementation platform and dependency.

3.2.2 Producing the Digital Item

In section 3.1.1 it was mentioned that a Digital Item comprises of the compressed video along with its gBSD. Generation of gBSD process is not normatively specified in [2], so the encoder was enhanced with the gBSD generation functionality. In order for implementation, the ITU-T reference software implementation of the H.264 encoder available at [27] has been enhanced with gBSD generation functionality.

While the encoder performs compression from the uncompressed video i.e. YUV input, the start of each frame in the compressed bitstream, the length of each encoded frame along with NAL units are computed as follows:

$$length_n = (total\_bit\_usage_n - total\_bit\_usage_{n-1})/8$$
$$start_{n+1} = (start_n + length_n)$$
where, n refers to frame number

After computing the frame start, length and frame type of each encoded frame, an XML description containing this information is generated for the whole bitstream. Block diagram of this implementation is shown in Figure 18.

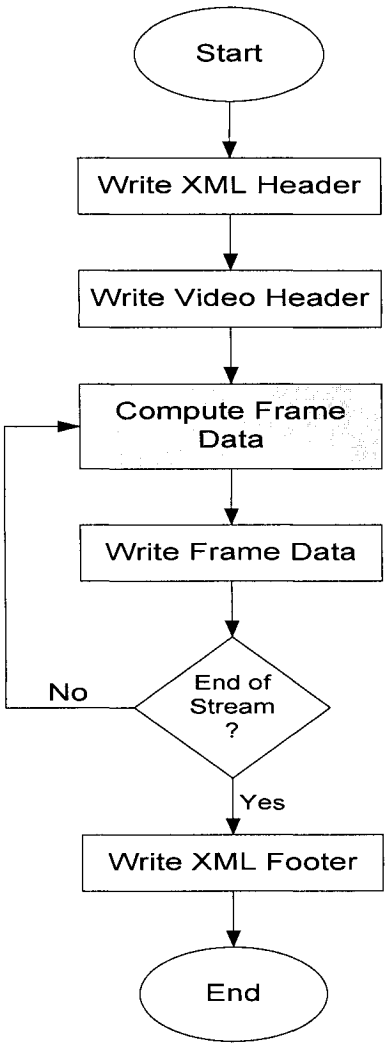


Figure 18. Block diagram of gBSD generation

Writing XML Header

XML document header specifies the XML version number and the character encodings followed by Uniform Resource Identifiers (URIs) to identify MPEG-21 descriptions for each gBSD document. Sample code is given below:

```
fprintf(fp_out_gbsd,"<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
fprintf(fp_out_gbsd,"<dia:DIA xmlns=\"urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS\"
xmlns:dia=\"urn:mpeg:mpeg21:2003:01-DIA-NS\"
```

```
xmlns:xs=\"http://www.w3.org/2001/XMLSchema\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">\n");
fprintf(fp_out_gbsd,"        <dia:Description xsi:type=\"gBSDType\">\n");
```

where "*fp_out_gbsd*" is the output file buffer.

Writing Video Header

To define the default values for address mode, address unit and bitstream URI for the entire gBSD, `dia:Description` element is used with the `xsi:type="gBSDType"` as described in chapter 4. Sample code to generate the video header is given below:

```
fprintf(fp_out_gbsd,"<Header>\n");
fprintf(fp_out_gbsd,"        <ClassificationAlias alias=\"MV4\"
href=\"urn:mpeg:mpeg4:video:cs:syntacticalLabels\"/>\n");
fprintf(fp_out_gbsd,"        <DefaultValues addressUnit=\"byte\"
addressMode=\"Absolute\" globalAddressInfo=\"%s.raw\"/>\n",iFilename);
fprintf(fp_out_gbsd,"</Header>\n");
```

Writing Frame Data

The `gBSDUnit` describing the individual section of the bitstream is assembled with start, length and marker in bytes. Since, the bitstream consists of the NAL headers and video frames, the syntax to write gBSD information was chosen using switch-case statements with a "`DataType`" id passed as a parameter from the caller function. For implementation convenience, 'Frameset' has been assumed to represent the number of frames equal to the frame rate at which the raw video is being encoded (usually 30fps for prerecorded video). Frameset is used to identify a cluster of frames for the encoded bitstream while represented in XML and repeated from beginning when the max value is reached. Sample code to write frame data is shown below:

```
switch(iDataType)
{
    case GBSD_TYPE_ELEMENT_SANDE_P:
            fprintf(fp_out_gbsd,"<gBSDUnit syntacticalLabel=\"Frame-%ld\" start=\"%ld\"
length=\"%ld\" marker=\"P\"/>\n",iNumber, iStart, iLength);
            break;
    case GBSD_TYPE_ELEMENT_SANDE_I:
            fprintf(fp_out_gbsd,"<gBSDUnit syntacticalLabel=\"Frame-%ld\" start=\"%ld\"
length=\"%ld\" marker=\"I\"/>\n",iNumber, iStart, iLength);
            break;
    case GBSD_TYPE_ELEMENT_SANDE_IDR:
            fprintf(fp_out_gbsd,"<gBSDUnit syntacticalLabel=\"Frame-%ld\" start=\"%ld\"
length=\"%ld\" marker=\"IDR\"/>\n",iNumber, iStart, iLength);
            break;
    case GBSD_TYPE_NAL_UNIT:
            fprintf(fp_out_gbsd,"<gBSDUnit syntacticalLabel=\"NALUnit\" start=\"%ld\"
length=\"%ld\"/>\n",iStart, iLength);
            break;
    default:
            return(false);
            break;
}
```

Writing XML Footer

After writing all frame data in the gBSD document, when end of encoded stream is reached gBSDescription output buffer is ended by closing "`dia:Description`". Sample code is given below:

```
void WriteXMLFooter(void)
{
    fprintf(fp_out_gbsd,"    </dia:Description>\n");
    fprintf(fp_out_gbsd,"</dia:DIA>\n");
}
```

Final output from the H.264 encoder is the gBS Description (i.e. XML) and compressed H.264 compliant video. Sample XML representation of the compressed bitstream (i.e. H.264 video) is shown in Figure 19:

```
<?xml version="1.0" encoding="UTF-8" ?>
<dia:DIA xmlns="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS" xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-
NS" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <dia:Description xsi:type="gBSDType">
    <Header>
      <ClassificationAlias alias="MV4" href="urn:mpeg:mpeg4:video:cs:syntacticalLabels"/>
      <DefaultValues addressUnit="byte" addressMode="Absolute" globalAddressInfo="test.raw"/>
    </Header>
      <gBSDUnit syntacticalLabel="NALUnit" start="0" length="23" />
      <gBSDUnit syntacticalLabel="Frame-1" start="23" length="6693" marker="IDR" />
      <gBSDUnit syntacticalLabel="Frame-2" start="6716" length="6177" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-3" start="12893" length="6186" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-4" start="19079" length="6177" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-5" start="25256" length="6227" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-6" start="31483" length="6210" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-7" start="37693" length="6251" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-8" start="43944" length="6252" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-9" start "50196" length="6302" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-10" start="56498" length="6884" marker="I" />
      <gBSDUnit syntacticalLabel="Frame-11" start="63382" length="6288" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-12" start="69670" length="6431" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-13" start="76101" length="6470" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-14" start="82571" length="6497" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-15" start="89068" length="6519" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-16" start="95587" length="6525" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-17" start="102112" length="6548" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-18" start="108660" length="6546" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-19" start="115206" length="7076" marker="I" />
      <gBSDUnit syntacticalLabel="Frame-20" start="122282" length="6575" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-21" start="128857" length="6581" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-22" start="135438" length="6558" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-23" start="141996" length="6645" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-24" start="148641" length="6675" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-25" start="155316" length="6669" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-26" start="161985" length="6603" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-27" start="168588" length="6605" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-28" start="175193" length="7200" marker="I" />
      <gBSDUnit syntacticalLabel="Frame-29" start="182393" length="6529" marker="P" />
      <gBSDUnit syntacticalLabel="Frame-30" start="188922" length="6464" marker="P" />
  </dia:Description>
</dia:DIA>
```

Figure 19. gBSD representation of the original H.264 bitstream

3.2.3 Implementing the Adaptation Module

The adaptation module itself can be seen as the major accomplishment for temporal adaptation as it avoids all the traditional transcoding approach of cascaded decoding and

re-encoding or selecting a partial stream from a pre-defined multiple video streams based on user/device requirement.

Inside the adaptation module, adaptation is thus performed in the following two steps:

1. Transformation of the gBSD characterizing the media bitstream via XSLT, and
2. Generation of the adapted bitstream using the transformed gBSD

Transformed gBSD Generation

MPEG-21 DIA does not standardize the transcoding of the bit stream or which transformation language should be used. XSLT has therefore been used to transform gBS Description. The end user's preference or device requirement in terms of frame rate is considered as an external input for transformation decision taking mechanism.
As described in section 3.1.2, a generic XSLT style sheet is first drafted with the adaptation template rules and output document format. All the source nodes are selected by applying the following template rule to generate the corresponding source tree:

```
<xsl:template name="tplAll" match="node()|@*">
      <xsl:copy>
            <xsl:apply-templates select="node()|@*"/>
      </xsl:copy>
</xsl:template>
```

A frame skip pattern is implemented in the template based on the original frame rate and required frame rate, which is matched against the nodes in the source tree of the gBSD. The frame skip mechanism enables consistent frame dropping pattern for a target frame rate in between 1 to 29fps, if original encoding frame rate is 30fps. For example, for any Digital Item, if the target frame rate is 10fps then the selected frame numbers to be dropped will be same for all DIs. The pseudo-code of this strategy is shown below:

```
FrameSkip(newFrameRate,oldFrameRate)
{
 a=1, b=1;
 temp = newFrameRate/oldFrameRate;
 if(newFrameRate<oldFrameRate && newFrameRate!=0)
 while(a<oldFrameRate)
 {
   copyFrame(a);
   a = ceiling(1+b*temp);
   b++;
 }
}
```

For the frame skip, in general case it is assumed that the new rate would be less than the old frame rate and greater than zero. If the new rate is greater or equal to the old frame rate then the source tree becomes the resulting tree and copies all nodes. XSLT template rule implementing the above pseudo code for frame skip is given below:

```
<xsl:param name="newframerate" select="0"/>
<xsl:param name="oldframerate" select="0"/>
<xsl:variable name="tempx" select="($oldframerate div $newframerate)"/>

<xsl:template name = "loop" match="gbsd:gBSDUnit[@syntacticalLabel]">
   <xsl:param name="a" select="1"/>
   <xsl:param name="b" select="1"/>
   <!—new_rate is less than old_rate and new_rate is not zero -->
   <xsl:if test=
      "($newframerate) &lt; ($oldframerate) and ($newframerate) != 0">
     <xsl:if test="(substring-after(@syntacticalLabel,'-') = $a)">
        <xsl:copy>
            <xsl:apply-templates select="node()|@*"/>
        </xsl:copy>
     </xsl:if>
     <xsl:if test="($a) &lt;= ($oldframerate)" >
       <!-- loop condition check -->
       <xsl:call-template name="loop">
         <xsl:with-param name="a"
```

```
            select="ceiling(1 + (($b)*($tempx)))" />
          <xsl:with-param name="b" select="($b)+1" />
        </xsl:call-template>
      </xsl:if>
  </xsl:if>
<!—new_rate is greater than or equal to old_rate-->
  <xsl:if test="($newframerate) &gt;= ($oldframerate)">
      <xsl:copy>
            <xsl:apply-templates select="node()|@*"/>
      </xsl:copy>
  </xsl:if>
</xsl:template>
```

NAL header information for the H.264 stream in gBS Description is copied to the resulting gBSD applying following template rule:

```
<xsl:template name="NAL"
      match="gbsd:gBSDUnit[@syntacticalLabel = 'NALUnit']">
      <xsl:copy>
            <xsl:apply-templates select="node()|@*"/>
      </xsl:copy>
</xsl:template>
```

The second step towards adapted gBSD generation was implementing the XSLT processor. Towards this end, the MSXML version 3.0 is incorporated to parse and process the gBSD. The implementation steps for processing the gBSD are divided into the following :

1. COM initialization
2. Loading of XML and Stylesheet files
3. Initializing the XSLT processor
4. Perform transformation
5. Output the transformation to adapted gBSD

A type library for MSXML (i.e. msxml3.dll) is first imported. All the interfaces that need to be used are declared which includes IXMLDOMDocument, IXSLTemplate and IXSLProcessor. Sample code is given below:

```
CoInitialize(NULL);
MSXML2::IXMLDOMDocumentPtr
        pXml(MSXML2::CLSID_DOMDocument);
MSXML2::IXMLDOMDocumentPtr
        pXslt(CLSID_FreeThreadedDOMDocument);
IXSLTemplatePtr pTemplate(CLSID_XSLTemplate);
IXSLProcessorPtr pProcessor;
```

The next step was to load the input gBSD and XSLT style sheet i.e. XSL file. A sample code is given below:

```
variant_t vResult;
try
 {
  vResult = pXml->load(_bstr_t(iFileXml));
  vResult = pXslt->load(_bstr_t(iFileXsl));
 } catch(_com_error &e) {
   exit(-1);
}
```

The XSL Template object is used to cache the input XSLT style sheet and to transform the gBSD file. An XSL Processor object is created for the transformation. Sample code is given below:

```
try
 {
   vResult = pTemplate->putref_stylesheet(pXslt);
   pProcessor = pTemplate->createProcessor();
 } catch(_com_error &e) {
     exit(-1);
}
```

For the transformation output, a buffer is set by creating a Stream object and storing its interface pointer in a variable of the IStream type. In a later code, the processor is assigned its XML file that will be transformed depending on, and assigning values from, the new frame rate and old frame rate. Finally, the transformation procedure was called. Sample code is given below:

```
CreateStreamOnHGlobal(NULL,TRUE,&pOutStream);
 pProcessor->put_output(_variant_t(pOutStream));
 // attach to processor XML file we want to transform,
 // add parameter,  and  // do the transformation
  vResult = pProcessor->put_input(_variant_t((IUnknown*)pXml));
  /*here you send the value for NEWFRAMERATE*/
 pProcessor->addParameter(_bstr_t("newframerate"),
                  _variant_t(newFrameRate),_bstr_t(""));
 /*here you send the value for OLDFRAMERATE*/
 pProcessor->addParameter(_bstr_t("oldframerate"),
                  _variant_t(oldFrameRate),_bstr_t(""));
 pProcessor->transform();
```

Upon successful transformation, output buffer was saved to the file output i.e. adapted gBSD.

Generation of Adapted Bitstream

Even though the MPEG-21 standard specifies DIA tools to aid the adaptation process, the implementation or framework for adaptation engines is not stipulated. The gBSDtoBin process mentioned in section 2.2.4 is implemented inside the adaptation engine. Adapted bitstream generation module consists of following 3 steps:

1. Initialize and parse adapted gBSD
2. Extract parsed gBSD information from video stream
3. Write adapted video stream to file

File control structure for streams object iFileStream is used to access the input file stream. Each line is scanned from the adapted gBSD. `syntacticalLabel`, `start` and `length` value is tokenized by each field according to its size from the line buffer. Sample code is given below:

```
char *match_NAL = "NALUnit", *match1 = "Frame-", *match2 = "start", *match3 =
"length";
fscanf(iFileStream, "%s[^\n]", line);
if(!((strstr(line,match_NAL)==NULL||strstr(line,match1)==NULL) &&
strstr(line,match2)==NULL && strstr(line,match3)==NULL))
{
    char *starttok = strstr(line,"start=");
    if(starttok)
    {
            char temp[50];
            strcpy(temp,starttok+7);
            sscanf(temp, "%d", &start);
    }
    char *lengthtok = strstr(line,"length=");
    if(lengthtok)
    {
            char *lengthtokp;
            lengthtokp = strtok(lengthtok+7,"\"");
            sscanf(lengthtokp, "%d", &length);
    }
}
```

Once the `start` and `length` of each `syntacticalLabel` is parsed, adapted H.264 bitstream is processed by discarding gBSD portions corresponding to specific frames and updating information for subsequent dependencies for the transformed H.264 bitstream. Initial buffer for reading each frame data is initialized at 1MB to read and perform

operations in one go. In case of larger frame size, the buffer is resized according to length of the current frame. Sample code is given below:

```
char *buf;
buf = (char *)malloc(1);
readFile rf(iFileString);//iFileString points to input video file
fSize = rf.fileSize;
unsigned long begin = 0;
while(!feof(rf.fin))
{
    if(begin!=start)
    {
            for(begin=0;begin<=(start-1);begin++)
            {
                    fread(buf,1,1,rf.fin);//read upto (start-1)
            }
    }
    buf = (char *)malloc(length);//resize buffer according to length of the current frame
    fread(buf,1,length,rf.fin);//read from start of the frame till end in one go!
    char *b = buf;//point to the buffer
    int written = write(handle,b,length);
    if(written==length) break;
}
free(buf);
```

Final output is the adapted H.264 format compliant bitstream.

3.3 Experimental add-on: Color Adaptation

What we have seen so far is the dynamic temporal adaptation of H.264 bitstream conforming to MPEG-21 DIA. The goal of this thesis was to present an adaptation technique to adapt video streams on demand directly from the bitstream avoiding

cascaded transcoding or pre-coded streams with multiple formats. This goal for temporal adaptation is achieved by applying the design presented above. To experiment the performance of color adaptation in the above adaptation module, an alternative to instantaneous transcoding is followed which produce and store two separate streams for color and monochrome in one single sequence of bitstreams. Upon receipt of the user preference or device requirement, appropriate portion from the combined bitstream is chosen and after temporal adaptation, the content is transmitted to the end user. A drawback of this approach is surely the relatively larger file size for each video and which eventually requires a nearly double storage requirement in the resource server.

For color adaptation, the adaptation architecture works exactly the same way it is described for temporal adaptation conforming to MPEG-21. The only modification is made to the generation of digital item. For this purpose, the raw YUV 4:2:0 input is encoded twice – first for color bitstream and second for monochrome bitstream. For both of the sequence, gBS Description is generated while it was being encoded. Both the bitstreams are thus merged in to a combined bitstream adding an 8-byte separator key to distinguish the separate bitstreams. For this particular implementation, color bitstream is put in the first chunk of the combined bitstream. The gBS descriptions are also put in to a common XML file where the separator key is identified as a gBSDUnit. Figure 20 below illustrates an example of the original and adapted gBSD transformation for color adaptation.

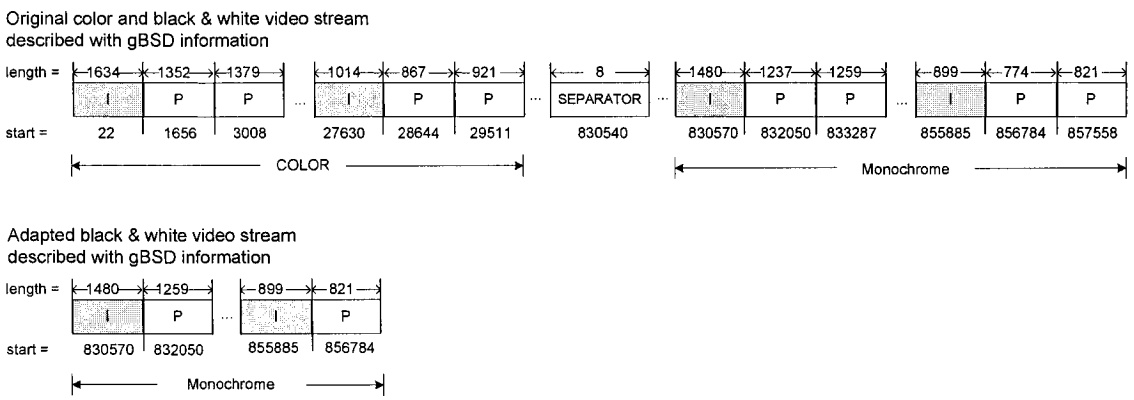


Figure 20. Example of original and adapted gBSD transformation for color adaptation

Sample XML representation of the combined bitstream along with the Separator key is shown in Figure 21.

```
<?xml version="1.0" encoding="UTF-8" ?>
<dia:DIA xmlns="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS" xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dia:Description xsi:type="gBSDType">
      <Header>
            <ClassificationAlias alias="MV4" href="urn:mpeg:mpeg4:video:cs:syntacticalLabels"/>
            <DefaultValues addressUnit="byte" addressMode="Absolute" globalAddressInfo="test.raw"/>
      </Header>
      <gBSDUnit syntacticalLabel="NALUnit" start="0" length="22"/>
      <gBSDUnit syntacticalLabel="Frame-1" start="22" length="6800" marker="IDR"/>
      <gBSDUnit syntacticalLabel="Frame-2" start="6822" length="6069" marker="P"/>
      <gBSDUnit syntacticalLabel="Frame-3" start="12891" length="6197" marker="P"/>
      <gBSDUnit syntacticalLabel="Frame-4" start="19088" length="6082" marker="I"/>
      <gBSDUnit syntacticalLabel="Frame-5" start="25170" length="6245" marker="P"/>
      <gBSDUnit syntacticalLabel="Frame-6" start="31415" length="6138" marker="P"/>
      <gBSDUnit syntacticalLabel="SEPARATOR" start="1782748" length="8"/>
      <gBSDUnit syntacticalLabel="NALUnit" start="1782756" length="22"/>
      <gBSDUnit syntacticalLabel="Frame-1" start="1782778" length="5827" marker="IDR"/>
      <gBSDUnit syntacticalLabel="Frame-2" start="1788605" length="5305" marker="P"/>
      <gBSDUnit syntacticalLabel="Frame-3" start="1793910" length="5443" marker="P"/>
      <gBSDUnit syntacticalLabel="Frame-4" start="1799353" length="5329" marker="I"/>
      <gBSDUnit syntacticalLabel="Frame-5" start="1804682" length="5444" marker="P"/>
      <gBSDUnit syntacticalLabel="Frame-6" start="1810126" length="5371" marker="P"/>
  </dia:Description>
</dia:DIA>
```

Figure 21. Sample XML representation of the combined bitstream

Upon request from the user, the system selects a color or monochrome bitstream from the combined video file based on the separator key from the generic Bitstream Description. This particular bitstream is thus sent to the adaptation engine for frame rate adaptation along with its appropriate gBSD portion.

3.4 Experimental add-on: Spatial Adaptation

To experiment the performance of spatial adaptation in the above adaptation module, an alternative general instantaneous approach is followed. Figure 22 portrays the spatial adaptation scenario.

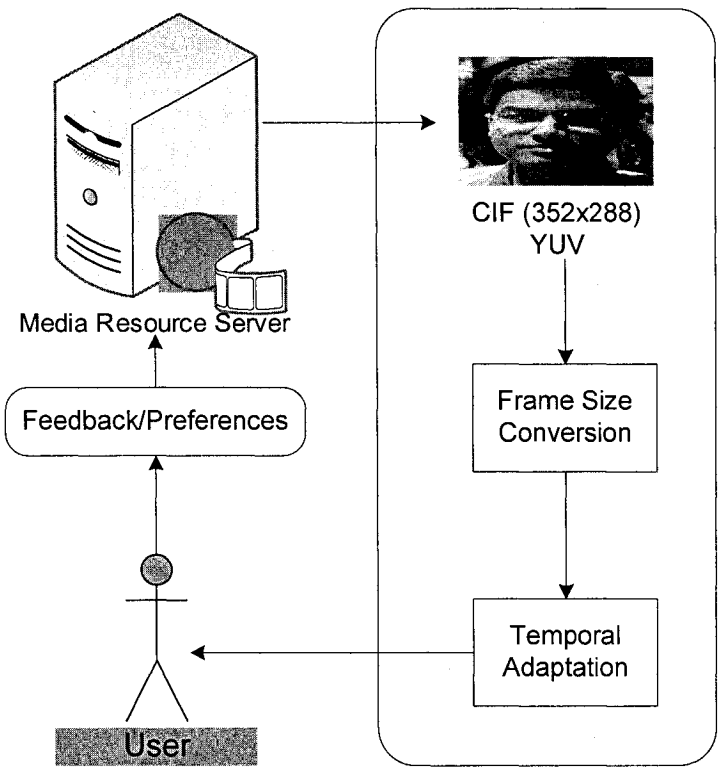


Figure 22. Spatial Adaptation

Upon receipt of the user preference or device requirement, the appropriate raw video file (i.e. YUV) is selected from the media resource server and is sent for frame size conversion (e.g. CIF to QCIF or CIF to SQCIF). Once the conversion is made, this converted raw video input fuels the temporal adaptation scheme mentioned in this thesis. A frame size conversion scheme designed for this purpose is shown in Figure 23. Sample code for CIF to QCIF and CIF to SQCIF is given in Appendix A.

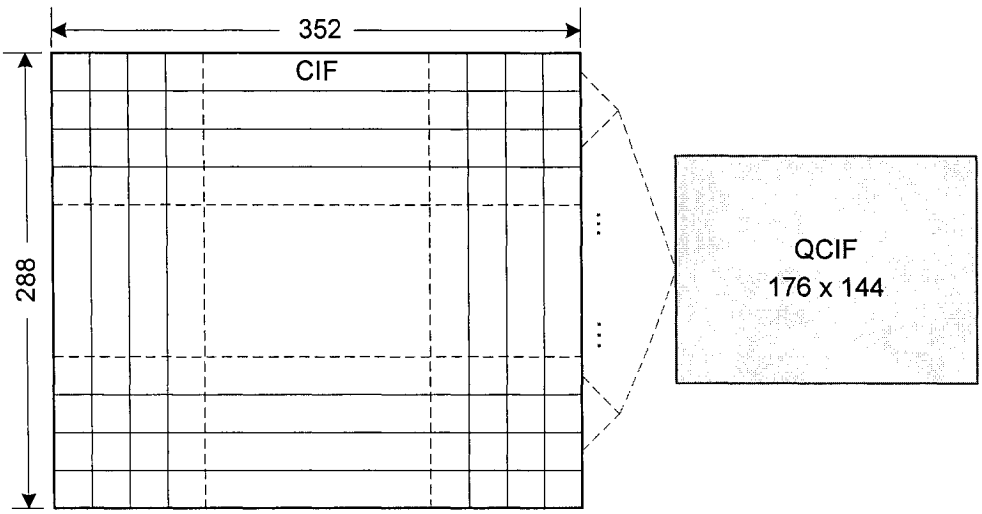


Figure 23. Frame Size Conversion (CIF to QCIF)

A drawback of this approach is that the media resource server must contain the raw video files which are of large file sizes. After transmission, there may be need to deploy some caching mechanism to avoid the same computation for serving another user with the same preference.

3.5 Limitation of MPEG-21 Adaptation applying H.263

The H.263 video coding standard [28], by the ITU-T, is a low bitrate video encoding scheme for video-conferencing and video-telephony applications. H.263 was developed as an improvement based on experience from H.261 [29], the previous ITU-T standard for video compression. At the earlier stage of this research, H.263 encoded video was chosen to apply MPEG-21 Digital Item Adaptation. Architectural design shown in Figure 24 was applied for the adaptation. As we see from the figure, instead of generating the gBSD from the encoder, it was generated directly from the encoded video since bitstream structure is defined for H.263 in the specification. So, Picture Start Code (PSC) of each frame is identified in the "BintogBSD" processor by parsing the encoded bitstream and then start and length of each frame is stored in the XML conforming to MPEG-21. The XML is then processed to generate the adapted XML using XSLT processor and XSL Style sheet. Finally, adaptation engine adapts the original compressed H.263 video based on the adapted XML.

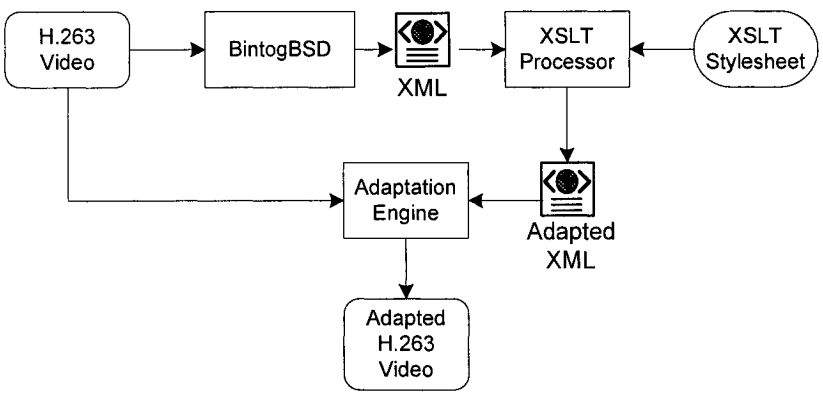


Figure 24. Adaptation Architecture applying H.263 video

The limitation of applying H.263 in this adaptation architecture is that H.263 does not allow multiple reference frames for motion compensation. In H.263 encoder, the previous transmitted frame is subtracted from the current frame so that only the difference or residue needs can be encoded. For motion compensation, the H.263 encoder estimates

areas of the previous frame which have moved to in the current frame and compensates this movement. In the decoder, the difference values are added to reconstruct the current frame from the previous frame. Motion vector information is used to pick the correct area which is the same reference area that was used in the encoder. The result is a reconstruction of the original frame. So, for temporal adaptation, if a frame is discarded then the subsequent frame(s) can not be re-generated correctly which results in non-perceivable video quality. Figure 25 shows an example of adapted video quality for a H.263 video (miss_america, QCIF).

Original, 30 fps *Adapted, 15 fps*

Figure 25. H.263 Adapted Video Quality

3.6 Summary

To perform the adaptation, we need to generate the high level structure of the bitstream (i.e. gBSD) while it is being encoded. The output from the encoder is thus the compressed bitstream (Video(C)) and the gBSD in the form of XML. Inside the transcoder, gBSD is transformed to adapted gBSD (XML(A)) to infer the required bitstream representation for the desired frame rate by means of Extensible Stylesheet Language Transformation (XSLT). The adaptation specification is formed in a generic style sheet which is fed to the XSLT processor to transform the gBSD. An adaptation engine is thus designed where both the original H.264 bitstream and the adapted gBSD is placed from which the adapted H.264 bitstream (Video(C+A)) is generated and served. Figure 26 shows the complete architecture of the H.264 Video Adaptation.

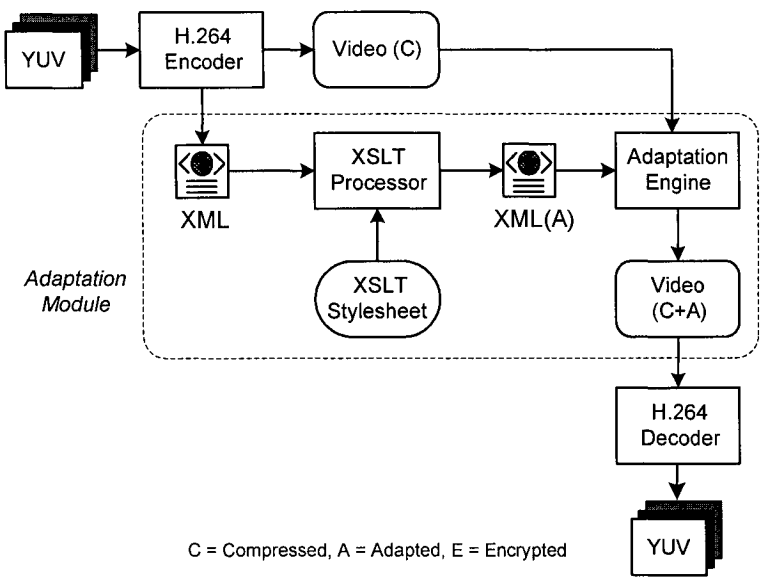


Figure 26. Complete Architecture of the H.264 Video Adaptation

Chapter 4

Prototype and Performance Evaluation

In this thesis a test application is implemented to experiment the applicability and performance of the designed adaptation framework for a contemporary scenario. The implemented system is divided into two disjoint subsystems imitating the server-client construction. The test application consists of the following:

1. Adaptation Server – for pre-recorded and live adaptation
2. Client module – to request and play video stream

The adaptation server is responsible for the following:

i. Accept client connection

ii. Accept client preferences upon successful connection setup

iii. Adapt requested content according to preferences

iv. Send adapted video file

The client module is responsible for the following:

i. Connect to the available adaptation server in the network

ii. Send video preferences

iii. Request for specific video from the received file list or the live stream

iv. Decode and play received video

4.1 Design of the test application

4.1.1 Use cases

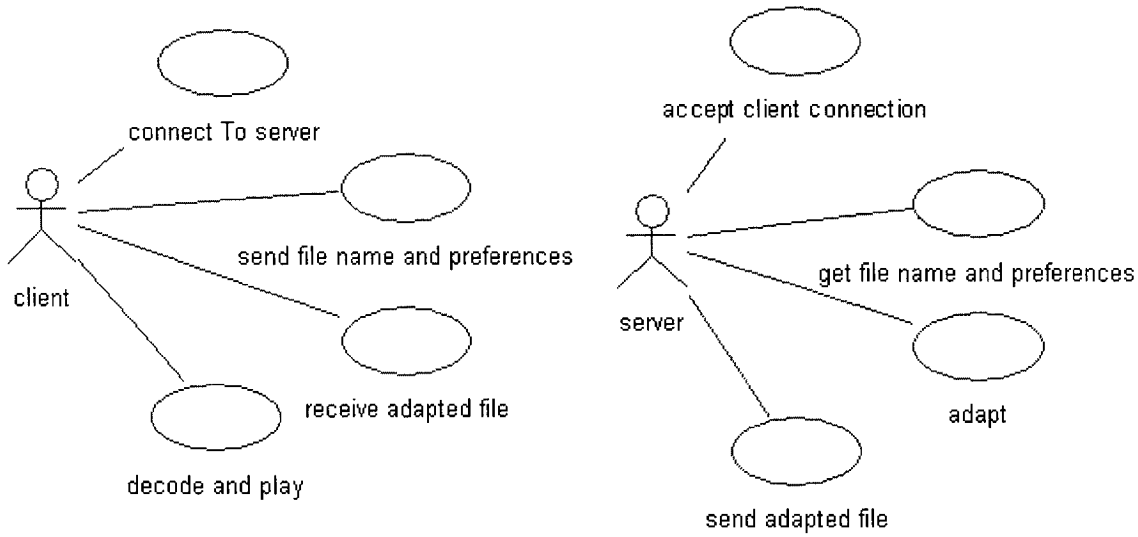


Figure 27. Use case diagram of test application

The use cases are described below:

Connect To server
Actor: client
Description:
1. Connect to the available server
2. Receive file list

Send file name and preferences
Actor: client
Description:
1. Choose file – pre-recorded or live
2. Send preference to the server

Receive adapted file
Actor: client
Description:
1. Receive adapted file from server
2. Close connection

Decode and play
Actor: client
Description:
1. Decode received file
2. Play decoded file

Accept client connection
Actor: server
Description:
1. Accept client connection request
2. Send file list

Adapt
Actor: server
Description:
1. Adapt requested file according to preferences

Get file name and preferences
Actor: server
Description:
1. Receive file name and preferences
2. Search for file

Send adapted file
Actor: server
Description:
1. Send adapted file
2. Close connection

4.1.2 Sequence Diagram

Figure 28 shows the sequence diagram of the client requests and server response of the implemented system. When a user wants to request a video file from the server, the user needs to connect to the resource server first. On successful connection setup, the resource server sends the file list of the available videos. The user can select the file name along with the frame rate, size and color preferences. Preferences are thus sent to the server over the network. Upon successful receipt of the preferences and file name, the resource server initiates the adaptation operation on the video content. The time required to adapt the bitstream depends on the video length, size and frame rate. Thus, a variable delay is obvious for the server while serving the request. When the file is adapted, the adapted video is sent to the client.

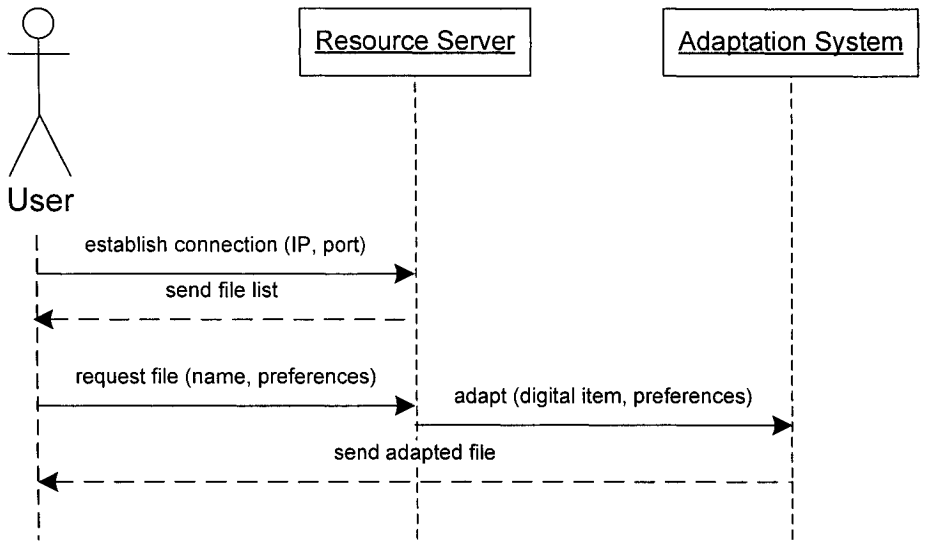


Figure 28. Sequence diagram of the client requests and server response

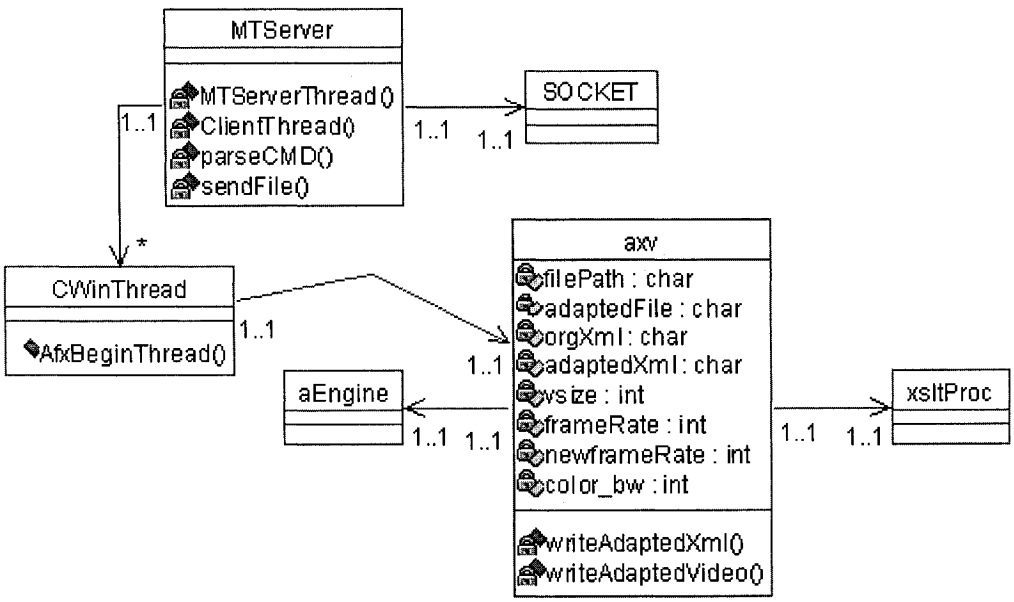
4.1.3 Class diagram

As already mentioned, the implemented system is divided in to two disjoint subsystems imitating the server-client construction. The first one is the streaming server where the adaptation module is embedded and responsible for serving adapted video streams on demand. The second one is the client subsystem, which is responsible for connecting to the server over the network using TCP/IP and for requesting video streams.

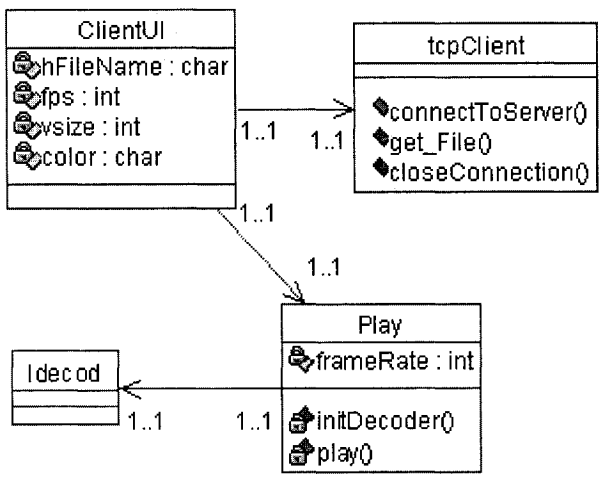
Figure 29A shows the class diagram of the server side. Whenever a client is connected a new thread is initialized to serve that client by MTServer. Once the adaptation is complete and the file is sent, server closes the connection. For a specific client thread, the server opens a socket to handle connection and send/receive files. Once the file choice is received along with preferences, the client thread in the server calls axv to adapt the file from its (transformed) gBSD.

Figure 29B shows the class diagram of the client side. ClientUI uses the tcpClient class object to connect to server and handle connection. Once the adapted file is received it

calls decode class object to decode and then play the decoded file where "play" class is responsible for that.



A. Server



B. Client

Figure 29. Class diagram of the implemented system

4.2 Design of Live adaptation

The aforementioned design is applied to the live demo with trivial modifications. We know that, gBSD and XSLT is designed and standardized for static items. To offer live adaptation, video streams need to be processed as small clips in a pseudo live fashion as shown in Figure 30.

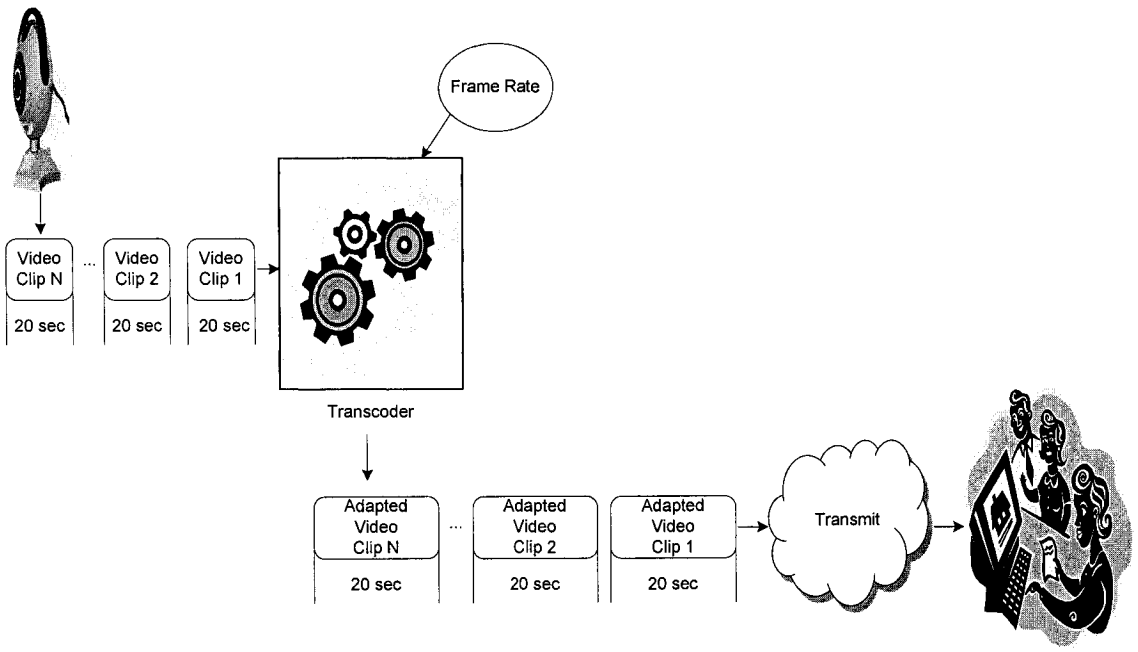


Figure 30. Adapting Live Video stream

Video clips might need type conversion and pre-process for specific resolution and frame rate before putting it in the adaptation module to encode, adapt and transmit. The streaming will require an obvious fixed delay for pre-processing and adaptation of clips. In addition, a longer initial delay is usually acceptable in this type of live adaptation. The sequence diagram for live streaming is shown in Figure 31.

The user first establishes connection with the live server and sends preferences like that of pre-recorded videos. While live streaming server receives the request, it adapts the live clips according to user requirements and sends the adapted clips until the end of the live

stream. As soon as the live capture stops, the server sends a token to the client informing it about the end of the live stream.

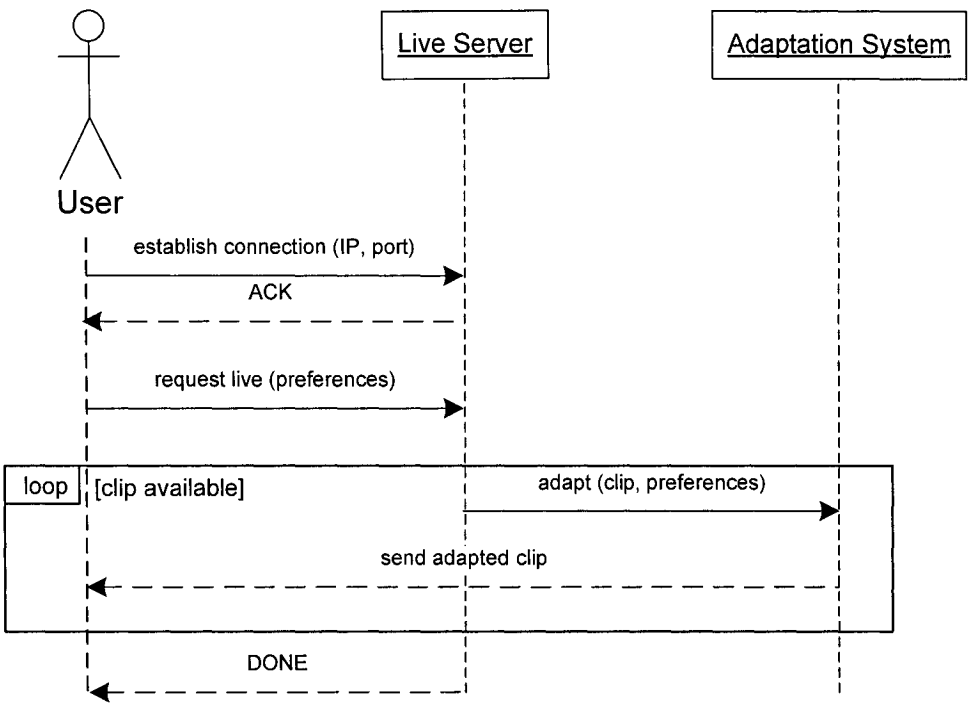


Figure 31. Sequence diagram of the live system

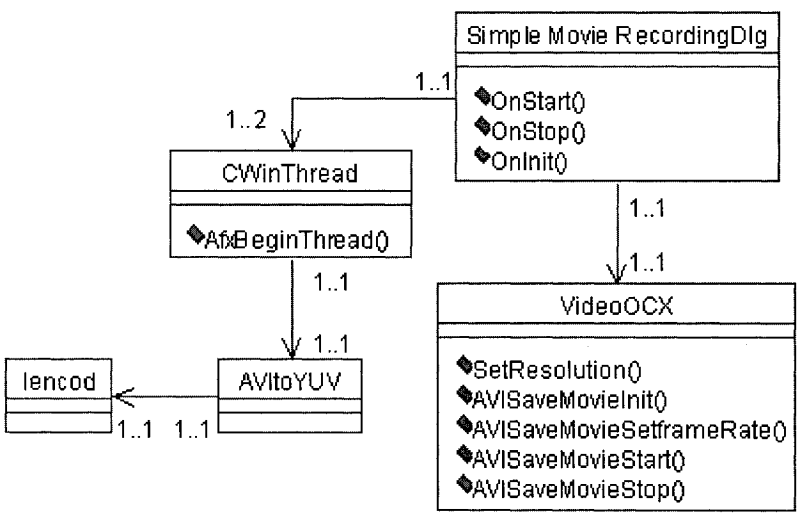


Figure 32. Class diagram of live stream processing

The same server is used for live adaptation. Instead of sending one file, the server sends continuous adapted video clips for live requests. Figure 32 above shows the class diagram of live video capture and processing. "Simple Movie RecordingDlg" uses "VideoOCX" class object to capture a clip every 20 seconds and opens a thread to convert it from AVI to YUV. Once done, the second step is to encode the YUV file to H.264 video and generate its gBSD; "lencod" class is responsible for it. Meanwhile, capture and processing of consecutive clips proceeds simultaneously.

4.3 Performance Evaluation

4.3.1 Testing Environment Setup

Performance evaluation was conducted at DISCOVER Lab in University of Ottawa. For the testing, the media resource/streaming server had following configuration:

CPU	:	Pentium 4, 3.4 GHz
Memory	:	512 MB
O/S	:	Windows XP Professional with SP2

In the media server, there were 2 uncompressed (YUV 420) sample videos - *corvette* and *airshow*. Each of the video files had 600 frames of frame size CIF (352 × 288).

For pre-coded video adaptation, the uncompressed video is encoded first which generates the H.264 encoded file and its gBSD. These are kept in the server. If a client requests a frame size less than CIF format (i.e. QCIF or SQCIF), then the uncompressed video is converted for its frame size (CIF to QCIF or CIF to SQCIF) and encoded. In both cases, the encoded video sequence comprises color and monochrome video bitstream.

Figure 33. Adaptation and Streaming Server

When the server is started (Figure 33), it starts listening to port 3333. For an incoming client request, the server initiates a new thread and sends "File.lst". "File.lst" consists of the available video file names. Server waits for client preferences. When received, client request/preferences are stored in a buffer and the buffer is parsed as follows to retrieve the information:

1. VSIZE : Preferred Size of Video (CIF = 1, QCIF = 2, SQCIF = 3)
2. COLOR : Color preference (color = 1, monochrome = 2)
3. FPS : Preferred Frame Rate (1 to 29)
4. FILE : Video file name

After parsing each of the preferences, client thread initiates the adaptation module "axv". Finally, adapted video "adapted.264" is sent to the client and server closes the connection. The server was tested for 3 simultaneous client requests with different preferences in terms of color, frame size, frame rate and video file over the Local Area Network (LAN).

Figure 34. Live Capture Module

For live streaming, on the server side, both "adaptation and streaming server" and "Live Capture Module" (Figure 34) need to be started. "Set Driver" button is used to select the driver of the webcam. "Init" initiates video through webcam. Once the "Start" button is pressed, live capture starts. A new video clip is saved in the server after every 20 seconds until "Stop" button is pressed. Each of the clips is then converted to YUV format, from

which Digital Item is generated and stored. For live stream requests, once the preference is received (i.e. frame rate), server starts sending adapted live video clips starting from the most recent one. To ensure live characteristics, this particular test system allows only temporal adaptation of color video, where the capture frame size is fixed to SQCIF and the capture frame rate is 15 fps. The system has been tested for 2 simultaneous clients, requesting live stream with different frame rate from the same LAN.

On the client side, user enters the IP of the media resource server (Figure 35) and connects to the available server.

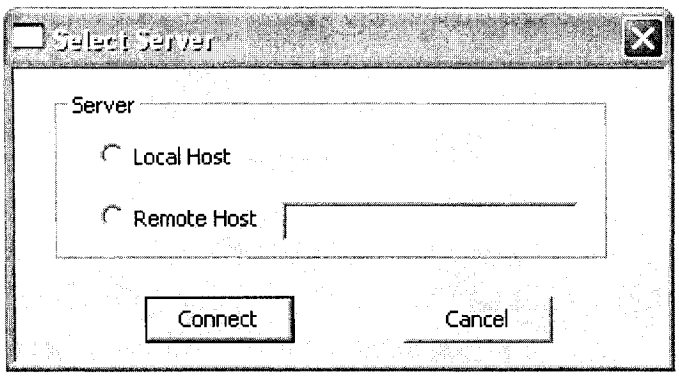


Figure 35. Client Connection Setup

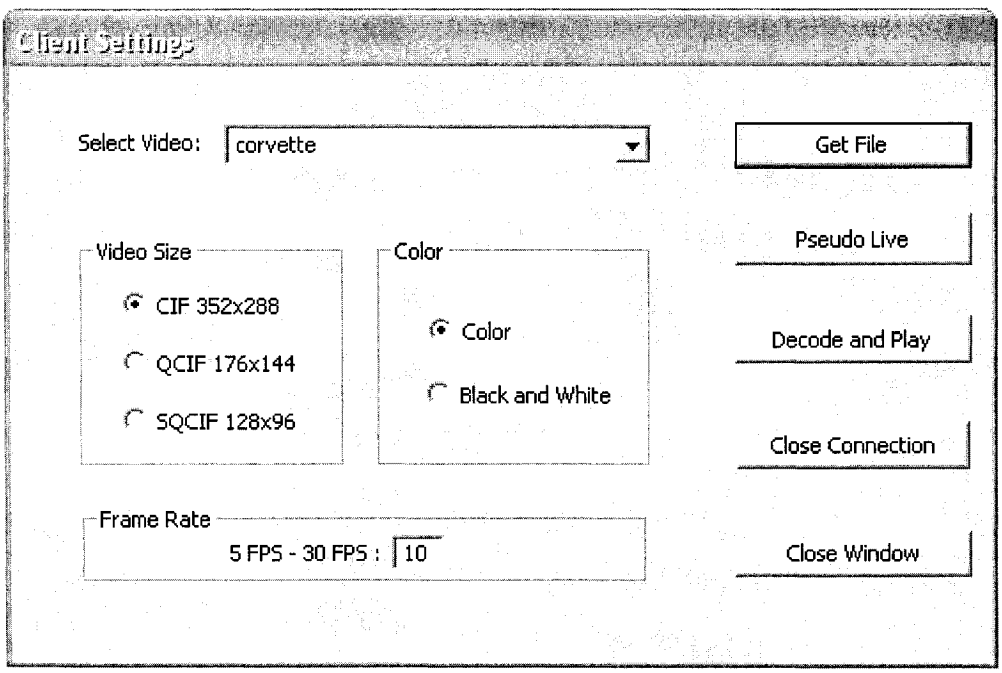


Figure 36. Client Settings

After connection is established, client receives "File.lst". The client program parses the file list and gives a list of available video files in a drop down menue in Client Settings (Figure 36). User sets other preferences (i.e. Video Size, Color, Frame Rate) and requests the file by pressing "Get File". Client then waits for the server to adapt the video and listens for incoming data. After receiving the adapted file, user has to press "Decode and Play" to play the received video.

For live streaming, user setups the connection as before. In Client Settings, user has to enter desired frame rate (1 to 14) regardless of other preferences and press "Pseudo Live". The client program will then start receiving the adapted live clips after fixed initial delay (usually 40 seconds) and will automatically decode and play the stream. Figure 37 shows a snapshot of the video player on the client side.

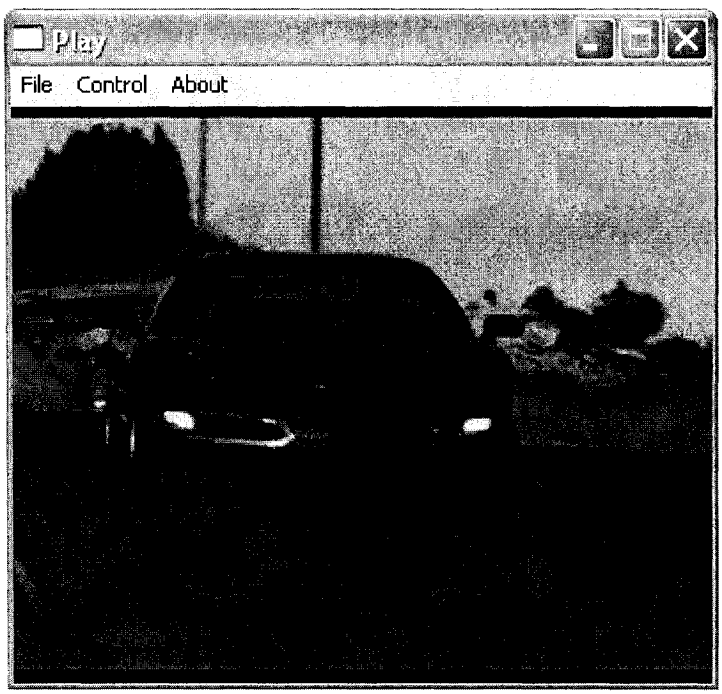


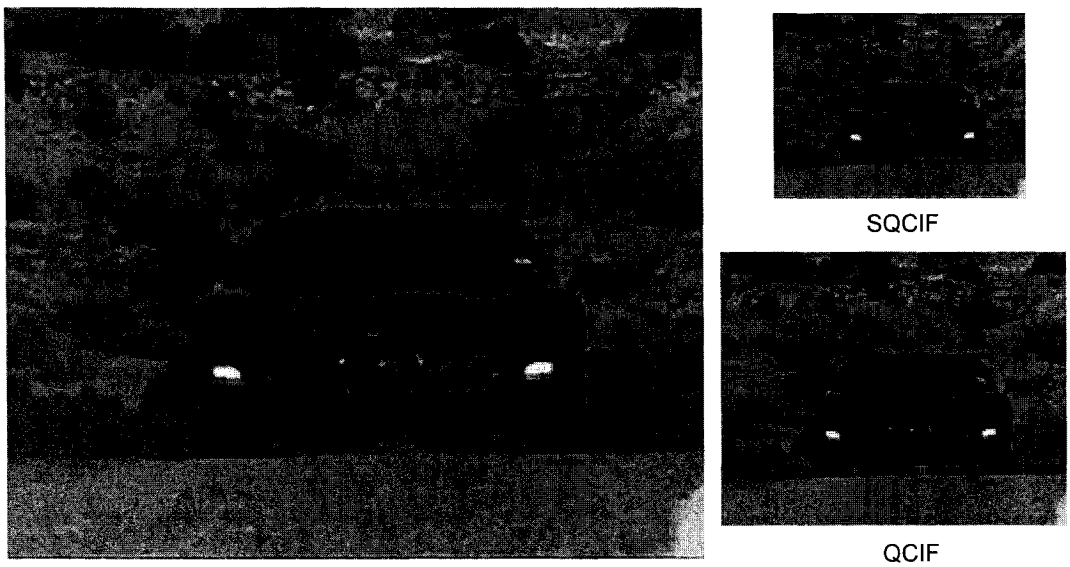
Figure 37. The Player

4.3.2 Frame size conversion performance

From Table 2, the time required to convert frame size from CIF to QCIF and CIF to SQCIF using the implemented converter program is evident. This is the time required for any YUV 4:2:0 video sequence.

Table 2. Frame size conversion performance for sample video (corvette)

CIF (352 x 288) → QCIF (176 x 144)	CIF (352 x 288) → SQCIF (128 x 96)
0.97 sec	0.83 sec



Corvette (600 frames)

4.3.3 Digital Item generation performance

Digital Item generation is one of the major tasks. The ITU-T reference software implementation of H.264 encoder is modified with gBSD generation functionality.

Table 3. Digital Item generation performance

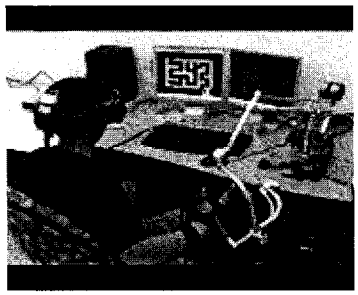
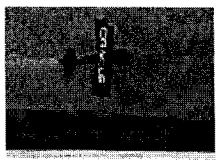
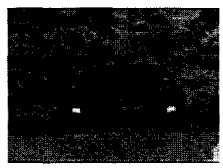
File: Haptic	Resolution	Time
 Color only	CIF	217 sec (@2.76 fps)
	QCIF	57 sec (@10.53 fps)
	SQCIF	25 sec (@24.0 fps)
 Combined Stream	CIF	435 sec (@2.75 fps)
	QCIF	112 sec (@10.71 fps)
	SQCIF	49 sec (@24.49 fps)
Frame Rate: 30 fps, Intra Period: 9, Total frames: 600		

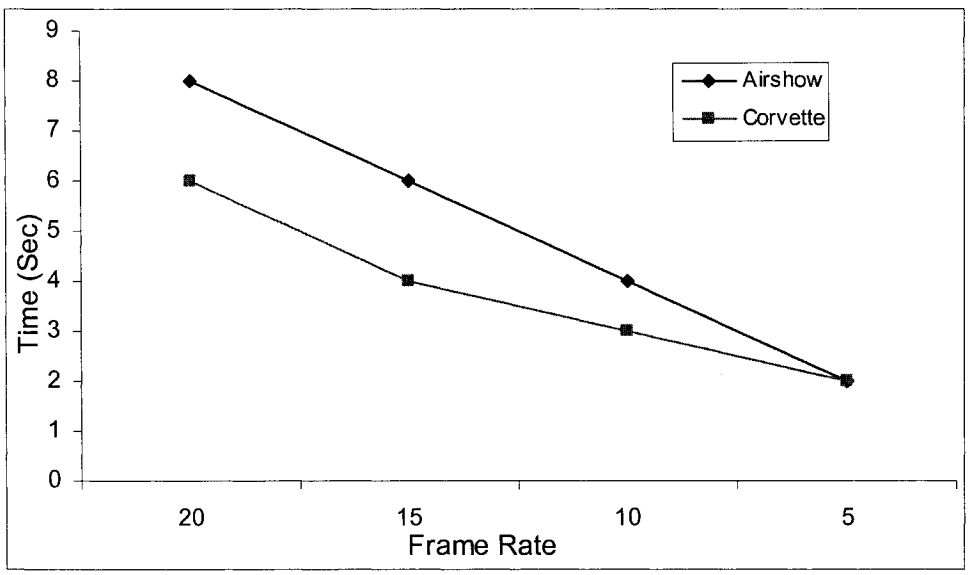
Table 3 shows the time required to encode YUV format to H.264 and generate corresponding gBSD for one single stream and for a combined stream (color and monochrome video in one file using separator key). For combined video, the time required is less compared to that of color because monochrome streams lack chroma components in the bitstream and thus requires less time to process. But time to merge the bitstream and gBSD for combined video is included in the calculation.

4.3.4 Temporal adaptation performance for pre-recorded video

Table 4 and Figure 38 presents the temporal adaptation performance of the adaptation module for two test sequences (Airshow and Corvette).

Table 4. Temporal adaptation performance chart

	Resolution	New Frame Rate	Time (Airshow)	Time (Corvette)	
 Airshow	CIF	5 fps	5 sec	3 sec	 Corvette
		10 fps	9 sec	6 sec	
		15 fps	13 sec	8 sec	
	QCIF	5 fps	2 sec	2 sec	
		10 fps	4 sec	3 sec	
		15 fps	6 sec	4 sec	
	SQCIF	5 fps	1 sec	1 sec	
		10 fps	2 sec	2 sec	
		15 fps	4 sec	3 sec	
Original Frame Rate: 30 fps, H.264 Profile: High (100) Intra Period: 9, Total number of frames: 600					



Airshow, Corvette : Resolution – QCIF, Total frames: 600

Figure 38. Temporal adaptation performance graph

4.3.5 Temporal adaptation performance for live video

Sample test adaptation performance of the live adaptation system for a live video session is shown in Table 5:

Table 5. Live adaptation performance

Capture Video	Type	AVI to YUV	YUV to DI	Adaptation	Adapted Video
SQCIF	AVI	2 sec	15 sec	3 sec	SQCIF
15 fps					1-14 fps*
Length 20 sec					Length 20 sec

** any frame rate between 1 to 14*

Based on Table 5, we can infer that total initial delay for the video processing before transmission will be 40 seconds. The initial delay is computed as follows:

$$\begin{aligned} &Total\ Initial\ Delay = \\ &Capture\ time + AVItoYUV\ conversion\ time + \\ &YUVtoDI\ generation\ time + Adaptation\ time \end{aligned}$$

Chapter 5

Conclusion & Future Work

5.1 Conclusion

Seamless adaptation and transcoding techniques to adapt the digital content have achieved significant focus to serve the consumers with the desired content in a feasible way. The novelty of this work is evident since the existing literature lacks any other implemented and tested adaptation architecture for dynamic adaptation conforming to MPEG-21 DIA and high performance H.264 video standard. This adaptation approach not only eliminates cascaded transcoding, but also avoids multiple preprocessed bitstream compilation. Final output from the system is H.264 format compliant adapted bitstream which is decodable in any standard H.264 decoder. Multi point adaptation on the encoded bitstream is achievable as long as the bitstream description is preserved for the new adapted stream, provided the description conforms to the generic Bitstream Syntax schema as described in the MPEG framework, or at least follows the transformation sequence provided in the implementation scheme presented here. The achieved benefit of the work is that end devices are free of any transcoding operations. End devices will be able to directly play the adapted content as soon as it is received or streamed. For the above implementation purpose, the adaptation module is embedded along with sub systems in the media resource server, but once the Digital Item is generated, any intermediary node will be able to adapt the bitstream according to its subordinate client's requirement, eventually leading to multi point adaptation design. As a whole, seamless access to multimedia content in terms of video is achieved impeccably. Live adaptation is challenging because one has to perform all the operations in a compressed domain complying with explicit format specification. By the appropriate implementation of the aforesaid framework, this challenge is now achievable, applicable and successful.

5.2 Future Work

Based on the current adaptation framework, the big picture incorporates the variation of choices that a user can request (frame rate, color and size). So the best possible solution is to "encode once, and play any time, anywhere, anyhow you want". In this purpose, investigation to apply resolution and color adaptation in the compressed domain and from the content's gBSD is needed. The only hurdle towards this triumph is that H.264 standard is under development. Modifications of the standard, error corrections, revisions are still in progress.

Inclusion of DIA Usage Environment Description Tools, Terminal and Network QoS tools for adaptation will provide user characteristics, terminal capabilities, network characteristics and natural environment characteristics towards an automatic adaptation decision taking mechanism. These tools will provide descriptive information about the various properties of the usage environment, and will provide means to trade-off various adaptation parameters with respect to quality so that an adaptation strategy can be formulated and optimal adaptation decisions can be made.

MPEG (Moving Picture Experts Group) and VCEG (Video Coding Experts Group) that engineered H.264 AVC, is now extending this standard with Scalable Video Coding (SVC) technology. The SVC extension to Advanced Video Coding will add spatial, temporal and quality scalability to further enable advanced video coding usage in a wide variety of applications, particularly including highly-heterogeneous environments. Once the working draft is finalized, applying the same adaptation approach presented in this thesis will ease content accessibility in ubiquitous multimedia environment.

Beyond the developments presented in this thesis for adaptation, a framework for secured adaptation, where the adapted content itself would be encrypted ensuring format compliance before and after decryption, has partly been investigated during the course and progress of this research. If macroblocks in each frame are considered as encryptable plaintext and macroblock information is assembled in the gBSD, then an encryption engine placed in the adaptation module can encrypt the macroblocks of a pre-coded or

live stream parsing the gBSD. Other than this, setting permission to perform certain adaptation on the content in the UMA environment is another potentially interesting field of research.

References

[1] S-F Chang and A. Vetro, "Video Adaptation: Concepts, Technologies, and Open Issues," Proc. IEEE, Vol. 93, No. 1, pp. 148 - 158, Jan. 2005.

[2] ISO/IEC 21000-7:2004, Information Technology – Multimedia Framework – Part 7: Digital Item Adaptation.

[3] www.chiariglione.org/mpeg/standards/mpeg-21/mpeg-21.htm

[4] W3C Consortium. "XSL Transformations (XSLT) Version 1.0", W3C Recommendation (1999). [Online]. Available: http://www.w3.org/TR/xslt

[5] P. Cimprich. (2003) Streaming Transformations for XML (STX) Version1.0. Working Draft. [Online]. Available: http://stx.sourceforge.net/documents/

[6] L. Sahafi, T.S. Randhawa, and R.H.S.Hardy, "Context-based complexity reduction of H.264 in video over wireless applications," IEEE 6th Workshop on Multimedia Signal Processing, pp. 23-26, 2004.

[7] http://www.apple.com/quicktime/technologies/h264/

[8] http://www.itu.int/rec/T-REC-H.263/en

[9] http://www.chiariglione.org/mpeg/standards/mpeg-2/mpeg-2.htm

[10] T. Wiegand, G.J. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Trans. on Circuits and Systems for Video Technology, Vol. 13, Issue: 7, pp. 560- 576, Jul. 2003.

[11] S. Wenger, "H.264/AVC over IP," IEEE Trans. on Circuits and Systems for Video Tech., Vol.13, Issue 7, pp. 645-656, 2003.

[12] C. Gomila and P. Yin, "New features and applications of the H.264 video coding standard," in Proc. of Intl. Conf. on Info. Tech.: Research and Education, pp. 6 – 10, 2003.

[13] J. R. Smith, R. Mohan, and C. Li, "Scalable multimedia delivery for pervasive computing," presented at the ACM Multimedia Conf., Orlando, FL, 1999.

[14] A. Fox and E. A. Brewer, "Reducing WWW latency and bandwidth requirements by real timer distillation," presented at the Intl. WWW Conf., Paris, France, 1996.

[15] "MPEG-7 Overview v.9," Intl. Standards Org./Int. Electrotech. Comm. (ISO/IEC) JTC 1, ISO/IEC JTC1/SC29/WG11N5525, Mar. 2003.

[16] Exchange protocol based on HTTP extension framework [Online]. Available: http://www.w3.org/TR/NOTE-CCPPexchange

[17] TV-Anytime forum [Online]. Available: http://www.tv-anytime.org

[18] J. Xin, C.-W. Lin, and M.-T. Sun, "Digital video transcoding," Proc. IEEE, vol. 93, no. 1, pp. 84–97, Jan. 2005.

[19] C. Timmerer and H. Hellwagner, "Interoperable adaptive multimedia communication," IEEE Multimedia, Vol.12, Issue 1, pp. 74–79, 2005.

[20] K. Taehyun, M.H. Ammar, "Optimal quality adaptation for scalable encoded video," IEEE Journal on Selected Areas in Communications, Vol. 23, pp. 344-356, 2005.

[21] A. Vetro and C. Timmerer, "Digital item adaptation: overview of standardization and research activities," IEEE Trans. on Multimedia, Vol. 7, Issue 3, pp. 418-426, Jun. 2005.

[22] L. Rong and I. Burnett, "Dynamic multimedia adaptation and updating of media streams with MPEG-21," First IEEE Consumer Communications and Networking Conference, pp. 436-441, 2004.

[23] S. Devillers, C. Timmerer, J. Heuer, and H. Hellwagner, "Bitstream syntax description-based adaptation in streaming and constrained environments," IEEE Trans. on Multimedia, Vol. 7, Issue 3, pp. 463-470, Jun. 2005.

[24] M.A. Bonuccellit, F. Lonetti, and F. Martelli, "Temporal transcoding for mobile video communication," Second Annual Intl. Conf. on Mobile and Ubiquitous Systems: Networking and Services, pp.502-506, 2005.

[25] F. D. Vito, T. Ozcelebi, R. Civanlar, A. M. Tekalp, and J. C. D. Martin, "Rate Control For GOP-Level Rate Adaptation in H.264 Video Coding," Intl. Workshop on very low bit-rate video coding, Italy, 2005.

[26] I. Shin, Y. Lee, and H. Park, "Motion estimation for frame-rate reduction in H.264 transcoding," in Proc. of Second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, pp.63-67, 2004.

[27] http://ftp3.itu.ch/av-arch/jvt-site/reference_software/

[28] ITU-T/SG15. Recommendation H.263, video coding for low bitrate communication, 1996.

[29] ITU-T. Recommendation H.261, video codec for audio visual services at 64 - 1920 kbit/s, 1993.

Appendix A - Frame size conversion code

```
/*****************************************************
*       Author          : Razib Iqbal                *
*       Email           : riqbal@discover.uottawa.ca  *
*       Last update     : June 10, 2005              *
*****************************************************/
/*****************************************************
Type: .cpp
Name: cif2sqcif.cpp

Description: converts *.cif to *.qcif & *.sqcif

Input:
          1. File input: original .cif file (without extension)
          2. User input: none

Output:
          1. output: .qcif & .sqcif
*****************************************************/

#include "stdafx.h"

int handle1,handle2;

unsigned char yBuff_CIF[288][352];
unsigned char cbBuff_CIF[144][176];
unsigned char crBuff_CIF[144][176];

unsigned char yBuff_QCIF[144][176];
unsigned char cbBuff_QCIF[72][88];
unsigned char crBuff_QCIF[72][88];

unsigned char yBuff_SQCIF[96][128];
unsigned char cbBuff_SQCIF[48][64];
unsigned char crBuff_SQCIF[48][64];

char infile[80];
char outfile[80];

void writeSQCIF();
void writeQCIF();
time_t t1, t2,t3,t4,elapsed_time;

int _tmain(int argc, _TCHAR* argv[])
{
        strcpy(infile,"corvette");
        t1 = time(NULL);
        writeSQCIF();
        t2= time(NULL);
        elapsed_time = (t2 - t1);
        printf("SQCIF: %d sec",elapsed_time);

        t3 = time(NULL);
        writeQCIF();
        t4 = time(NULL);
        elapsed_time = (t4 - t3);
        printf("QCIF: %d sec",elapsed_time);

        return 0;//bye
}

void writeSQCIF()
{
        handle1 = open(infile,O_RDONLY|O_BINARY);// enter the input file name here
```

```
        if(handle1==-1)
        {
                printf("error: *read* failed.\n");
                exit(1);
        }

        strcpy(outfile, infile);
        strcat(outfile, ".sqcif");
        handle2  =  open(outfile,O_TRUNC|O_CREAT|O_WRONLY|O_BINARY);//  output  file  name;
default: test.sqcif

        if(handle2==-1)
        {
                printf("error: *write* failed.\n");
                exit(1);
        }

        int bytes=1,point=1;
        int i=0,j=0,p=0,q=0;

        printf("\nWriting to File... ");
        while(1)
        {
                point++;

                bytes=read(handle1,yBuff_CIF,288*352);        // Y Plane
                bytes=read(handle1,cbBuff_CIF,144*176); // U Plane
                bytes=read(handle1,crBuff_CIF,144*176); // V Plane

                if(point%5==0) printf(".");

                if(bytes==0)
                {
                        printf("Done!\n\n");
                        break;
                }
                p = 0;

                // i for height; j for width
                // p for height; q for width

                /* for Y plane */
                for(i=0;i<288;i++)
                {
                        if(i%3==1)
                        {
                                q=0;
                                for(j=0;j<352;j++)//j for width
                                {
                                        if(j%11==0||j%11==3||j%11==6||j%11==9)
                                        {
                                                yBuff_SQCIF[p][q]=yBuff_CIF[i][j];
                                                q++;
                                        }
                                }
                                p++;
                        }
                }

                write(handle2,yBuff_SQCIF,96*128);//Y written

                /* for Cb */
                p=0;
                for(i=0;i<144;i++)
                {
```

```
                        if(i%3==1)
                        {
                                q=0;
                                for(j=0;j<176;j++)
                                {
                                        if(j%11==0||j%11==3||j%11==6||j%11==9)
                                        {
                                                cbBuff_SQCIF[p][q]=cbBuff_CIF[i][j];
                                                q++;
                                        }
                                }
                                p++;
                        }
                }
                write(handle2,cbBuff_SQCIF,48*64);//Cb written

                /* for Cr */
                p=0;
                for(i=0;i<144;i++)
                {
                        q=0;
                        if (i%3==1)
                        {
                                for(j=0;j<176;j++)
                                {
                                        if(j%11==0||j%11==3||j%11==6||j%11==9)
                                        {
                                                crBuff_SQCIF[p][q]=crBuff_CIF[i][j];
                                                q++;

                                        }
                                }
                                p++;
                        }
                }
                write(handle2,crBuff_SQCIF,48*64);//Cr written
        }

        close(handle1);
        close(handle2);
}

void writeQCIF()
{
        handle1 = open(infile,O_RDONLY|O_BINARY);// enter the input file name here

        if(handle1==-1)
        {
                printf("error: *read* failed.\n");
                exit(1);
        }

        strcpy(outfile, infile);
        strcat(outfile, ".qcif");
        handle2  =  open(outfile,O_TRUNC|O_CREAT|O_WRONLY|O_BINARY);//  output  file  name;
default: test.sqcif

        if(handle2==-1)
        {
                printf("error: *write* failed.\n");
                exit(1);
        }

        int bytes=1,point=1;
        int i=0,j=0,p=0,q=0;

        printf("\nWriting to File... ");
```

```
while(1)
{
        point++;

        bytes=read(handle1,yBuff_CIF,288*352);        // Y Plane
        bytes=read(handle1,cbBuff_CIF,144*176); // U Plane
        bytes=read(handle1,crBuff_CIF,144*176); // V Plane

        if(point%5==0) printf(".");

        if(bytes==0)
        {
                printf("Done!\n\n");
                break;
        }
        p = 0;

        // i for height; j for width
        // p for height; q for width

        /* for Y plane */
        for(i=0;i<288;i++)
        {
                if(i%2==0)
                {
                        q=0;
                        for(j=0;j<352;j++)//j for width
                        {
                                if(j%2==0)
                                {
                                        yBuff_QCIF[p][q]=yBuff_CIF[i][j];
                                        q++;
                                }
                        }
                        p++;
                }
        }

        write(handle2,yBuff_QCIF,144*176);//Y written

        /* for Cb */
        p=0;
        for(i=0;i<144;i++)
        {

                if(i%2==0)
                {
                        q=0;
                        for(j=0;j<176;j++)
                        {
                                if(j%2==0)
                                {
                                        cbBuff_QCIF[p][q]=cbBuff_CIF[i][j];
                                        q++;
                                }
                        }
                        p++;
                }
        }
        write(handle2,cbBuff_QCIF,72*88);//Cb written

        /* for Cr */
        p=0;
        for(i=0;i<144;i++)
        {
                q=0;
                if (i%2==0)
```

```
                {
                        for(j=0;j<176;j++)
                        {
                                if(j%2==0)
                                {
                                        crBuff_QCIF[p][q]=crBuff_CIF[i][j];
                                        q++;
                                }
                        }
                        p++;
                }
        }
        write(handle2,crBuff_QCIF,72*88);//Cr written
}

close(handle1);
close(handle2);
```