Formaatonafhankelijke aanpassing en aflevering van multimediale data

Format-Independent Media Resource Adaptation and Delivery

Davy Van Deursen

Promotoren: prof. dr. ir. R. Van de Walle, dr. lic. W. De Neve
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Computerwetenschappen

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. J. Van Campenhout
Faculteit Ingenieurswetenschappen
Academiejaar 2008 - 2009

UNIVERSITEIT
GENT

ISBN 978-90-8578-287-2
NUR 980, 983
Wettelijk depot: D/2009/10.500/45

Dankwoord

“Dit stuk moet beter gemotiveerd worden.”, “Mooie locatie!”, “Vier troeven, altijd gewonnen.”, “Wij zijn perfect complementair.”, “Ben je bereid om hieraan mee te werken?”, “Ik ben blij dat je terug bent.”. Het zijn slechts enkele fragmenten die ik te horen kreeg in de voorbije drie en een half jaar, een periode die toeliet om mij zowel op wetenschappelijk als op persoonlijk vlak verder te kunnen ontwikkelen en die uiteindelijk geleid heeft tot het neerleggen van dit proefschrift. Bij deze wil ik dan ook een aantal mensen bedanken die een bijdrage hebben geleverd aan dit proefschrift.

In de eerste plaats wil ik mijn twee promotoren, prof. dr. ir. Rik Van de Walle en dr. lic. Wesley De Neve, bedanken. Ik bedank Rik voor de mogelijkheid die hij me gegeven heeft om in 2005 als onderzoeker van Multimedia Lab te starten en voor onze vele leerrijke gesprekken. Verder wil ik hem ook bedanken omdat hij mij de kans gegeven heeft om mee te werken aan verschillende nationale en internationale onderzoeksprojecten, om mijn onderzoek voor te stellen op internationale conferenties en om mee te werken aan Siruna. Wesley wil ik bedanken voor de inhoudelijke bijdrage die hij, als thesisbegeleider en als collega, tot dit proefschrift heeft geleverd. Hij heeft mij geïntroduceerd in de wetenschappelijke wereld en heeft mij de kunst van het publiceren geleerd.

Ik wens ook mijn vroegere en huidige collega’s van Multimedia Lab te bedanken voor het creëren van een aangename werksfeer. Zij zorgden ervoor dat ik steeds met plezier en veel motivatie kwam werken. Ik bedank in het bijzonder degene die mijn artikels of delen van dit proefschrift hebben nagelezen. Hun steeds opbouwende kritiek hebben de kwaliteit van dit werk in grote mate verhoogd. Ik bedank ook Rita Breems en Ellen Lammens voor hun administratieve ondersteuning.

Een bijzonder woord van dank gaat uit naar Wim Van Lancker, voor zijn bijdrage aan MuMiVA en NinSuna. Zonder hem was dit proefschrift één hoofdstuk korter geweest. Verder wil ik ook Davy De Schrijver bedanken voor de vele boeiende discussies over BSDL. Ik wil ook Erik Mannens bedanken voor zijn ondersteuning en begeleiding in het kader van onderzoeksprojecten

en om mij te introduceren in de wereld van W3C.

Ik bedank ook mijn familie en in het bijzonder mijn ouders voor de kansen die zij mij gaven, alsook voor hun wijze raad en steun in mindere tijden. Ook mijn vrienden bedank ik voor de ontspannende momenten die zij mij bezorgden.

Tot slot, en zeker niet het minst, wil ik Elke bedanken. Ondanks het feit dat ze het zelf niet altijd gemakkelijk had, heeft zij mij steeds gesteund en kon ik altijd op haar rekenen. Tijdens de vele leerrijke buitenlandse reizen die ik mocht maken in het kader van dit proefschrift (en waar zij mij altijd moest missen) is zij nooit uit mijn gedachten geweest. Ik kijk dan ook reikhalzend uit naar het vervolg van onze reis door het leven.

Davy Van Deursen
5 maart 2009

Summary

The multimedia landscape is characterized by a growing amount of multimedia content. This is due to ongoing digitization processes by public and private broadcasters, as well as to the growing popularity of user-generated content. There is also a large diversity in end-user devices that are able to consume multimedia. For instance, these devices differ in terms of device characteristics such as screen size, processing power, and battery life, and in terms of network characteristics, such as bandwidth, jitter, and error robustness. Furthermore, end-users with specific preferences often want to consume personalized versions of multimedia content (e.g., an end-user only requesting scenes satisfying his/her interests). Finally, the number of multimedia coding standards (e.g., MPEG-1 Audio, H.264/AVC, and JPEG2000) and multimedia delivery formats (e.g., MP4, Ogg, and RTP) has grown significantly over the last few years. Hence, the efficient delivery of multimedia content in today's world of ubiquitous multimedia consumption is an important technological challenge.

In order to obtain Universal Multimedia Access and thus providing (personalized) multimedia content anywhere, at anytime, and on any device, a transparent multimedia content adaptation and delivery approach is needed. In this context, metadata, which are generally defined as 'data about data', play a crucial role. Multimedia metadata enable the effective organization, access, and interpretation of multimedia content. Therefore, metadata have an increasingly important role in bringing order to the growing amount of available multimedia content. In this dissertation, we tackle the aforementioned problems by using format-independent content adaptation and delivery techniques. Furthermore, these techniques provide a seamless integration with today's manifold available multimedia metadata schemes.

In this dissertation, we first give an overview of existing format-independent content adaptation techniques. These techniques rely on XML-based Bitstream Syntax Descriptions (BSDs), which contain information about the high-level structure of a media bitstream. They typically apply a three-step-based adaptation chain to obtain format-independent adaptation, i.e., BSD gen-

eration, BSD transformation, and adapted bitstream generation. Rather than directly operating on the binary bitstream, the actual adaptation is performed on the BSD level, enabling the use of already existing XML transformation technologies such as XSLT or STX. Note that a BSD is not meant to replace the original binary data; it rather acts as an additional layer, similar to metadata. Further, we discuss how a BSD-based content adaptation chain can be implemented by means of generic, coding-format independent software modules. Examples of existing format-independent adaptation techniques following the three-step-based approach are MPEG-B BSDL, MPEG-21 gBS Schema, XFlavor, and BFlavor.

One important restriction of BSD-driven content adaptation techniques is that they can only perform high-level adaptation operations. Removing particular data blocks or modifying the value of certain syntax elements are considered as high-level adaptation operations. We identify two main target applications for BSD-driven content adaptation techniques, i.e., structural and semantic adaptations. Structural adaptations are typically performed to adapt media bitstreams by exploiting their scalability properties in order to meet the terminal and network characteristics of the end-user. Hence, structural adaptations are possible on condition that the media bitstreams consist of a number of scalability layers. Semantic adaptations are high-level adaptations based on semantic information about the multimedia content (e.g., selection of specific temporal segments that are of interest to the user). In this dissertation, we focus on semantic adaptations along the temporal axis. Hence, semantic adaptations are possible on condition that media bitstreams are provided with random access points that are occurring in a regular way.

Although format-independent content adaptation techniques seem to be very promising in the context of multimedia delivery in UMA environments, we have identified a number of remaining challenges:

- efficient and format-independent generation of generic Bitstream Syntax Descriptions (gBSDs);
- format-independent definition and implementation of high-level adaptation operations;
- integration with metadata standards;
- design and implementation of a fully integrated adaptation platform, based on format-independent content adaptation;
- format-independent packaging of adapted multimedia content;
- reduction of the structural metadata overhead.

The first challenge we tackle is specific for MPEG-21 gBS Schema, i.e., efficient and format-independent generation of generic Bitstream Syntax Descriptions (gBSDs). Within the MPEG-21 DIA specification, only the gBS Schema is described, together with the behaviour of a gBSDtoBin parser. This implies that a gBSD may be generated in any proprietary way. Therefore, we propose gBFlavor, which is an efficient tool for the generation of gBSDs in a format-independent manner. An existing format-independent solution is a two-step approach as currently proposed in the scientific literature, i.e., format-specific BSD generation, followed by a transformation in a gBSD. However, the latter solution requires knowledge of two different technologies (i.e., BSD generation and BSD-to-gBSD transformation). Furthermore, often more detail is needed in the BSD than in the resulting gBSD, implying a decrease in execution speed of the format-specific BSD generation process. These detailed BSDs are necessary for the BSD-to-gBSD transformation to produce an application-specific gBSD.

gBFlavor makes it possible to automatically generate a format-specific parser that is able to produce an application-specific gBSD for a given bitstream. Such a parser is generated by the gbflavorc translator, which takes as input a gBFlavor code. We propose the specification for this code, which describes the high-level structure of a particular coding format. The gBFlavor specification, which is an extension of the BFlavor specification, provides support for the addition of semantically meaningful information to gBSDs (by means of markers). Hence, gBSDs targeting specific applications can be obtained.

We compare gBFlavor with the existing two-step approach. This comparison allows getting an estimate of the expressive power and performance of a gBFlavor-enabled adaptation framework. The creation of gBSDs using gBFlavor avoids the two-step approach, since only one technology is needed to obtain gBSDs targeting a specific application. The exploitation of the scalable properties of two coding formats, i.e., SVC and JPEG2000, is used as the target application in our gBFlavor-enabled adaptation framework. Performance results show that gBFlavor outperforms the two-step approach in terms of execution speed.

In order to solve a number of problems with XML-driven content adaptation (i.e., lack of support for defining high-level adaptation operations in a format-independent way and an ad-hoc integration with content metadata), we present a new format-independent adaptation technique, called model-driven content adaptation. It relies on a model for media bitstreams that takes into account the structural metadata, content metadata, and scalability information. The model is implemented using the Web Ontology Language (OWL). Ex-

isting coding formats are mapped to the structural part of the model, while existing metadata standards can be linked to the content metadata model. Our new adaptation technique is based on executing SPARQL Protocol And RDF Query Language (SPARQL) queries over instances of the model for media bitstreams.

A comparison is made between model-driven content adaptation and existing format-independent and format-specific content adaptation techniques using different criteria. Compared to XML-driven content adaptation, advantages of model-driven content adaptation are the low amount of knowledge needed to define adaptation operations and the seamless integration with content metadata. Furthermore, both techniques have a comparable performance in terms of execution speed.

Next, we introduce two multimedia delivery platforms relying on format-independent software modules: MuMiVA and NinSuna. MuMiVA tackles the diversity in the current multimedia landscape by streaming multimedia content that is adapted according to the constraints of a certain usage environment. The multimedia content is customized using XML-driven content adaptation engines which, in their turn, use MPEG-B BSDL and MPEG-21 gBS Schema. We discuss MuMiVA's architecture and functioning, and elaborate on its extensibility features. The platform is evaluated using two different adaptation operations (i.e., exploitation of temporal scalability and shot selection) applied to two different coding formats (i.e., MPEG-4 Visual and H.264/AVC). We can conclude that the performance of MuMiVA in terms of CPU usage is not only dependent on the adaptation operation and coding format, but also on the level of detail of the (g)BSDs used.

NinSuna is a format-independent multimedia content adaptation and delivery platform that provides solutions for the shortcomings of MuMiVA, i.e., interoperability problems of XML-driven content adaptation and the lack of a generic multimedia delivery solution. NinSuna is based on our model for media bitstreams. It provides support for a seamless integration of adaptation operations and content metadata, and supports format-independent packaging of multimedia content. Multimedia adaptation is performed by selecting and adapting portions of the structural metadata using SPARQL. Multimedia packaging is obtained by encapsulating the selected and adapted structural metadata within a specific delivery format. This packaging process is implemented using XML transformation filters and MPEG-B BSDL. We evaluate the NinSuna platform by enabling the user to select news fragments matching his/her specific interests and usage environment characteristics. The multimedia content, encoded with H.264/AVC and AAC, can be delivered through RTP or MP4 according to the choice of the user. Both the generation of the structural metadata

and the adaptation and delivery of news fragments can occur in real time. The performance of the media adaptation and delivery steps in terms of memory usage and latency depends on the delivery format.

To conclude, we have shown that format-independent media resource adaptation and delivery is feasible in practice. Multimedia content can be adapted and delivered in real-time using a feasible amount of memory. The generation of the structural metadata, which is less time-critical than the actual adaptation and delivery, is also characterized by reasonable execution times and memory consumption. Furthermore, in order to prove the practicability of format-independent content adaptation and delivery techniques in real-life scenarios, two fully integrated format-independent multimedia content adaptation and delivery platforms are developed. We hope that this dissertation convinced the reader that within the wide range of multimedia adaptation techniques, there is room for format-independent content adaptation. Furthermore, we also hope that this dissertation provides the necessary knowledge needed to deploy format-independent adaptation techniques in real-world multimedia applications.

Samenvatting

Het multimedialandschap wordt gekenmerkt door een toenemende hoeveelheid van multimediale data. Dit komt door het digitaliseren van multimedia door openbare en private omroepen, alsook door op de groeiende populariteit van user-generated content. Er is ook een grote diversiteit in de apparaten die in staat zijn om multimedia af te spelen. Deze apparaten verschillen bijvoorbeeld in technische specificaties zoals schermgrootte, processorkracht en de levensduur van de batterij, maar ook in netwerkkarakteristieken zoals bandbreedte, jitter en foutrobuustheid. Bovendien hebben eindgebruikers vaak specifieke voorkeuren en vragen ze bijgevolg gepersonaliseerde versies op van de beschikbare multimediale data. Ten slotte is het aantal codeerformaten (bv. MPEG-1 Audio, H.264/AVC en JPEG2000) en het aantal afleveringsformaten (bv. MP4, Ogg en RTP) aanzienlijk toegenomen in de afgelopen jaren. Vandaar dat de efficiënte aflevering van multimediale data in de huidige wereld van alomtegenwoordige multimedia consumptie een belangrijke technologische uitdaging is.

Om universele multimedia toegang te bekomen, en dus (gepersonaliseerde) multimediale data overal, op elk moment en op elk apparaat voorzien, is er een transparante aanpak nodig voor de aanpassing en aflevering van multimedia. In deze context speelt metadata, welke in het algemeen gedefinieerd wordt als 'gegevens over gegevens', een cruciale rol. Metadata zorgt voor de effectieve organizatie, de toegankelijkheid en de interpretatie van multimediale data. Daarom krijgt metadata een steeds belangrijkere rol in het ordenen en organizeren van de groeiende hoeveelheid beschikbare multimediale data. In dit proefschrift pakken we de eerder genoemde problemen aan met behulp van formaatonafhankelijke technieken voor de aanpassing en aflevering van multimediale data. Bovendien zijn deze technieken vlot geïntegreerd met de veelvuldig voorkomende metadataschema's.

In deze dissertatie geven we eerst een overzicht van bestaande formaatonafhankelijke aanpassingstechnieken. Deze gebruiken XML-gebaseerde bitstroomsyntaxbeschrijvingen (Eng. Bitstream Syntax Descriptions; BSDs),

welke informatie bevatten over de hoogniveaustructuur van een binaire mediabron. Dergelijke technieken bestaan typisch uit drie stappen om formaatonafhankelijke aanpassing te bekomen: BSD-generatie, BSD-transformatie en de generatie van de aangepaste binaire mediabron. In plaats van rechtstreeks op de binaire mediastroom te werken, wordt de werkelijke aanpassing uitgevoerd op het niveau van de BSD zodat bestaande XML-transformatietechnologieën zoals XSLT of STX gebruikt kunnen worden. Merk op dat een BSD niet bedoeld is ter vervanging van de oorspronkelijke binaire mediabron, maar veeleer als een extra laag, vergelijkbaar met metadata. Verder bespreken we hoe een BSD-gebaseerde aanpassing van mediabronnen kan geïmplementeerd worden door middel van generieke, codeerformaatonafhankelijke software modules. Voorbeelden van bestaande formaatonafhankelijke aanpassingstechnieken die de driestappenaanpak gebruiken zijn MPEG-B BSDL, MPEG-21 gBS Schema, XFlavor en BFlavor.

Eén belangrijke beperking van de BSD-gebaseerde aanpassingstechnieken is dat ze enkel hoogniveau-aanpassingsoperaties kunnen uitvoeren. Het verwijderen van bepaalde datablokken of het wijzigen van de waarde van bepaalde syntaxelementen worden beschouwd als hoogniveau-aanpassingsoperaties. Wij hebben twee belangrijke toepassingen geïdentificeerd voor BSD-gebaseerde aanpassingstechnieken: structurele en semantische aanpassingen. Structurele aanpassingen worden doorgaans uitgevoerd om mediabronnen aan te passen zodat ze voldoen aan de beperkingen van een gebruikersomgeving. Structurele aanpassingen zijn echter maar mogelijk op voorwaarde dat de mediabronnen uit een aantal schaalbaarheidslagen bestaan. Semantische aanpassingen zijn hoogniveau-aanpassingen op basis van semantische informatie over de mediabron (bv. de selectie van specifieke temporele segmenten waar de eindgebruiker interesse voor heeft). In dit proefschrift hebben we ons gericht op semantische aanpassingen langs de temporele as. Vandaar dat semantische aanpassingen enkel mogelijk zijn op voorwaarde dat de mediabronnen voorzien zijn van regelmatig voorkomende toegangspunten.

Ondanks het feit dat formaatonafhankelijke aanpassingstechnieken veelbelovend lijken in de context van universele multimedia toegang hebben we toch een aantal resterende uitdagingen geïdentificeerd:

- efficiënte en formaatonafhankelijke generatie van generieke bitstroomsyntaxbeschrijvingen (Eng. generic Bitstream Syntax Descriptions; gBSDs);
- formaatonafhankelijke definitie en implementatie van hoogniveau-aanpassingsoperaties;
- integratie met metadatastandaarden;

- ontwerp en implementatie van een volledig geïntegreerd aanpassingsplatform, gebaseerd op formaatonafhankelijke aanpassingstechnieken;
- formaatonafhankelijke verpakking van aangepaste multimediale data;
- beperking van de structurele metadata kost.

De eerste uitdaging die we aanpakken is specifiek gericht op MPEG-21 gBS-schema: efficiënte en formaatonafhankelijke generatie van generieke bitstroomsyntaxbeschrijvingen. In de MPEG-21 DIA specificatie wordt alleen het gBS Schema beschreven, samen met het gedrag van een gBSDtoBin parser. Dit impliceert dat het generatieproces van een gBSD niet vastgelegd is. Daarom stellen we gBFlavor voor, een efficiënte tool voor de generatie van gBSDs op een formaatonafhankelijke manier. Een bestaande formaatonafhankelijke oplossing is een tweestappenaanpak, zoals voorgesteld in de wetenschappelijke literatuur: formaatspecifieke BSD-generatie, gevolgd door een transformatie in een gBSD. Echter, de laatste oplossing vereist kennis van twee verschillende technologieën (BSD-generatie en BSD-naar-gBSD-transformatie). Bovendien zijn vaak meer details nodig in de BSD dan in de resulterende gBSD, wat resulteert in een daling van de uitvoeringssnelheid van het formaatspecifieke BSD-generatieproces. Deze gedetailleerde BSD's zijn nodig voor de BSD-naar-gBSD-transformatie om een applicatiespecifieke gBSD te bekomen.

gBFlavor maakt het mogelijk om automatisch een formaatspecifieke parser te genereren die in staat is om een applicatiespecifieke gBSD te produceren voor een bepaalde mediabron. Deze parser wordt gegenereerd door de gbflavorc-vertaler, op basis van een gBFlavor code. Deze code beschrijft de hoogniveaustructuur van een bepaald codeerformaat. In dit proefschrift stellen we de specificatie op voor deze code. De gBFlavor-specificatie is een uitbreiding van de BFlavor-specificatie en biedt ondersteuning voor het toevoegen van semantische informatie aan gBSDs (door middel van markers). Op die manier kunnen gBSDs bekomen worden die gericht zijn op specifieke toepassingen.

We vergelijken gBFlavor met de bestaande tweestappenaanpak. Deze vergelijking staat ons toe om een schatting te maken van de expressieve kracht en performantie van een aanpassingsraamwerk gebaseerd op gBFlavor. De creatie van gBSDs gebruikmakend van gBFlavor vermijdt de tweestappenaanpak omdat er maar één enkele technologie nodig is om applicatiespecifieke gBSDs te creëren. Het uitbuiten van de schaalbare eigenschappen van twee codeerformaten (SVC en JPEG2000) wordt gebruikt als toepassing voor het testen van gBFlavor. Resultaten tonen aan dat gBFlavor beter presteert dan de tweestappenaanpak in termen van uitvoeringssnelheid.

Om een aantal problemen met XML-gedreven aanpassing van multimediale data (meer bepaald het gebrek aan ondersteuning voor de definitie van hoogniveau-aanpassingsoperaties op een formaatonafhankelijke manier en een ad-hoc integratie met inhoudelijke metadata), stellen we een nieuwe formaatonafhankelijke aanpassingstechniek voor: model-gedreven aanpassing. Deze techniek is gebaseerd op een model voor mediabronnen, waarin structurele metadata, inhoudelijke metadata en schaalbaarheidsinformatie beschreven worden. Het model is geïmplementeerd met behulp van de Web Ontology Language (OWL). Bestaande codeerformaten kunnen gelinkt worden aan het structurele gedeelte van het model, terwijl de bestaande metadataformaten kunnen gekoppeld worden aan het inhoudelijke metadata model. Onze nieuwe aanpassingstechniek is gebaseerd op de uitvoering van SPARQL protocol en RDF Query Language (SPARQL) zoekopdrachten, toegepast op instanties van het model voor mediabronnen.

We maken een vergelijking tussen model-gedreven aanpassing en bestaande formaatonafhankelijke en formaatspecifieke aanpassingstechnieken op basis van verschillende criteria. De voordelen van model-gedreven aanpassing ten opzichte van XML-gedreven aanpassing zijn de weinige kennis die nodig is voor de definitie van aanpassingsoperaties en de vlotte integratie met inhoudelijke metadata. Bovendien hebben beide technieken vergelijkbare prestaties in termen van uitvoeringssnelheid.

Vervolgens introduceren we twee platforms voor de aflevering van multimediale data op basis van formaatonafhankelijke software modules: MuMiVA en NinSuna. MuMiVA pakt de diversiteit in het huidige multimedialandschap aan door multimediale data te stromen die aangepast is aan de beperkingen van een gebruikersomgeving. De multimediale data wordt aangepast met behulp van XML-gedreven aanpassingsmodules die op hun beurt gebruikmaken van MPEG-B BSDL en MPEG-21 gBS Schema. We bespreken de architectuur en de werking van MuMiVA, alsook de uitbreidingsmogelijkheden. Het platform wordt geëvalueerd aan de hand van twee verschillende aanpassingsoperaties (het uitbuiten van temporele schaalbaarheid en shotselectie) toegepast op twee verschillende codeerformaten (MPEG-4 Visual and H.264/AVC). We kunnen tot de conclusie komen dat de prestaties van MuMiVA in termen van CPU-gebruik niet alleen afhankelijk zijn van de aanpassingsoperatie en het codeerformaat, maar ook van de mate van detail van de gebruikte (g)BSD's.

NinSuna is een platform voor de formaatonafhankelijke aanpassing en aflevering van multimediale data. Het platform biedt oplossingen voor de tekortkomingen van MuMiVA: interoperabiliteitsproblemen van XML-gedreven aanpassing en het ontbreken van een generieke oplossing voor het afleveren van multimediale data. NinSuna is gebaseerd op ons model voor mediabron-

nen en biedt ondersteuning voor een vlotte integratie tussen aanpassingsoperaties en inhoudelijke metadata. Verder ondersteunt NinSuna ook formaatonafhankelijke verpakking van multimediale data. Aanpassing van multimediale data wordt bekomen door het selecteren en aanpassen van delen van de structurele metadata met behulp van SPARQL. Aflevering van multimediale data wordt bekomen door het verpakken van de geselecteerde en aangepaste structurele metadata in een specifiek afleveringsformaat. Deze verpakking wordt uitgevoerd met behulp van XML-transformatiefilters en MPEG-B BSDL. We evalueren het NinSuna platform door de eindgebruiker nieuwsfragmenten te laten selecteren die voldoen aan zijn/haar interesses en gebruikersomgeving. De multimediale data, gecodeerd met H.264/AVC en AAC, kan worden afgeleverd via RTP of MP4 volgens de keuze van de gebruiker. Zowel de generatie van de structurele metadata en de aanpassing en aflevering van nieuwsfragmenten kan gebeuren in ware tijd. De prestaties van de aanpassings- en afleveringsstappen in termen van geheugengebruik en latentie hangen af van het afleveringsformaat.

We hopen met dit proefschrift te hebben aangetoond dat formaatonafhankelijke aanpassing en aflevering van multimediale data haalbaar is in de praktijk. Multimediale data kan worden aangepast en afgeleverd in ware tijd, met een aanvaardbaar geheugengebruik. De generatie van structurele metadata, die minder tijdskritisch is dan de werkelijke aanpassing en aflevering, wordt ook gekenmerkt door redelijke uitvoeringstijden en geheugenverbruik. Om de uitvoerbaarheid te bewijzen van formaatonafhankelijke aanpassing en aflevering van multimediale data in werkelijke scenario's werden twee volledig geïntegreerde formaatonafhankelijke platforms ontwikkeld voor de aanpassing en aflevering van multimediale data. Wij hopen dan ook met dit proefschrift de lezer ervan overtuigd te hebben dat, binnen het brede scala van aanpassingstechnieken voor multimediale data, er ruimte is voor formaatonafhankelijke technieken. Verder hopen we ook dat dit proefschrift de nodige kennis aanlevert die nodig is voor het implementeren van formaatonafhankelijke aanpassingstechnieken in hedendaagse multimediatoepassingen.

List of abbreviations

AAC	Advanced Audio Coding
AAC SSR	AAC Scalable Sample Rate
ADTE	Adaptation-Decision Taking Engine
APE	Adaptation and Packaging Engine
AVC	Advanced Video Coding
BBL	Bitstream Binding Language
BiM	Binary MPEG format for XML
BFlavor	BSDL + XFlavor
BS Schema	Bitstream Syntax Schema
BSD	Bitstream Syntax Description
BSDL	Bitstream Syntax Description Language
CMML	Continuous Media Markup Language
CMS	Content Management System
COMM	Core Ontology for MultiMedia
CPU	Central Processing Unit
DANAE	Dynamic and distributed Adaptation of scalable multimedia coNtent in a context-Aware Environment
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
RDF DAWG	RDF Data Access Working Group
DIA	Digital Item Adaptation
DIS	Digital Item Streaming
DOM	Document Object Model
DSS	Darwin Streaming Server
EBNF	Extended Backus-Naur Form
FGS	Fine-Grained quality Scalability
FMO	Flexible Marcroblock Ordening
FOAF	Friend Of A Friend
FLAVOR	Formal Language for Audio-Visual Object Representation
gBFlavor	gBS Schema + BFlavor
gBS Schema	generic Bitstream Syntax Schema
gBSD	generic Bitstream Syntax Description
GOP	Group Of Pictures
GRDDL	Gleaning Resource Descriptions from Dialects of Languages
HD	High Definition

HTTP	HyperText Transport Protocol
IDR	Instantaneous Decoding Refresh
IIS	Internet Information Server
I/O	Input/Output
ISIS	Intelligent Scalability for Interoperable Services
Java EE	Java Enterprise Edition
JPEG	Joint Photographic Experts Group
LALR	Look-Ahead Left to right, Rightmost derivation
MFWG	Media Fragments Working Group
MMSem	Multimedia Semantics Incubator Group
MP3	MPEG-1 Audio Layer 3
MPEG	Moving Picture Experts Group
MPML	MPEG Video Markup Language
MuMiVA	Mutare, Mittere, Videre, Audire
MXF	Material eXchange Format
N3	Notation 3
NAL	Network Abstraction Layer
NALU	NAL Unit
NinSuna	NinSuna INtelligent Search framework for UNiversal multimedia Access
OWL	Web Ontology Language
P2P	Peer-to-Peer
PC	Personal Computer
PDA	Personal Digital Assistant
POC	Picture Order Count
PPS	Picture Parameter Set
RAP	Random Access Point
RAU	Random Access Unit
RDBMS	Relational Data Base Management System
RDF	Resource Description Framework
RIF	Rule Interchange Format
ROI	Region Of Interest
RTP	Real-time Transport Protocol
RTSP	Real-Time Streaming Protocol
SAX	Simple API for XML
SDL	Syntax Description Language
SDP	Session Description Protocol
SEI	Supplemental Enhancement Information
SMG	Structural Metadata Generator
SNR	Signal-to-Noise Ratio
SPARQL	SPARQL Protocol And RDF Query Language
SPARUL	SPARQL Protocol And RDF Update Language
SPS	Sequence Parameter Set
STX	Streaming Transformations for XML
SVC	Scalable Video Coding

UMA	Universal Multimedia Access
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VC-1	Video Codec 1
VLC	Variable Length Code
VLC	VideoLan Client
VOS	Video Object Sequence
W3C	World Wide Web Consortium
WMA	Windows Media Audio
DSL	Digital Subscriber Lines
XFlavor	Flavor, extended with XML features
XML	Extensible Markup Language
XML-WRT	Word Replacing Transform for XML
XPath	XML Path Language
XSLT	Extensible Stylesheet Language Transformations
XWRT	XML-WRT
Yacc	Yet another compiler compiler

Contents

Chapter 1

Introduction

It is not the strongest of the species that survives, nor the most intelligent, but rather the one most responsive to change.

Charles Darwin (1809 - 1882)

1.1 Context

The growth in multimedia content including collections of photos, music, and videos has been significant in recent years. There are two major reasons for this growth. First, many public and private broadcasters are in the process of digitizing their complete radio and television broadcasting production process. Thanks to this digitization process, media assets can be preserved for the future. Furthermore, the media assets become more easily available, with many manual or labor-intensive steps in a production process eliminated. Second, end-users nowadays not only consume multimedia content, they have also become content producers [90]. With the introduction of digital photo and video cameras, end-users can easily produce multimedia content. Web sites such as YouTube[1] and Flickr[2] offer the possibility to upload, annotate, manage, and share this user-generated content.

Content providers want to make sure that their multimedia content can be accessed by a broad range of devices, ranging from mobile phones over Personal Computers (PCs) to High Definition (HD) televisions. The multimedia content is delivered using various network technologies such as Universal Mobile Telecommunications System (UMTS, [140]) or broadband Digital Sub-

[1] `http://www.youtube.com/`
[2] `http://www.flickr.com/`

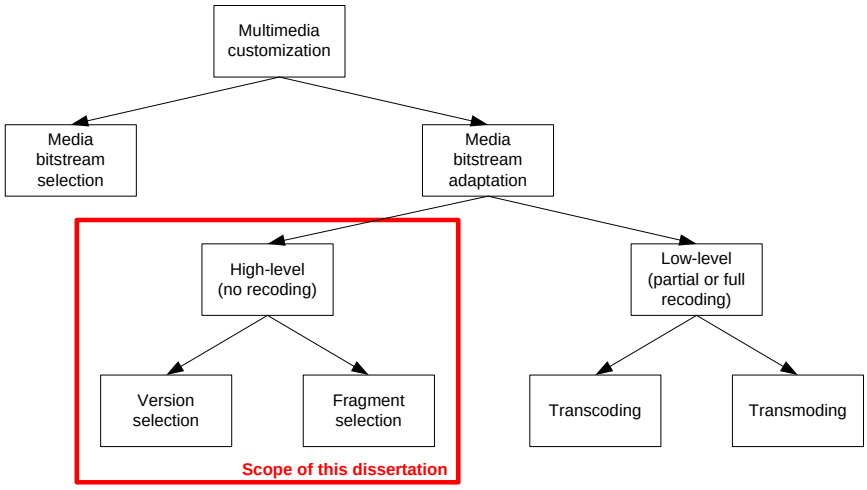


Figure 1.1: Multimedia content customization approaches.

scriber Lines (xDSL, [52]). The tremendous diversity in end-user devices and network technologies introduces difficulties for multimedia delivery systems. For instance, end-user devices differ in terms of screen size, processing power, and battery life. Network technologies may differ in terms of bandwidth, jitter, and error robustness. Further, end-users with specific preferences often want to obtain a personalized version of multimedia content (e.g., an end-user only requesting scenes satisfying his/her interests). Hence, the efficient delivery of multimedia content in today's world of ubiquitous multimedia consumption is an important technological challenge. It is clear that a transparent approach is needed in order to provide multimedia content anywhere, at anytime, and on any device. This vision is generally known as Universal Multimedia Access (UMA, [136]).

1.1.1 Multimedia Content Customization

Multimedia content customization is an emerging field due to the above described UMA paradigm. A multimedia content customization system tries to meet the user needs by customizing the content based on the properties of the usage environment and user preferences. There exists a wide variety of multimedia customization approaches in the compressed domain, as described by Magalhães *et al.* in [77]. These approaches are also visualized in Figure 1.1. Two major categories can be distinguished: media bitstream selection and adaptation.

Media Bitstream Selection

One possibility to meet the usage environment constraints is to choose between several media bitstreams representing the same content. Media bitstream selection, also known as simulstore or simulcast, corresponds to the identification of the most adequate media bitstream from those available to be sent to the end-user. The selected bitstream may already be adequate enough or may need further adaptation using techniques discussed below. Content versions may include different media types (e.g., video or a thumbnail image), coding formats (e.g., H.264/AVC or MPEG-2 Video), or bitstream characteristics (e.g., resolution or bit rate). Examples of media bitstream selection technologies are RealNetworks' SureStream [104], Multiple Bit Rate (MBR) profile used in the Windows Media Series [144], Microsoft's Smooth Streaming [86], and the alternative track selection mechanism used in Quicktime [100]. Further, the Synchronized Multimedia Integration Language (SMIL, [16]) provides support for content selection based on the usage environment (e.g., language, device characteristics, or available bandwidth).

Media Bitstream Adaptation

Media bitstream adaptation [18] can be divided into two categories:

- *Low-level* adaptation operations completely or partially recode the compressed media bitstream to customize it according to the constraints of the usage environment. Transcoding is one low-level adaptation technique, which typically uses signal-processing operations [1, 137, 146]. Examples of such operations are bit rate reduction, spatial scaling, and coding format conversions. Transmoding is another low-level adaptation technique and is used when the usage environment conditions do not allow to consume the compressed media bitstream with its original media type. In this case, a modality transformation can be used from one media type to another. Examples are the conversion of text to audio, video to key frame images, and large images to video [9].

- *High-level* adaptation operations typically perform the removal of high-level bitstream structures or the modification of high-level syntax elements. Hence, compressed media bitstreams can be customized without the need of a complete or partial recode process. Two categories of high-level adaptation operations exist: version and fragment selection. Applying version selection results in a tailored version of the media bitstream based on the constraints of the terminal and network constraints of the end-user. Hence, the resulting media bitstream will contain the

same content, but the visual quality of this content will be different (e.g., lower resolution or lower frame rate). On the other hand, fragment selection maintains the visual quality of the content of media bitstreams, but the resulting tailored media bitstream will contain only a subset of the original content (e.g., a specific scene or a specific region). Hence, with fragment selection, the media bitstream is customized based on the semantics of the multimedia content and the users' preferences. Note that both high-level adaptation approaches can also be combined. For example, scene selection can be applied together with frame rate reduction.

Scalable coding is an interesting coding technique in the context of version selection. A scalable media resource consists of different layers providing different quality. Hence, multiple (lower quality) versions of the same media resource can be extracted by performing simple editing operations [91]. The bitstream extraction process typically involves the removal of particular data blocks and the modification of the values of certain syntax elements. In contrast to transcoding, scalable coding intrinsically assumes that the content shall be distributed through heterogeneous usage environments. This implies that the coding format already provides several layers, decoupling coding and adaptation processes. Examples of scalable coding formats are Scalable Video Coding (SVC, [107]) and JPEG2000 [19]. Both standards offer several possibilities to exploit scalability such as spatial, Signal-to-Noise Ratio (SNR), and Region Of Interest (ROI) scalability.

The number of multimedia coding standards has grown significantly over the last few years, especially with the introduction of new coding formats such as H.264/AVC [68], SVC, AAC [62], and JPEG XR [115]. At the same time, older standards such as H.262/MPEG-2 Video [58], MPEG-1 Audio [56], and JPEG2000 are still present. Next to coding formats, there also exists a wide variety of delivery formats, i.e., formats encapsulating encoded media bitstreams (e.g., the MP4 file format [59] or the Real-time Transport Protocol (RTP, [105])). In order to deal with current and future multimedia coding and delivery formats, format-independent adaptation and delivery systems are gaining importance. A format-independent adaptation technique was first introduced by Amielh *et al.* in [3]. It is based on automatically generated XML descriptions of the high-level structure of media bitstreams, called Bitstream Syntax Descriptions (BSDs) or *structural metadata*.

As depicted in Figure 1.1, the scope of this dissertation is situated within the category of high-level adaptation operations. Moreover, we use format-independent adaptation techniques to implement these high-level adaptation operations.

1.1.2 Multimedia Metadata

Multimedia metadata enables the effective organization, access, and interpretation of multimedia content. Searching, indexing, linking, sharing, and presentation of multimedia content are example applications of multimedia metadata. Another important application of multimedia metadata is adaptation of multimedia content. Indeed, adaptation engines can use multimedia metadata as input to produce adapted multimedia content. In this dissertation, we use multimedia metadata mainly for adaptation purposes. Due to the growth of available multimedia content in recent years, metadata has an increasingly important role in bringing order to the emerging chaos [110]. Metadata, which is generally defined as 'data about data', can come in many forms:

- metadata represented as binary or textual data;
- structured or loose metadata;
- metadata which are tightly or loosely coupled with the corresponding media resource.

The World Wide Web Consortium (W3C[3]) has developed a number of recommendations for the representation of metadata. Multiple standards for the description of multimedia content developed in recent years are based on the Extensible Markup Language (XML, [14]). XML is a simple, very flexible text format which plays already an important role in the exchange of a wide variety of data on the Web and elsewhere. It is designed for markup in documents of arbitrary structure. A well-formed XML document creates a balanced tree of nested sets of open and close tags, each of which can include several attribute-value pairs. There is no fixed tag vocabulary or set of allowable combinations, so these can be defined for each application. XML is used to serve a range of purposes such as the serialization syntax for other markup languages, semantic markup of Web pages, and uniform data exchange [41]. Further, the XML Schema language [13, 118] provides a means for defining the structure and content of XML, i.e., defining a grammar for XML documents. W3C has also developed an XML transformation language, i.e., Extensible Stylesheet Language Transformations (XSLT, [21]), which is a language for transforming XML documents into other XML documents. Examples of XML-based metadata standards for the representation of multimedia content are MPEG-7 [80], Dublin Core [43], and MPEG-21 [17].

With the introduction of the Semantic Web, enhanced interoperability among different metadata standards is obtained thanks to a more natural repre-

[3] `http://www.w3.org/`

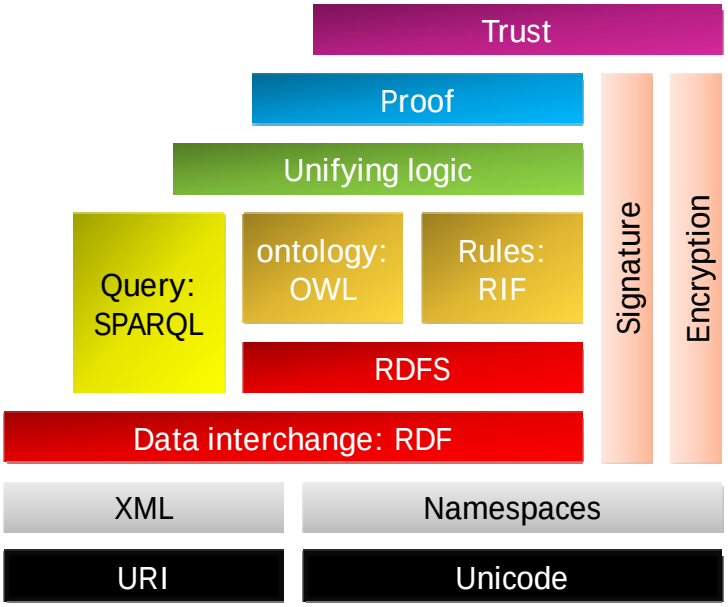


Figure 1.2: The Semantic Web stack as introduced by Tim Berners-Lee[a].

[a] http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

sentation of objects and relationships. The Semantic Web is an extension of the World Wide Web that provides a common framework allowing people to share content beyond the boundaries of applications and Web sites. It derives from W3C director Sir Tim Berners-Lee's vision of the Web as a universal medium for data, information, and knowledge exchange [12]. The Semantic Web is about two things. It is about common formats for integration and combination of data drawn from diverse sources, while the original Web is mainly concentrated on the interchange of (XML-based) documents. It is also about a language for recording how the data relates to real world objects. This language provides formal and explicit specifications of domain models, which define the terms used and their relationships. Such formal domain models are called *ontologies*. Ontologies define data models in terms of classes, subclasses, and properties.

At its core, the Semantic Web comprises a set of design principles and a variety of enabling technologies. Some elements of the Semantic Web are expressed as prospective future possibilities that are yet to be implemented or realized. Other elements of the Semantic Web are expressed in formal specifications. An overview of existing Semantic Web technologies is given below.

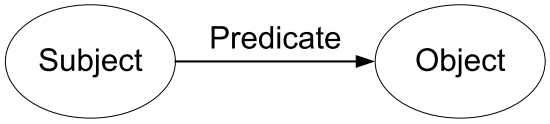


Figure 1.3: An RDF triple represented by a directed-arc diagram.

Note that these technologies can also be visualized as different layers of the Semantic Web stack, as shown in Figure 1.2.

- One of the building blocks of the Semantic Web is the *Resource Description Framework* (RDF, [72]). RDF introduces a graph data model. The underlying structure of any expression in RDF is a collection of triples, each consisting of a subject, a predicate, and an object. A set of such triples is called an RDF graph. As shown in Figure 1.3, this can be illustrated by a node and directed-arc diagram, in which each triple is represented as a node-arc-node link. A node may be a Uniform Resource Identifier (URI) with optional fragment identifier, a literal, or a blank node. A blank node is a node that is not a URI reference or a literal; it is just a unique node that can be used in one or more RDF statements, but which has no intrinsic name. An arc or property is represented by a URI reference. An example of an RDF triple is `<http://foo.com#pat> <http://foo.com#age> 24.` (in Notation 3 or N3 [119]), which states that Pat's age is 24.

- RDF's vocabulary description language, *RDF Schema* [15], is a semantic extension of RDF. It provides mechanisms for describing groups of related resources and the relationships between these resources. RDF Schema provides the framework to describe application-specific classes and properties. Classes in RDF Schema are much like classes in object oriented programming languages. This allows resources to be defined as instances of classes, and subclasses of classes.

- The *Web Ontology Language* (OWL, [83]) is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is developed as a vocabulary extension of RDF and is derived from the DAML+OIL Web ontology language [70]. OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema by providing additional vocabulary along with formal

semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

- The *SPARQL Protocol And RDF Query Language* (SPARQL, [99]) is used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries. The results of SPARQL queries can be results sets or RDF graphs. Furthermore, the SPARQL protocol describes a means for conveying SPARQL queries to a SPARQL query processing service and returning the query results to the entity that requested them.

- The *Gleaning Resource Descriptions from Dialects of Languages* (GRDDL, [24]) specification introduces markup based on existing standards for declaring that an XML document includes RDF-compatible data and for linking to algorithms for extracting this data from the document. Such algorithms are usually implemented by using Extensible Stylesheet Language Transformations (XSLT, [71]).

- The *Rule Interchange Format (RIF)* is currently being developed by the RIF Working Group[4]. The goal of this Working Group is to specify a format for rules, so that they can be used across diverse systems. This format (or language) will function as an interlingua into which established and new rule languages can be mapped, allowing rules written for one application to be published, shared, and re-used in other applications and other rule engines.

XML and RDF each have their merits as a foundation for the Semantic Web, but RDF provides more suitable mechanisms for applying ontology representation languages like OWL to the task of interoperability. XML's major limitation is that it just describes grammars. There is no way to recognize a semantic unit from a particular domain because XML aims at document structure and imposes no common interpretation of the data contained in the document [41]. Note that XML is useful for data interchange between applications that both know what the data is. However, for situations where new communication partners using different domain models are frequently added (e.g., on the Web), using XML requires much more effort than necessary. Indeed,

[4]The Working Group Charter is available on `http://www.w3.org/2005/rules/wg/charter.html`

original domain models must be reengineerd from the XML grammars and the mappings between the concepts and relationships must be defined. Subsequently, these domain mappings must be translated using mapping procedures such as XSLT.

When it comes to semantic interoperability, RDF has significant advantages over XML: the triple structure provides natural semantic units because all objects are independent entities. Furthermore, with RDF Schema and OWL, which are built on top of RDF, domain models can be represented naturally, so translation steps are not necessary (in contrast to XML). Also, techniques from research in knowledge representation are directly applicable for defining mappings between two RDF descriptions. Examples of OWL-based multimedia metadata specifications are the Core Ontology for Multimedia (COMM, [6]) and the DIG35 vocabulary[5]. XML, XML Schema, XSLT, RDF, RDF Schema, OWL, and SPARQL are key technologies and are extensively used in this dissertation to define new algorithms for format-independent multimedia adaptation and delivery.

1.2 Goals and Outline

As described in Section 1.1, format-independent content adaptation and delivery is an interesting approach since it tackles the UMA paradigm, while also taking into account the huge diversity in coding and delivery formats. The main goal of this dissertation is to investigate the feasibility of format-independent content adaptation and delivery in a multimedia landscape that is characterized by heterogeneity in terms of end-user devices, network technologies, user preferences, coding formats, and delivery formats. Therefore, we address the following topics in this dissertation:

- *Efficient generation of structural metadata*: format-independent content adaptation systems rely on structural metadata (i.e., BSDs), as mentioned in Section 1.1. Therefore, given compressed media bitstreams, it is important to be able to efficiently generate their structural metadata.

- *Adaptation and metadata*: with the increasingly important role of multimedia metadata, it is important to know how adaptation systems (and more specifically format-independent adaptation systems) can interact with multimedia metadata. Therefore, we investigate how format-independent content adaptation and delivery systems can bridge the gap

[5] Available on `http://http://www.w3.org/2005/Incubator/mmsem/XGR-vocabularies/#formal`

between multimedia metadata and adaptation of compressed media bitstreams.

- *Fully integrated adaptation and delivery platforms*: defining algorithms for the adaptation of media bitstreams is one thing, but it is also important to address the design and implementation of a fully integrated multimedia adaptation and delivery platform, which is based on these adaptation algorithms. More specifically, we want to create a fully integrated adaptation and delivery platform that relies on format-independent content adaptation engines.

The outline of this dissertation is as follows. Chapter 2 gives the reader an overview of format-independent content adaptation techniques. The general principles of a format-independent content adaptation system are discussed, together with the main target applications and existing technologies. An overview of the existing challenges of format-independent content adaptation techniques is given as well. Chapter 3 introduces gBFlavor, a novel tool to optimize format-independent content adaptation systems based on generic Bitstream Syntax Schema (gBS Schema). An overview of the gBFlavor specification is provided, as well as an outline of the general functioning of a gBFlavor-enabled adaptation framework. A new format-independent adaptation technique, which is called model-driven content adaptation, is presented in Chapter 4. It relies on a model for media bitstreams that takes into account the structural metadata, content metadata, and scalability information. Further, we elaborate on the adaptation chain of model-driven content adaptation and evaluate our new adaptation technique against other adaptation techniques. In Chapter 5, we address the design and functioning of two fully integrated platforms for multimedia adaptation and delivery. The first platform, which is called MuMiVA, relies on two standardized, XML-driven content adaptation tools (i.e., MPEG-B BSDL and MPEG-21 gBS Schema) and is able to deliver the multimedia content using the RTP/RTSP protocol. Our second platform, which is called NinSuna, solves a number of problems of the MuMiVA platform and relies on model-driven content adaptation, our new adaptation technique introduced in Chapter 4. Furthermore, NinSuna also provides support for the packaging of adapted multimedia content in a format-independent way. Finally, conclusions are drawn in Chapter 6.

1.3 Overview of Publications

The research activities that led to this dissertation resulted in a number of A1 publications (7 accepted, 2 submitted): three papers are published in *Signal*

Processing: Image Communication, one paper is published in *Multimedia Tools and Applications*, and three papers are published in *Lecture Notes in Computer Science*. Further, one paper is submitted to *Multimedia Systems* and one paper is submitted to *Multimedia Tools and Applications*. Next to this, the work described in this dissertation contributed to 20 papers that were presented at international conferences.

1.3.1 A1 Publications

1. W. De Neve, D. Van Deursen, D. De Schrijver, K. De Wolf, and R. Van de Walle. Using Bitstream Structure Descriptions for the Exploitation of Multi-layered Temporal Scalability in H.264/AVC's Base Specification. *Lecture Notes in Computer Science – Advances in Mulitmedia Information Processing - PCM 2005*, volume 3768, pages 641–652, November 2005

2. W. De Neve, D. Van Deursen, D. De Schrijver, S. Lerouge, K. De Wolf, and R. Van de Walle. BFlavor: a Harmonized Approach to Media Resource Adaptation Inspired by MPEG-21 BSDL and XFlavor. *Signal Processing: Image Communication*, 21(10):862–889, November 2006

3. D. Van Deursen, F. De Keukelaere, L. Nachtegaele, J. Feyaerts, and R. Van de Walle. A Scalable Presentation Format for Multichannel Publishing Based on MPEG-21 Digital Items. *Lecture Notes in Computer Science – Multimedia Content Representation, Classification and Security*, volume 4105, pages 650–657, September 2006

4. P. Lambert, D. De Schrijver, D. Van Deursen, W. De Neve, Y. Dhondt, and R. Van de Walle. A Real-Time Content Adaptation Framework for Exploiting ROI Scalability in H.264/AVC. *Lecture Notes in Computer Science – Advanced Concepts for Intelligent Vision Systems*, volume 4179, pages 442–453, September 2006

5. S. De Bruyne, D. Van Deursen, J. De Cock, W. De Neve, P. Lambert, and R. Van de Walle. A Compressed-domain Approach for Shot Boundary Detection on H.264/AVC Bit Streams. *Signal Processing: Image Communication – Special Issue on Semantic Analysis for Interactive Multimedia Services*, 23(7):473–498, August 2008

6. D. Van Deursen, W. De Neve, D. De Schrijver, and R. Van de Walle. gBFlavor: a New Tool for Fast and Automatic Generation of generic Bitstream Syntax Descriptions. *Multimedia Tools and Applications*, 40(3):453–494, December 2008

7. W. De Neve, D. Van Deursen, W. Van Lancker, Y. M. Ro, and R. Van de Walle. Improved BSDL-based Content Adaptation for JPEG 2000 and HD Photo (JPEG XR). *Signal Processing: Image Communication – Special Issue on Scalable Coded Media beyond Compression*, 24(6):452–467, July 2009

8. D. Van Deursen, W. Van Lancker, S. De Bruyne, W. De Neve, E. Mannens, and R. Van de Walle. Format-independent and Metadata-driven Media Resource Adaptation using Semantic Web Technologies. Submitted to *Multimedia Systems Journal*

9. D. Van Deursen, W. Van Lancker, T. Paridaens, W. De Neve, E. Mannens, and R. Van de Walle. NinSuna: a Fully Integrated Platform for Format-independent Multimedia Content Adaptation and Delivery based on Semantic Web Technologies. Submitted to *Multimedia Tools and applications – Special Issue on Data Semantics for Multimedia Systems*

1.3.2 Other Publications

1. W. De Neve, D. De Schrijver, D. Van Deursen, and R. Van de Walle. XML-Driven Bitstream Extraction Along the Temporal Axis of SMPTE's Video Codec 1. In *Proceedings of the 7th International Workshop on Image Analysis for Multimedia Interactive Services*, pages 233–236, April 2006, Seoul, South Korea

2. D. Van Deursen, W. De Neve, D. De Schrijver, and R. Van de Walle. BFlavor: an Optimized XML-based Framework for Multimedia Content Customization. In *Proceedings of the 25th Picture Coding Symposium*, 6 pages on CD-ROM, April 2006, Beijing, China

3. D. De Schrijver, W. De Neve, D. Van Deursen, J. De Cock, and R. Van de Walle. On an Evaluation of Transformation Languages in a Fully XML-driven Framework for Video Content Adaptation. In *Proceedings of the first International Conference on Innovative Computing, Information and Cntrol (ICICIC06)*, volume 3, pages 213–216, September 2006, Beijing, China

4. F. De Keukelaere, D. Van Deursen, and R. Van de Walle. Multichannel Distribution for Universal Multimedia Access in Home Media Gateways. *Lecture Notes in Computer Science – Entertainment Computing - ICEC 2006*, volume 4161, pages 147–152, September 2006

5. W. De Neve, D. De Schrijver, D. Van Deursen, P. Lambert, and R. Van de Walle. Real-Time BSD-Driven Adaptation Along the Temporal Axis of H.264/AVC Bitstreams. *Lecture Notes in Computer Science – Advances in Multimedia Information Processing - PCM 2006*, volume 4261, pages 131–140, November 2006

6. D. Van Deursen, D. De Schrijver, W. De Neve, and R. Van de Walle. A Real-Time XML-Based Adaptation System for Scalable Video Formats. *Lecture Notes in Computer Science – Advances in Multimedia Information Processing - PCM 2006*, volume 4261, pages 339–348, November 2006

7. D. De Schrijver, W. De Neve, D. Van Deursen, S. De Bruyne, and R. Van de Walle. Exploitation of Interactive Region of Interest Scalability in Scalable Video Coding by Using an XML-driven Adaptation Framework. In *Proceedings of the 2nd International Conference on Automated Production of Cross Media Content for Multi-channel Distribution*, pages 223–231, December 2006, Leeds, United Kingdom

8. S. De Bruyne, D. De Schrijver, W. De Neve, D. Van Deursen, and R. Van de Walle. Enhanced Shot-Based Video Adaptation using MPEG-21 generic Bitstream Syntax Schema. In *Proceedings of the 2007 IEEE Symposium Series on Computational Intelligence*, 6 pages on CD-ROM, April 2007, Honolulu, Hawaii

9. D. De Schrijver, W. De Neve, K. De Wolf, P. Lambert, D. Van Deursen, and R. Van de Walle. XML-driven Exploitation of Combined Scalability in Scalable H.264/AVC Bitstreams. In *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems*, pages 1521–1524, May 2007, New Orleans, United States

10. D. De Schrijver, W. De Neve, D. Van Deursen, Y. Dhondt, and R. Van de Walle. XML-based Exploitation of Region of Interest Scalability in Scalable Video Coding. In *Proceedings of the 8th International Workshop on Image Analysis for Multimedia Interactive Services*, 4 pages on CD-ROM, June 2007, Santorini, Greece

11. D. Van Deursen, D. De Schrijver, S. De Bruyne, and R. Van de Walle. Fully Format Agnostic Media Resource Adaptation Using an Abstract Model for Scalable Bitstreams. In *Proceedings of the 2007 IEEE International Conference on Multimedia and Expo*, pages 240–243, July 2007, Beijing, China

12. D. De Schrijver, W. De Neve, K. De Wolf, D. Van Deursen, and R. Van de Walle. Exploitation of Combined Scalability in Scalable H.264/AVC Bitstreams by Using an MPEG-21 XML-Driven Framework. *Lecture Notes in Computer Science – Advanced Concepts for Intelligent Vision Systems*, volume 4678, pages 699-710, August 2007

13. D. Van Deursen, W. De Neve, D. De Schrijver, and R. Van de Walle. Automatic Generation of generic Bitstream Syntax Descriptions Applied to H.264/AVC SVC Encoded Video Streams. In *Proceedings of the 14th International Conference on Image Analysis and Processing*, pages 382–387, September 2007, Modena, Italy

14. D. Van Deursen, S. De Bruyne, W. Van Lancker, W. De Neve, D. De Schrijver, H. Hellwagner, and R. Van de Walle. MuMiVA: a Multimedia Delivery Platform using Format-agnostic, XML-driven Content Adaptation. In *Proceedings of the 9th International Symposium on Multimedia*, pages 131–138, December 2007, Taichung, Taiwan

15. W. De Neve, S. Yang, D. Van Deursen, C. Kim, Y.M. Ro, and R. Van de Walle. Analysis of BSDL-Based Content Adaptation for JPEG 2000 and HD Photo (JPEG XR). In *Proceedings of the 5th International Conference on Visual Information Engineering: Workshop on Scalable Coded Media Beyond Compression*, pages 717–722, July 2008, Xi'an, China

16. D. Van Deursen, C. Poppe, G. Martens, E. Mannens, and R. Van de Walle. XML to RDF Conversion: a Generic Approach. In *Proceedings of the 4th International Conference on Automated Production of Cross Media Content for Multi-channel Distribution*, pages 138–143, November 2008, Florence, Italy

17. D. Van Deursen, W. Van Lancker, T. Paridaens, W. De Neve, E. Mannens, and R. Van de Walle. NinSuna: a Format-independent Multimedia Content Adaptation Platform based on Semantic Web Technologies. In *Proceedings of the 10th International Symposium on Multimedia*, pages 491–492, December 2008, Berkeley, United States

18. E. Mannens, R. Troncy, K. Braeckman, D. Van Deursen, W. Van Lancker, R. De Sutter, and R. Van de Walle. Automatic Information Enrichment in News Production. In *Proceedings of the 10th International Workshop on Image Analysis for Multimedia Interactive Services*, pages 61–64, May 2009, London, United Kingdom

19. D. Van Deursen, W. Van Lancker, W. De Neve, T. Paridaens, E. Mannens, and R. Van de Walle. Semantic Adaptation of Synchronized Multimedia Streams in a Format-independent Way. In *Proceedings of the 27th Picture Coding Symposium*, 4 pages on CD-ROM, May 2009, Chicago, United States

20. S. Coppens, E. Mannens, D. Van Deursen, and R. Van de Walle. Semantic Bricks for Performing Arts Archiving and Dissemination. Accepted for publication in *the Insurance Accounting & Systems Association (IASA) 2009 Annual Educational Conference and Business Show*

Chapter 2

Format-independent content adaptation

All truths are easy to understand once they are discovered; the point is to discover them.

Galileo Galilei (1564 - 1642)

2.1 Introduction

Nowadays, we face a growing diversity in devices that are able to consume multimedia content. These devices have varying characteristics such as screen size, processing power, and battery life. Furthermore, network technologies, used to transport the multimedia content to the end-user, may differ in terms of bandwidth, jitter, and error robustness. Therefore, the delivery of multimedia content needs to occur in a transparent way in order to obtain Universal Multimedia Access (UMA).

Multimedia content customization is a well-established field due to the UMA paradigm. A multimedia content customization system tries to meet the user needs by customizing the content based on the usage environment and user preferences. To take into account the restrictions of the usage environment, a wide variety of multimedia customization approaches exist in the compressed domain (as discussed in Chapter 1), such as content selection, transcoding, scalable coding, and transmoding [18, 77]. Furthermore, user preferences can be fulfilled by applying semantic adaptations such as dropping violent scenes or scene-of-interest selection in video streams.

Over the last few years, the number of multimedia coding standards has

grown significantly, particularly with the introduction of new formats such as H.264/Advanced Video Coding (H.264/AVC, [68]), Advanced Audio Coding (AAC, [62]), and JPEG XR [115]. At the same time, multimedia delivery systems must still deal with a few of the older standards such as H.262/MPEG-2 Video [58], JPEG2000 [19], and MPEG-1 Audio [56]. In order to deal with current and future multimedia coding formats, format-independent adaptation systems are gaining importance.

This chapter gives an overview of existing format-independent adaptation techniques. Section 2.2 elaborates on the general principles of a format-independent adaptation system. We discuss the main target adaptation operations in Section 2.3 and apply them to a number of existing coding formats. Next, existing technologies are discussed in Section 2.4. A list of the remaining challenges regarding format-independent content adaptation techniques is identified in Section 2.5. Finally, conclusions are drawn in Section 2.6.

2.2 Overall Approach

Current coding-format independent content adaptation techniques rely on automatically generated BSDs. In the remainder of this dissertation, these techniques will be further referred to as *description-driven adaptation*.

2.2.1 Adaptation Chain

A description-driven adaptation framework typically consists of three main processes, which are illustrated in Figure 2.1.

(1) *BSD generation*: given a media bitstream, a BSD is generated containing information about the high-level structure of the bitstream. In particular, a BSD describes how the bitstream is organized in layers or packets of data. Note that a BSD is not meant to replace the original binary data; it rather acts as an additional layer, similar to metadata. Therefore, information occurring in a BSD is called *structural metadata*. Also, a BSD is typically expressed by making use of XML.

(2) *BSD transformation*: the actual adaptation process takes place in the XML domain during the BSD transformation. The BSD is transformed (e.g., by dropping descriptions of layers or packets) according to the constraints of a given usage environment (e.g., available bandwidth and screen resolution).

(3) *Bitstream generation*: the process for the generation of the adapted bitstream takes as input the transformed BSD and the original bitstream.

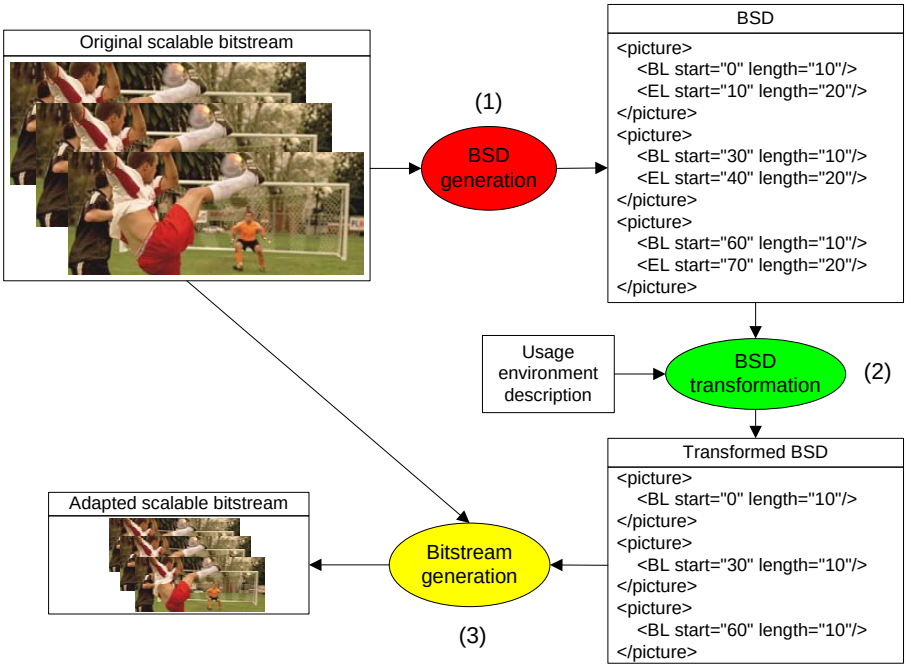


Figure 2.1: Exploiting spatial scalability using a description-driven adaptation framework. BL and EL denote Base Layer and Enhancement Layer respectively.

The transformed BSD is used to steer the generation of an adapted bitstream, which is then suited for playback in a given usage environment. This is mainly done by copying byte ranges described in the transformed BSD from the original bitstream to the adapted bitstream.

Thanks to the use of XML-based BSDs, already existing tools for manipulating XML documents can be used. Two different approaches exist for transforming XML documents.

- The Document Object Model (DOM, [75]) is a standard object model for representing XML formats in a platform- and language-independent way. It provides an interface allowing to dynamically access and update the content, structure, and style of XML documents. Furthermore, it allows a straightforward mapping to the XPath model [142]. However, this straightforward mapping implies that the full XML document needs to be available in order to correctly evaluate the XPath expressions.
- The Simple API for XML (SAX) is a serial access parser API for XML. SAX provides a mechanism for reading data from an XML document.

An XML parser based on SAX acts as a stream parser, based on events (i.e., SAX events). For each SAX event, a number of callback methods can be called when the event occurs. In contrast to DOM, SAX parsing is unidirectional; i.e., previously parsed data cannot be re-read without starting the parsing operation again. A workaround for this issue is buffering SAX events that are needed further in the parsing process. Note that, unlike DOM, there is no formal specification for SAX (the Java implementation[1] of SAX is considered to be normative). Also note that it is not straightforward to map SAX to the XPath model because of the streaming behaviour of SAX; only some experimental approaches (i.e., so-called 'streaming XPath engines') exist [10, 95].

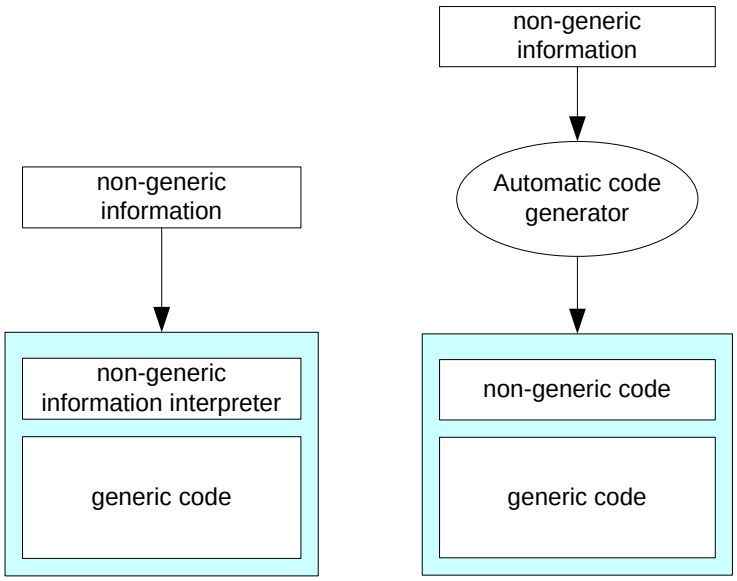
XML filters can be implemented by making use of a procedural programming language in order to create an XML parser with additional transformation logic. An alternative is to use a format-agnostic transformation engine that is able to interpret different transformation stylesheets. These stylesheets use a standardized (XML-based) language to describe the transformation logic. Examples of such XML-based languages are Extensible Stylesheet Language Transformations (XSLT, [71]), which relies on DOM, and Streaming Transformations for XML (STX, [20]), which relies on SAX. Note that examples of XSLT and STX stylesheets are available in Annex A.

As discussed in [39] and [42], SAX-based transformation of BSDs is a feasible solution because of its high execution speed, low memory usage, and streaming capabilities. STX can be used when format-independent, SAX-based transformation engines are desired. Hence, XML transformations needed for performance evaluations in this dissertation are implemented by making use of STX.

2.2.2 Generic Software Modules

As illustrated in Figure 2.2, two possibilities exist to obtain software modules that are independent of a particular input format. The first possibility (Figure 2.2(a)) is to create a generic software module that is able to interpret a document describing the non-generic (i.e., input-format specific) information. The second possibility (Figure 2.2(b)) is to automatically generate a (non-generic) software module for a particular input format. Although the obtained software module is not generic, it is generated in a generic way. The automatic code generator takes as input a document describing the non-generic information and generates the desired software module.

[1] `http://www.saxproject.org/`



(a) Interpreting non-generic information.

(b) Automatic non-generic code generation.

Figure 2.2: Obtaining generic software modules.

Using BSDs for adapting media bitstreams enables the creation of a coding-format independent multimedia content adaptation engine, since the three main processes in a description-driven adaptation chain (discussed in Section 2.2.1) can be represented by generic software modules.

- During the BSD generation process, the input media bitstream needs to be parsed. Because this parsing process is dependent on the input coding format, descriptions containing the high-level structure of particular coding formats are needed to steer the BSD generation process. It is important to note that the BSD generation process can also occur during the encoding process of media bitstreams, when all the information necessary for the creation of a BSD is available. In this case, no generic software module is needed for the BSD generation process.

- The BSD transformation engine is independent of the underlying coding format because it operates on the BSD level (i.e., XML level). Furthermore, generic software modules for the transformation of XML documents are provided by XML transformation stylesheet languages such as XSLT or STX.

- The bitstream generation process needs to generate a media bitstream encoded in a particular coding format based on the original bitstream and the transformed BSD. Similar to the BSD generation process, the bitstream generation process needs descriptions containing the high-level structure of particular coding formats to correctly interpret the transformed BSD and to write adapted media bitstreams compliant with the coding format of the original bitstream.

2.2.3 Advantages

Using description-driven multimedia content adaptation results in the following advantages.

- The bitstream parsing process is *abstracted*. As BSDs are typically generated in an automatic way, it is no longer required to implement cumbersome bitstream parsing operations in a particular programming language in order to discover the bitstream structure.

- The adaptation process occurs in the *XML domain*. The complexity of the adaptation process is shifted from the compressed domain to the XML domain. This enables the use of many already existing XML tools for manipulating BSDs, such as XSLT or STX.

- A *coding-format independent* multimedia content adaptation engine can be implemented. As discussed in Section 2.2.2, using high-level XML descriptions for adapting multimedia content enables the use of coding-format independent software modules within an adaptation framework. Hence, the software supports future coding formats without having to be rewritten, is suited for hardware implementations, and can be used for the adaptation of still images, audio, and video bitstreams.

2.3 Target Adaptation Operations

Since description-driven adaptation enables the creation of a generic, format-agnostic adaptation engine and tries to abstract the adaptation process, some restrictions regarding the adaptation possibilities need to be taken into account. Obtaining format-independent content adaptation implies that only high-level bitstream structures can be removed and that only high-level syntax elements can be modified. In other words, the compressed media bitstreams need to be encoded in such a way that it is possible to perform the adaptations without the need of a complete or partial recode process. For example, techniques such as

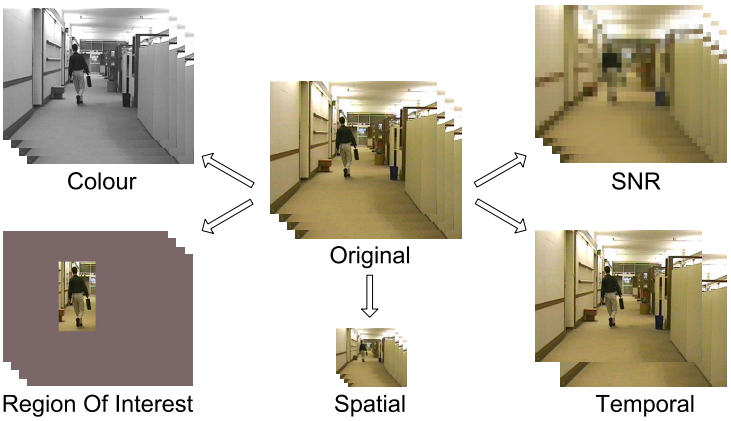


Figure 2.3: Different forms of scalability in video bitstreams.

requantization transcoding in video bitstreams [27] are too low-level and too coding-format specific for generic adaptation systems.

In this dissertation, we focus on two kinds of adaptation operations: structural and semantic. However, these adaptations are only possible if some conditions are met with respect to the structure of the media bitstream, as will be discussed in the next subsections.

2.3.1 Structural Adaptations

The main target adaptation operation for description-driven adaptation is the exploitation of scalability in media bitstreams (i.e., performing structural adaptations). Structural adaptations are typically performed to adapt media bitstreams in order to meet the terminal and network characteristics of the end-user. Hence, scalable coding is an important tool to realize a UMA environment. It enables the extraction of multiple (lower quality) versions of the same media resource without the need of a complete recoding process. The bitstream extraction process typically involves the removal of particular data blocks and the modification of the value of certain syntax elements [91]. As discussed above, such operations can easily be executed using description-driven adaptation. However, it is obvious that performing structural adaptations using description-driven content adaptation can only be realized when the media bitstreams already contain scalability layers.

Examples of resulting video bitstreams after the exploitation of scalability along a particular scalability axis are shown in Figure 2.3. Spatial scalability adjusts the resolution of the video bitstream; temporal scalability lowers the

video frame rate; colour scalability drops colour components; Signal-to-Noise Ratio (SNR) scalability decreases the visual quality of the video bitstream; Region Of Interest (ROI) scalability only decodes a specific region of the video bitstream. Examples of scalable coding formats are Scalable Video Coding (SVC, [107]), JPEG2000 [19], and Advanced Audio Coding Scalable Sample Rate (AAC SSR, [62]). Note that in the context of video bitstreams, temporal scalability is by default available and can be realized by dropping frames which are not used by other frames for prediction.

Colleagues within Multimedia Lab (i.e., Wesley De Neve, Davy De Schrijver, and Peter Lambert) have used a number of coding formats as test cases for format-independent content adaptation techniques. An overview of these coding formats and their adaptivity provisions is given below.

H.264/AVC

In the first version of the H.264/AVC specification, exploitation of multi-layered temporal scalability and a form of Region Of Interest (ROI) scalability are available. Because pictures can consist of a mixture of different types of slices (i.e., I, P, and B slices) and B slices can be used as a reference slice, it is recommend to rely on sub-sequences for achieving temporal scalability in H.264/AVC [30]. A sub-sequence represents a number of inter-dependent pictures that can be disposed without affecting the decoding of any other sub-sequence in the same sub-sequence layer or any sub-sequence in any lower sub-sequence layer. Note that sub-sequences are typically created by relying on a hierarchical coding pattern.

An enhanced way of exploiting temporal scalability in H.264/AVC is to insert placeholder slices instead of dropping slices [28]. A placeholder or skipped slice is defined as a slice that is identical to a certain reference slice, or that is reconstructed by relying on a well-defined interpolation process between different reference slices. Note that this approach allows to maintain synchronization with other media streams in a particular container format, especially when varying Group Of Pictures (GOP) structures are in use.

ROI coding can be realized in H.264/AVC using the Flexible Macroblock Ordering (FMO) tool. By using H.264/AVC FMO type 2, rectangular regions can be coded independently from each other (i.e., each region corresponds to a slice group). Extracting a particular ROI corresponds to the detection of coded P and B slices located in non-ROI slice groups and the substitution of these slices with placeholder slices [74]. One drawback of this substitution approach is the introduction of drift in the decoded sequence because of a mismatch between the reference frames in the encoder and in the decoder.

SVC

SVC supports adaptation operations along three scalability axes: the temporal, spatial, and Signal-to-Noise Ratio (SNR) axis. Extraction of lower-quality versions of SVC bitstreams along these three axes relies on Supplemental Enhancement Information (SEI) messages and syntax elements located in the NALU header as elaborated on in [35] and [37].

Similar to H.264/AVC, ROIs in SVC can be defined by using the FMO coding tool. However, since SVC allows to define multiple quality layers, the ROI extraction is performed by removing one or several quality layers from the slices belonging to slice groups that are not corresponding to the ROI. Hence, in contrast to the substitution approach performed for H.264/AVC (see above), ROI extraction in SVC does not introduce drift effects [40]. It is important to notice that, due to the introduction of profiles, the use of FMO in SVC is restricted. More specifically, FMO is prohibited in an SVC base layer and only FMO type 2 is supported in the higher layers by the Scalable Baseline Profile (other FMO types are not supported).

Finally, SVC supports interactive ROI extraction. Interactive ROI extraction can be used in applications in which the ROI cannot be defined during the encoding phase of the video sequence. In order to support interactive (rectangular) ROI scalability, a video frame has to be divided into different tiles that can be selected on an individual basis. Each tile has to be coded as an individual slice such that the tiles can be decoded independently of other tiles. This way, the tiles belonging to a certain ROI can be selected on-the-fly during the extraction process. FMO type 0 (also called interleaved slice groups) can be used to obtain a tiled slice partition [38].

Video Codec-1

Similar to H.264/AVC, Video Codec-1 (VC-1, [112]) is a single-layered codec. Hence, only the exploitation of temporal scalability is possible. Note that the adaptation process is less complex than in H.264/AVC because B pictures cannot be used as reference frames. Hence, temporal scalability is obtained by eliminating B and Skipped pictures [29].

JPEG2000 and JPEG XR

In the context of description-driven adaptation systems, JPEG XR provides support for spatial scalability, lossless-to-lossy degradation, and tile-aligned ROI extraction. Adaptation provisions of JPEG2000 are spatial scalability, SNR scalability, color component scalability, lossless-to-lossy degradation,

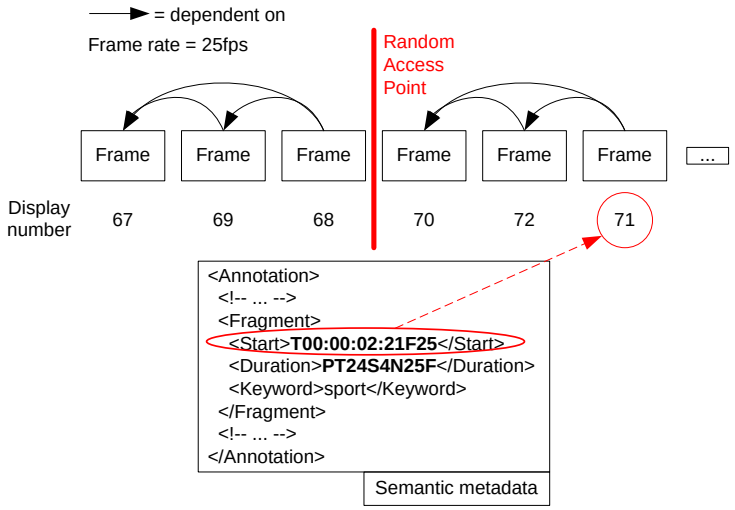


Figure 2.4: Mapping content metadata to display numbers of media bitstreams.

and tile-aligned ROI extraction. A discussion of these adaptation operations in the XML domain can be found in [32] and [33].

2.3.2 Semantic Adaptations

Next to the exploitation of scalability, high-level adaptations based on semantic information about the multimedia content can be realized. Examples of semantic adaptations are the selection of a Region Of Interest (ROI) or the selection of specific temporal segments that are of interest to the user. In this dissertation, we focus on semantic adaptations along the temporal axis (i.e., scene or shot selection/removal), based on content metadata. A content metadata excerpt is shown in Figure 2.4. Timing information in content metadata is usually expressed in terms of timestamps that may be formatted in different ways (in our example, the MPEG-7 datatypes for time and duration are used). However, compressed media bitstreams are addressed in terms of byte ranges corresponding to parse units (i.e., frames), which are in their turn related to display numbers (i.e., numbers indicating the order of display). Hence, in order to use content metadata for performing the selection of a certain scene in a media bitstream, a match between byte ranges and timing information in terms of timestamps needs to be obtained.

Next to finding a mapping between timing information and byte ranges, taking into account the dependencies between different frames within com-

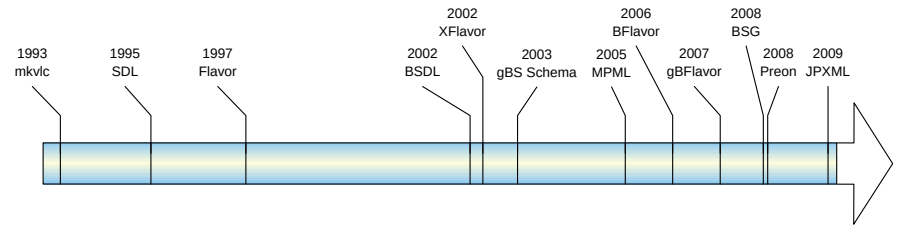


Figure 2.5: Timeline overview of existing bitstream syntax description tools.

pressed media bitstreams is another important issue. For instance, intra-coded frames are independent of other frames, while inter-coded frames are dependent on previous and/or future frames. To garantuee that the adapted media bitstream can be correctly decoded, cuts in the media bitstream should only be performed at random access points (as illustrated in Figure 2.4). Random access refers to the ability of a decoder to start decoding at a point in a video sequence other than at the beginning and to recover an exact or approximate representation of the decoded pictures [49].

2.4 Existing Technologies

In recent years, a number of bitstream syntax description tools have been developed. In Figure 2.5, an overview of existing bitstream syntax description tools is visualized on a timeline. The Syntax Description Language (SDL, [7]) provides a formal way to describe the entire structure of a bitstream. Its ancestor was a Perl script called mkvlc, which automatically generated C code for variable-length coding table declaration. SDL was needed in the MPEG-4 standardization activity, which was moving in a direction of flexible, even programmable, audio-visual decoding systems. The Formal Language for Audio-Visual Object Representation (Flavor, [44]), built on top of SDL, is an object-oriented media representation language designed for simplifying the development of applications that involve a significant media processing component (coding, editing, manipulation, etc.). It provides a formal way for describing any coded audio-visual or general media bitstream, and it comes with a translator that can automatically generate C++/Java code from the Flavor description. The generated code can readily be used as a bitstream parser, generator, or tracing tool. Finally, Flavor was extended with XML features (XFlavor, [51]), implying that the generated C++/Java code can also include a method for producing XML documents that correspond to the bitstreams described by Flavor. More information regarding XFlavor is provided in Section 2.4.3.

Two technologies were created and standardized within the MPEG-21 Multimedia Framework [17], which aims at realizing the 'big picture' in the multimedia production, delivery, and consumption chain. One important part of the framework, Digital Item Adaptation (DIA, [61]), provides several tools that can be used for creating an interoperable and description-driven adaptation framework: the Bitstream Syntax Description Language (BSDL, [4]) and the generic Bitstream Syntax Schema (gBS Schema, [121]), which are discussed in Section 2.4.1 and Section 2.4.2 respectively.

BFlavor (BSDL + XFlavor, [31, 126]) was developed to combine the strengths of BSDL and XFlavor and to eliminate their weaknesses. Section 2.4.4 discusses BFlavor in more detail. Further, gBFlavor, which is built on top of BFlavor, provides an efficient method for the generation of generic Bitstream Syntax Descriptions (gBSDs), i.e., BSDs compliant to gBS Schema. Chapter 3 contains an in-depth discussion of gBFlavor.

The MPEG Video Markup Language (MPML, [116]) is an XML application specifically designed for describing the syntax of bitstreams compliant with MPEG-4 Visual [60]. JPEG 2000 Part 14: XML structural representation & reference, also known as JPXML [66], standardises an XML representation for a JPEG 2000 file. The latter is expected to be finalized in Spring 2009. Both MPML and JPXML will not be discussed in further detail due to their rather format-specific nature.

Preon is an open source project[2] that allows to declaratively bind a Java based object model to its compressed binary encoded representation. The mapping declaration between the Java based representation and the encoded representation results in a decoder, an encoder, and hyperlinked documentation of the encoding format. Because Preon does not provide support for the generation of text-based descriptions of binary bitstreams, it will not be discussed in further detail.

Finally, Bitstream Segment Graphs (BSG, [50]) are designed as a complete, generally applicable and machine-processable model for coding format instances. Similar to Flavor and Preon, BSG's main motivation is to automatically generate parser software for coding formats. It is focussed on modelling low-level structures of coding formats and does not provide support for the generation of text-based descriptions of binary bitstreams. Hence, BSG will not be discussed in further detail.

[2] `http://preon.sourceforge.net/`

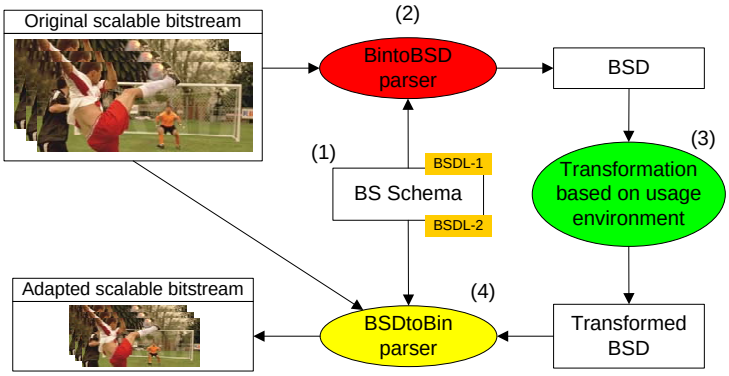


Figure 2.6: Multimedia content adaptation chain using BSDL.

2.4.1 MPEG-B BSDL

The primary motivation behind the development of BSDL is to assist in the adaptation of scalable bitstreams, such that the resulting bitstreams meet the constraints imposed by a particular usage environment. The generic character of BSDL, and hence its merit, lies in the coding-format independent nature of the logic that is responsible for the creation of BSDs and for the generation of the adapted bitstreams. This is possible due to the fact that all information, necessary for discovering the structure of a bitstream, is available in a document called a Bitstream Syntax Schema (BS Schema) [4]. The language constructs occurring in a BS Schema are built on top of the W3C XML Schema Language.

BSDL comes with two standardized, format-agnostic parsers: a BintoBSD parser that is responsible for producing BSDs and a BSDtoBin parser that is used for generating adapted bitstreams. Figure 2.6 shows the adaptation chain for BSDL. Explanatory notes are given below.

(1) A BS Schema is created for a particular coding format (i.e., the coding format of the media bitstream that needs to be adapted). Such a BS Schema contains a description of (a part of) the syntax of that particular coding format in the BSDL schema language. It consists of, among other things, information about the syntax element names, their datatypes, and their position in the syntactical structures of the coding format.

(2) The BS Schema is used by the BintoBSD parser to automatically generate a BSD for a given (scalable) bitstream.

(3) The BSD is transformed to meet the constraints of a certain usage envi-

ronment. Note that the way the BSD is transformed is not standardized by DIA. As discussed in Section 2.2.1, already existing tools for transforming XML documents can be used for the transformation of a BSD.

(4) An adapted bitstream is created by using the BSDtoBin parser, which takes as input the BS Schema, the customized BSD, and the original bitstream.

The language specification of BSDL introduces two normative successive sets of extensions and restrictions over the W3C XML Schema language. The extensions are expressed by means of two XML Schemas, while the restrictions are fixed in the standards document itself. The schema for BSDL-1 extensions defines a number of attributes and datatypes that can be used in a BS Schema. Both the BintoBSD and BSDtoBin parsers need these extensions for the correct generation of BSDs and the creation of adapted bitstreams. An example of such a datatype is the *byteRange* datatype, which can be used to point to a data range in the original bitstream when it is too verbose to be included in the BSD. The schema for BSDL-2 extensions defines a number of additional extensions to XML Schema that are needed by the BintoBSD parser to resolve ambiguities during the bitstream parsing process (i.e., control flow). Examples of such extensions are attributes expressing conditions indicating whether particular syntax elements need to be parsed or not (i.e., the *if* and *ifNext* attributes). Excerpts of examples of BSDL can be found in Annex A.

BSDL was originally developed as part of MPEG-21 DIA, but is now a standalone standard known as part 5 of MPEG-B [67]. The main goal of MPEG-B (MPEG systems technologies) is to specify technologies which can universally be used in the context of multimedia systems (e.g., methods for efficiently transmitting and compressing XML documents).

2.4.2 MPEG-21 gBS Schema

In contrast to BSDL and XFlavor, gBS Schema enables the creation of *format-independent XML descriptions*, which are called generic Bitstream Syntax Descriptions (gBSDs). More specifically, the syntax elements occurring in a gBSD are independent of the coding format and are specified in the generic BS Schema[3]. The functioning of a gBS Schema-based adaptation framework is shown in Figure 2.7. Explanatory notes are given below.

(1) The first step is the generation of a gBSD. This process is not described in the DIA specification since only the gBS Schema is described in the

[3] In contrast to BSDL, only one BS Schema is necessary, i.e., the generic BS Schema.

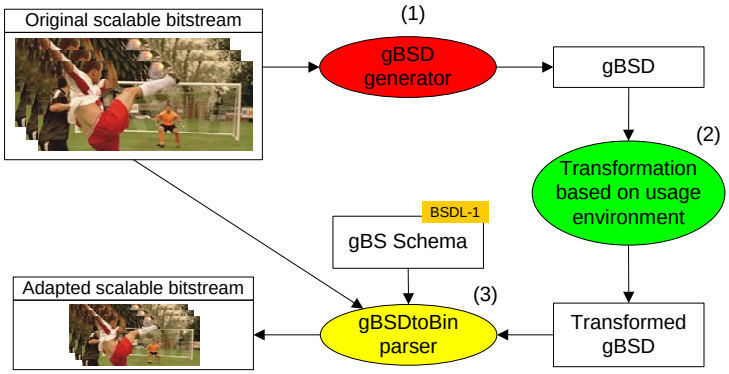


Figure 2.7: Multimedia content adaptation chain using gBS Schema.

specification, together with the behaviour of a gBSDtoBin parser. This implies that a gBSD may be generated in any proprietary way.

(2) According to the given usage environment constraints, the gBSD is transformed using existing XML transformation techniques.

(3) Finally, an adapted bitstream is obtained using the standardized gBSDtoBin parser, which takes as input the original bitstream and the transformed gBSD. The gBSDtoBin parser relies on the gBS Schema to steer the generation of the adapted bitstream. The latter includes the schema for BSDL-1 extensions (as discussed in Section 2.4.1), providing the necessary datatypes to correctly generate an adapted bitstream.

Excerpts of examples of gBS Schema can be found in Annex A. Note that gBS Schema will be discussed in more detail in Chapter 3, where a new gBSD generation method is proposed.

2.4.3 XFlavor

Flavor provides a formal way to specify how data are laid out in a serialized bitstream [44]. It is designed as a declarative language with a Java-like syntax to describe the bitstream syntax on a bit-per-bit basis. Its aim is to simplify and speed up the development of software that processes audio-visual bitstreams by automatically generating the required C++ or Java code to parse the data. Hence the developer can focus on the processing part of the software and does not need to deal with the parsing process. This is possible thanks to the fact

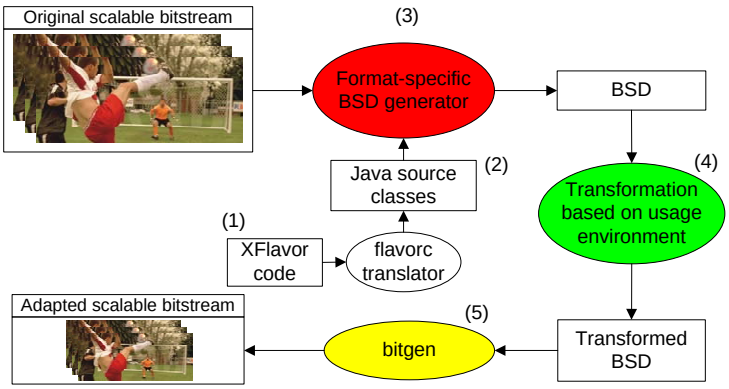


Figure 2.8: Multimedia content adaptation chain using XFlavor.

that the design of Flavor is based on the principle of separation between bitstream parsing operations and other encoding/decoding operations. Note that a Flavor-based parser does not generate a persistent description of the parsed data, but only an in-memory representation in the form of a collection of C++ or Java class objects.

XFlavor is an extension of Flavor, containing tools for generating an XML description of the entire bitstream syntax and for regenerating an (adapted) bitstream. The specification and software of XFlavor are open source and available on `http://flavor.sourceforge.net/`. An adaptation chain using XFlavor is provided in Figure 2.8. Explanatory notes are provided below.

(1) The first step is the creation of an XFlavor code, which contains a Java-like description of the syntax of a particular coding format. Note that the information included in an XFlavor code is comparable to the information occurring in a BS Schema. They both describe the high-level syntax structures and datatypes of a particular coding format.

(2) The flavorc translator is able to automatically translate an XFlavor code to Java or C++ source classes. For instance, code responsible for I/O operations during the parsing process of the media bitstream is automatically generated and integrated in the resulting Java or C++ source classes. These source classes can then be compiled using a Java or C++ compiler, resulting in a BSD generator that is specific for media bitstreams compliant with the coding format descibed by the initial XFlavor code.

(3) The coding-format specific BSD generator automatically generates a

BSD for a given input media bitstream.

(4) According to the given usage environment constraints, the BSD is transformed using existing XML transformation techniques.

(5) Finally, XFlavor's bitgen tool is used for translating the customized BSD into an adapted bitstream.

A remarkable difference between XFlavor and the MPEG-21 solutions (i.e., BSDL and gBS Schema) is the fact that a BSD in XFlavor cannot refer to the original bitstream. More specifically, in XFlavor, the complete bitstream data are actually embedded in the BSD, resulting in potentially verbose descriptions, while BSDL and gBS Schema use specific datatypes (specified in the schema for BSDL-1 extensions) to point to a data range in the original bitstream when it is too verbose to be included in the BSD. Excerpts of examples of XFlavor can be found in Annex A.

2.4.4 BFlavor: Optimizing MPEG-B BSDL

Although BSDL and XFlavor can be used as stand-alone tools [30], a harmonized approach can combine the strengths of the two technologies. In XFlavor, the bitstream generator (i.e., bitgen) only uses information from the BSD and thus is independent of the XFlavor code. Hence, the complete bitstream data are actually embedded in the BSD, resulting in potentially huge descriptions. On the contrary, BSDL makes use of a specific datatype to point to a data range in the original bitstream when it is too verbose to be included in the description (i.e., by making use of the language construct `bs1:byteRange`). This results in BSDs containing only the high-level structure of the bitstream. The strengths of XFlavor are the fast execution speed and the low and constant memory consumption of the coding format-specific parser, while BSDL's BintoBSD Parser struggles with an unacceptable execution speed and increasing memory consumption caused by an inefficient XPath evaluation process [31]. This is due to the fact that the entire description of the bitstream structure is kept in system memory in order to allow the evaluation of arbitrary XPath 1.0 expressions.

BFlavor bridges the gap between BSDL and XFlavor [126]. It is developed to combine the strengths of BSDL and XFlavor, i.e., to generate a compact high-level BSD at a fast execution speed and with a constant memory consumption. It is built on top of XFlavor by defining a number of restrictions and extensions (similar to the way BSDL is build on top of W3C XML Schema). By using the automatically generated parser of BFlavor, it is possible to generate BSDs that can be further processed by the BSDtoBin Parser of BSDL.

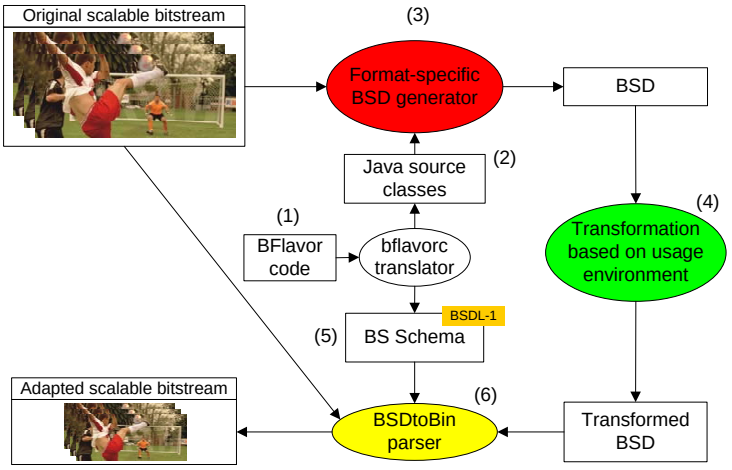


Figure 2.9: Multimedia content adaptation chain using BFlavor.

Figure 2.9 provides an overview of BFlavor's adaptation chain; explanatory notes are given below.

(1) A BFlavor code contains a Java-like description of the syntax of a particular coding format, similar to a corresponding XFlavor code. More information about the BFlavor specification is provided in Chapter 3.

(2) The bflavorc translator uses this code to generate Java source classes that can be compiled to a coding-format specific BSD generator.

(3) So far, the XFlavor approach has been followed. From this point, a switch is made to the BSDL approach. More specifically, the coding-format specific BSD generator produces a BSD which can be further processed by BSDL's BSDtoBin Parser.

(4) According to the given usage environment constraints, the BSD is transformed using existing XML transformation techniques.

(5) The BSDtoBin parser needs a BS Schema in order to correctly generate an adapted bitstream. The information necessary to create such a BS Schema is available in the BFlavor code. Hence, a mapping can be created between BFlavor and BSDL. This way, the bflavorc translator is also able to automatically generate a BS Schema based on the BFlavor code. This BS Schema can then be used by the BSDtoBin parser. Note that the resulting BS Schema makes only use of the schema for BSDL-1 extensions, since this is sufficient for the BSDtoBin parser.

(6) Finally, an adapted media bitstream is generated by the BSDtoBin parser, taking as input the transformed BSD, the generated BS Schema, and the original media bitstream.

As a result of this approach, it is possible to generate BSDs compliant with BSDL with the fast execution speed and the low memory consumption of the coding-format specific BSD generator of XFlavor. At the same time, the generated BSDs are compliant with BSDL and thus compact because they only contain a description of the high-level structure of the (scalable) bitstream.

BFlavor was developed and evaluated in the context of the author's master's thesis [124]. A number of refinements and optimizations were developed in the context of this dissertation. Furthermore, BFlavor served as a basis for the development of gBFlavor, which is a novel method for the generation of generic Bitstream Syntax Descriptions (gBSDs). Chapter 3 contains a detailed discussion of gBFlavor and the refinements made on top of BFlavor. Note that a Web page[4] has been set up, providing the full specification, a user manual, and a number of examples for BFlavor.

Since BFlavor is a proprietary approach, one solution for the performance issues of the BintoBSD process, which can be considered fundamental in the context of media bitstreams, is to enhance the BSDL standard. These enhancements have been adopted in the second amendment of the MPEG-21 DIA specification [65].

- Context Management: the in-memory representation of a BSD, which is commonly referred to as the context, is needed by the BintoBSD process for the correct evaluation of an arbitrary set of XPath 1.0 expressions. These XPath expressions are for example used to provide support for conditional parsing. Five new attributes are introduced in the schema for BSDL-2 extensions to support context management in a BS Schema [36]. Using these attributes, it is possible for the BintoBSD parser to achieve a minimal memory consumption and a constant BSD generation speed on the one hand, while still allowing the use of the entire XPath 1.0 specification on the other hand [34].

- User-defined XPath variables: they allow to cache frequently used node sets (e.g., the parameter sets in H.264/AVC) as object arrays. Besides the simplification of the notation of XPath expressions, the evaluation of XPath expressions can be sped up by reducing the number of predicates and the length of the location steps.

[4] Available at `http://multimedialab.elis.ugent.be/BFlavor/`.

2.5 Challenges

Although description-driven adaptation of media bitstreams seems to be a very promising technique in the context of multimedia delivery in UMA environments, there are still some remaining challenges. In the next three chapters of this dissertation, solutions are proposed for these challenges.

2.5.1 gBSD Generation Process

The first challenge is specific for the MPEG-21 gBS Schema description tool. As discussed in Section 2.4.2, the gBSD generation process is not described in the DIA specification. Only the gBS Schema is described in the standard in question, together with the behaviour of a gBSDtoBin parser. Hence, a gBSD may be generated in any proprietary way. However, as stated in [94] and [117], there exists no format-agnostic parser to produce gBSDs, despite the fact that gBS Schema is format-agnostic. Since a format-independent adaptation framework with generic software modules is desired, the gBSD generation process must be implemented in the form of a generic software module.

One possibility to generate gBSDs in a format-agnostic way is to transform format-specific BSDs (e.g., created with BSDL) into gBSDs [94]. The transformation can be done using common XML transformation technologies. Although this approach is format-agnostic, it has a number of disadvantages. On the one hand, two different technologies (i.e., the BSD generation and transformation technique) have to be used. On the other hand, the format-specific BSDs contain a lot more detail than is needed for the resulting gBSD, implying a decrease in execution speed of the format-specific BSD generation process. This amount of detail is needed for setting proper markers in the resulting gBSD. Note that a more detailed explanation of this format-agnostic solution and its problems will be discussed in Chapter 3. Further, in Chapter 3, we present a new gBSD generation method, called gBFlavor, enabling the automatic generation of gBSDs in a coding-format independent way. gBFlavor is a technology that enables the automatic generation of a format-specific parser. This parser is subsequently able to produce a gBSD for a given bitstream. gBFlavor is built on top of BFlavor, which is an efficient alternative for the generation of BSDs compliant with BSDL in terms of execution time (as discussed in Section 2.4.4).

2.5.2 Defining Adaptation Operations

Adaptation operations within a description-driven adaptation framework are currently expressed by means of XML transformations (e.g., XSLT or STX

stylesheets). Examples of such an adaptation operation implemented in XSLT and STX can be found in Listing A.6 and Listing A.7 respectively. This adaptation operation corresponds to the removal of non-referenced B slices (i.e., a trivial form of temporal scalability). It is important to mention that, although the underlying adaptation engines in a description-driven adaptation system are independent of the coding format, the XML descriptions themselves (i.e., the BSDs) are coding-format specific. Note that this is also the case for generic BSDs (i.e., BSDs compliant to gBS Schema), since the hierarchical structure of the BSD and the labels of the (generic) XML elements are coding-format specific. Hence, because of the coding-format specific character of BSDs, defining a (description-driven) adaptation operation requires knowledge of the underlying coding format. For example, two different XML transformations are needed for the adaptation operation 'frame rate scaling' (i.e., exploitation of temporal scalability) applied to H.264/AVC and MPEG-2 Video. This is necessary because the BSDs of H.264/AVC and MPEG-2 Video streams differ in terms of structure and syntax elements.

Because the coding-format dependency is shifted from the binary to the XML domain, creators of XML filters cannot think in terms of high-level adaptation operations but have to be aware of the underlying coding formats. Hence, with the current description-driven approach, format-independency is obtained in an ad-hoc manner. Therefore, a solution is proposed in Chapter 4, which enables the definition of adaptation operations on a higher level, i.e., independent of the coding format. This solution consists of the creation of a model for media bitstreams, which is needed to abstract the XML transformation process. The model provides support for exploitation of scalability along different axes and random access points, enabling the definition of structural and semantic adaptation operations in a coding-format independent way.

2.5.3 Integration with Metadata Standards

Description-driven adaptation introduces an extra layer, i.e., the structural metadata, which is directly related to the bits of a compressed media bitstream. Existing metadata standards regarding the visual content and characteristics of media bitstreams are usually independent of the coding format (e.g., MPEG-7, Dublin Core, or NewsML). More specifically, they do not refer to byte ranges of the described media bitstreams. For example, a scene description contains a start and length in terms of timestamps. In order to define semantic adaptation operations based on content metadata (e.g., scene selection), the mapping between the structural metadata (i.e., bytes) and content metadata (i.e., timestamps) needs to be calculated (see Section 2.3.2). This mapping is dependent

on the underlying coding format which implies that for different coding formats, different XML filters need to be written for the same semantic adaptation operation.

Another problem regarding the integration of metadata standards with description-driven adaptation is the lack of interoperability between those metadata standards. An XML-based metadata standard typically consists of an XML Schema accompanied with plain text which usually describes the semantics of the XML Schema. This is necessary because XML Schema is only capable of describing grammars and imposes no common interpretation of the data contained in the document. For example, the same tags in different metadata standards can have different meanings. Due to this interoperability problem, expressing the same adaptation operation (e.g., select the sport fragments) in the context of two different content metadata standards (e.g., MPEG-7 and NewsML) requires the development of two XML filters. Note that the same holds true for the structural metadata standards (e.g., BSDL and gBS Schema).

In order to obtain a seamless integration between description-driven adaptation and metadata standards, a solution based on Semantic Web technologies is proposed in Chapter 4. The model for media bitstreams (shortly discussed in Section 2.5.2) provides support for linking the structural metadata to the content metadata (i.e., for linking the bytes to the timestamps). This way, temporal semantic adaptations can be defined based on content metadata and independent of the underlying coding format. Furthermore, the model is implemented by making use of OWL, implying a seamless integration with other content metadata standards based on Semantic Web technologies.

2.5.4 Fully Integrated Description-driven Adaptation Platforms

In order to investigate the feasibility of using description-driven adaptation techniques in practice, a fully integrated adaptation platform based on description-driven adaptation needs to be designed and implemented. The platform must be able to support both structural and semantic adaptation operations. Furthermore, we want to deploy this platform in streaming environments, requiring real-time processing of the media bitstream using description-driven adaptation. Therefore, in Chapter 5, we address the design, implementation, and performance evaluation of two multimedia adaptation platforms that rely on description-driven adaptation engines.

2.5.5 Combining Adaptation and Packaging in Coding-format Independent Environments

After adapting a media bitstream, it is usually packed in a container format. Examples of container formats are the MP4 file format [59] and the Real-time Transport Protocol (RTP, [105]). Currently, the packaging process of a media bitstream is coding-format dependent. More specifically, media bitstreams need to be parsed in order to correctly fragment the bitstream and to correctly assign timestamps to these fragments. Since we are working towards a coding-format independent multimedia delivery system, the packaging process of media bitstreams needs to occur in a format-independent way.

In Chapter 5, a solution is presented which combines the adaptation and packaging processes in a format-independent way. It relies on an extension of the model for media bitstreams (see Section 2.5.2) to package media bitstreams independent of the underlying coding format. Furthermore, BSDL is used to serialize the resulting packaged media bitstream.

2.5.6 Structural Metadata Overhead

In contrast to dedicated software solutions for the adaptation of media bitstreams, description-driven content adaptation introduces structural metadata, which results in file size overhead. The structural metadata are expressed by means of XML and are therefore verbose.

An obvious solution for this problem is to compress the BSDs by using generic algorithms such as WinZip [145] or by using XML compression techniques such as Word Replacing Transform for XML (XWRT, [149]). However, these compressed BSDs need to be uncompressed before the BSD transformation process and possibly, the transformed BSD needs to be compressed again. This introduces additionally overhead in terms of execution time for the description-driven adaptation chain. An ideal solution would consist of a compression format which can be deployed in streaming environments and which supports XML transformations in the compressed domain. In [120], Timmerer *et al.* propose an approach to transform XML documents, encoded with BiM [63], in the binary domain. BiM is an XML Schema aware encoding scheme for XML documents. One of the main features of BiM is that it provides streaming capabilities for XML-based data. In [120], an event-based binary XML parser interface is proposed (i.e., BiM API for XML (BAX)), together with a number of processing instructions. The latter can be bound to existing XML transformation technologies such as XSLT or STX.

Besides compression, the BSD can be made more compact by removing information in the BSD that is not necessary for the BSD transformation or

bitstream generation process. In Chapters 3 and 4, methods for creating compact BSDs are proposed.

2.6 Conclusions and Original Contributions

Due to the growth of multimedia coding standards, format-independent adaptation systems are gaining importance. These adaptation systems can deal with current and future multimedia coding formats. In this chapter, we gave an overview of existing format-independent content adaptation techniques and elaborated on its main target adaptation operations. Furthermore, existing technologies were discussed and an overview was given of a number of remaining challenges in the context of format-independent content adaptation.

Current format-independent content adaptation techniques rely on automatically generated textual (XML-based) descriptions, called BSDs. They typically contain information about the high-level structure of a media bitstream. The actual adaptation takes place in the XML domain during the transformation of the XML description (e.g., by dropping descriptions of layers or packets). This transformation process takes into account the constraints of a given usage environment (e.g., available bandwidth and screen resolution). In this chapter, we elaborated on the three-step-based adaptation chain needed to obtain format-independent adaptation, i.e., BSD generation, BSD transformation, and adapted bitstream generation. We discussed how this adaptation chain could be implemented by means of generic, coding-format independent software modules. Furthermore, we gave an overview of existing format-independent adaptation techniques including MPEG-B BSDL, MPEG-21 gBS Schema, and XFlavor. Within the author's master's thesis, BFlavor was developed and evaluated. BFlavor bridges the gap between BSDL and XFlavor and is developed to combine the strengths of BSDL and XFlavor, i.e., to generate a compact high-level BSD at a fast execution speed and with a constant memory consumption.

One important precondition to use format-independent content adaptation techniques is that the adaptations can be realized by performing simple editing operations such as the removal of high-level syntax structures or the modification of high-level syntax elements. In this chapter, we identified two major categories of target adaptation operations for format-independent content adaptation systems, i.e., structural and semantic adaptations. Structural adaptations are used to customize a media bitstream depending on the end-user's terminal and network characteristics, which is possible on condition that the media bitstream consists of a number of scalability layers. Semantic adaptations are driven by semantic information about the multimedia content. Adaptation

operations such as video summarization and scene selection are examples of semantic adaptations. These are possible on condition that media bitstreams are provided with regularly occurring random access points.

Finally, we listed a number of remaining challenges for format-independent content adaptation systems. In the remainder of this dissertation, we propose solutions for these challenges.

Our research in this domain resulted in contributions that are incorporated in the publications listed below.

1. W. De Neve, D. De Schrijver, D. Van Deursen, and R. Van de Walle. XML-Driven Bitstream Extraction Along the Temporal Axis of SMPTE's Video Codec 1. In *Proceedings of the 7th International Workshop on Image Analysis for Multimedia Interactive Services*, pages 233–236, April 2006, Seoul, South Korea

2. P. Lambert, D. De Schrijver, D. Van Deursen, W. De Neve, Y. Dhondt, and R. Van de Walle. A Real-Time Content Adaptation Framework for Exploiting ROI Scalability in H.264/AVC. *Lecture Notes in Computer Science – Advanced Concepts for Intelligent Vision Systems*, volume 4179, pages 442–453, September 2006

3. D. De Schrijver, W. De Neve, D. Van Deursen, J. De Cock, and R. Van de Walle. On an Evaluation of Transformation Languages in a Fully XML-driven Framework for Video Content Adaptation. In *Proceedings of the first International Conference on Innovative Computing, Information and Control (ICICIC06)*, volume 3, pages 213–216, September 2006, Beijing, China

4. W. De Neve, D. De Schrijver, D. Van Deursen, P. Lambert, and R. Van de Walle. Real-Time BSD-Driven Adaptation Along the Temporal Axis of H.264/AVC Bitstreams. *Lecture Notes in Computer Science – Advances in Multimedia Information Processing - PCM 2006*, volume 4261, pages 131–140, November 2006

5. D. De Schrijver, W. De Neve, D. Van Deursen, S. De Bruyne, and R. Van de Walle. Exploitation of Interactive Region of Interest Scalability in Scalable Video Coding by Using an XML-driven Adaptation Framework. In *Proceedings of the 2nd International Conference on Automated Production of Cross Media Content for Multi-channel Distribution*, pages 223–231, December 2006, Leeds, U.K.

6. D. De Schrijver, W. De Neve, K. De Wolf, P. Lambert, D. Van Deursen, and R. Van de Walle. XML-driven Exploitation of Combined Scalability

in Scalable H.264/AVC Bitstreams. In *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems*, pages 1521–1524, May 2007, New Orleans, United States

7. D. De Schrijver, W. De Neve, D. Van Deursen, Y. Dhondt, and R. Van de Walle. XML-based Exploitation of Region of Interest Scalability in Scalable Video Coding. In *Proceedings of the 8th International Workshop on Image Analysis for Multimedia Interactive Services*, 4 pages on CD-ROM, June 2007, Santorini, Greece

8. D. De Schrijver, W. De Neve, K. De Wolf, D. Van Deursen, and R. Van de Walle. Exploitation of Combined Scalability in Scalable H.264/AVC Bitstreams by Using an MPEG-21 XML-Driven Framework. *Lecture Notes in Computer Science – Advanced Concepts for Intelligent Vision Systems*, volume 4678, pages 699-710, August 2007

9. W. De Neve, S. Yang, D. Van Deursen, C. Kim, Y.M. Ro, and R. Van de Walle. Analysis of BSDL-Based Content Adaptation for JPEG 2000 and HD Photo (JPEG XR). In *Proceedings of the 5th International Conference on Visual Information Engineering: Workshop on Scalable Coded Media Beyond Compression*, pages 717–722, July 2008, Xi'an, China

10. W. De Neve, D. Van Deursen, W. Van Lancker, Y. M. Ro, and R. Van de Walle. Improved BSDL-based Content Adaptation for JPEG 2000 and HD Photo (JPEG XR). *Signal Processing: Image Communication – Special Issue on Scalable Coded Media beyond Compression*, 24(6):452–467, July 2009

Chapter 3

gBFlavor

And so you touch this limit, something happens and you suddenly can go a little bit further. With your mind power, your determination, your instinct, and the experience as well, you can fly very high.

Ayrton Senna (1960 - 1994)

3.1 Introduction

As discussed in the previous chapter, description-driven adaptation techniques are gaining importance due to the growing diversity in terms of coding formats, end-user devices, and network technologies. In this chapter, we focus on one particular description-driven adaptation technique standardized within the MPEG-21 Multimedia Framework: generic Bitstream Syntax Schema (gBS Schema). It enables the use of a generic Bitstream Syntax Description (gBSD) to steer the format-independent adaptation of a binary media resource. In contrast to BSDL (i.e., the other MPEG description-driven adaptation technique), the BSD generation process is not defined for gBS Schema. Since one of the major advantages of description-driven adaptation is its format-agnostic character, a gBSD generation method independent of the underlying coding format is desired.

In this chapter, we propose a novel solution for the automatic and format-independent generation of gBSDs. It is called gBFlavor and offers the possibility to automatically generate a coding-format specific parser that is able to produce a gBSD, given as input a particular media resource. gBFlavor is built on top of BFlavor, which is an efficient alternative for the generation of BSDs compliant with BSDL in terms of execution time (as discussed in

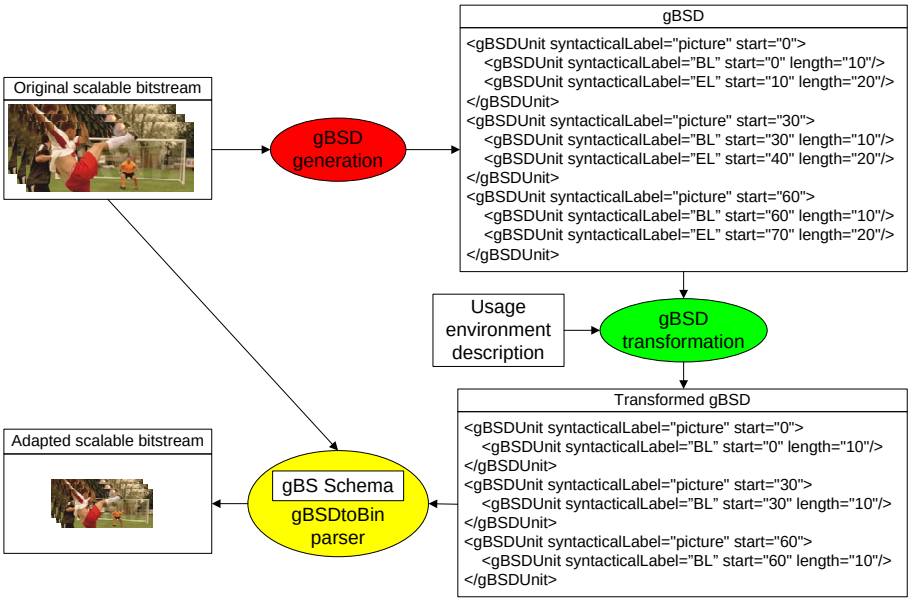


Figure 3.1: Functioning of a gBS Schema-based adaptation framework.

Section 2.4.4). This chapter gives an overview of the gBFlavor specification, which enables describing the high-level structure of a coding format and allows the insertion of semantically meaningful information into the resulting gBSD. The general functioning of a gBFlavor-enabled adaptation framework is discussed as well. JPEG2000 [19] and H.264/AVC Scalable Video Coding (SVC, [107]) are used as use cases for gBFlavor.

The outline of this chapter is as follows. Section 3.2 gives an overview of the specification and functioning of MPEG-21 gBS Schema. Next, Section 3.3 introduces gBFlavor, our new method for gBSD generation. Section 3.4 provides performance results regarding the different description generation methods. Finally, conclusions are drawn in Section 3.5.

3.2 gBS Schema

3.2.1 Functioning

The functioning of a gBS Schema-based adaptation framework is shown in Figure 3.1. The first step is the generation of a gBSD. This process is not

described in the DIA specification[1], which implies that a gBSD may be generated in any proprietary way. Note that this observation is key to a good understanding of the novel contribution of research proposed in this chapter. More details about the gBSD generation process are provided in Section 3.2.3. Subsequently, the gBSD is transformed by using common XML transformation technologies such as XSLT or STX. After the transformation of the gBSD, an adapted bitstream is obtained using the gBSDtoBin parser, which takes as input the original bitstream and the transformed gBSD. The gBSDtoBin parser relies on the gBS Schema to steer the generation of the adapted bitstream. The gBS Schema acts as a W3C XML Schema for the gBSDs and has the following properties [94].

- It is independent of the underlying coding format. Since gBS Schema acts as a W3C Schema for the gBSDs, only XML elements, regarding the structure of the bitstream, specified in the gBS Schema, can occur in the description. Hence, no format-specific XML elements are present in the gBSD, which implies that gBSDs are format-agnostic from a syntactical point of view. Furthermore, there exists only one gBS Schema which implies that there is no need to send the gBS Schema to the adaptation engine. This is in contrast with BSDL where several BS Schemas can be built according to corresponding coding formats. These BS Schemas need to be sent to the adaptation engine because the BSDtoBin parser needs these BS Schemas.

- It enables the semantically meaningful marking of syntactical elements. More specifically, semantic information can be added to the gBSDs by means of the XML attribute *marker*. This allows to include application- or domain-specific information in the gBSD, e.g., marking violent scenes within a gBSD describing an action movie.

- It provides support for hierarchical adaptations by describing hierarchies of syntactical units. Hence, gBS Schema supports the grouping of bitstream elements, allowing to vary the granularity of the gBSD.

- It contains an extensive addressing scheme to support efficient bitstream access. Supported units are bit and byte, while the addressing mode can be absolute, offset, or consecutive.

Since all the necessary information to generate the adapted bitstream (given the original bitstream) is included in the gBSD and the semantics of

[1] Only the gBS Schema is described in the DIA specification, together with the behaviour of a gBSDtoBin parser.

Listing 3.1: Example of a gBSD.

```
1  <gBSDUnit syntacticalLabel="bitStream" start="0">
     <gBSDUnit syntacticalLabel="header" start="0">
       <Parameter name="profile_idc" start="0" length="8">
         <Value xsi:type="b8">100</Value>
5      </Parameter>
     </gBSDUnit>
     <!-- ... -->
     <gBSDUnit syntacticalLabel="NalUnit" start="776" length="
         488" marker="TEMPORAL_LEVEL=1"/>
     <gBSDUnit syntacticalLabel="NalUnit" start="264" length="
         064" marker="TEMPORAL_LEVEL=2"/>
10   <!-- ... -->
   </gBSDUnit>
```

the gBS Schema elements, the gBSDtoBin process does not have to be aware of the underlying coding format. The two most important gBS Schema elements are *gBSDUnit* and *Parameter*. In Listing 3.1, an example of a gBSD corresponding to an SVC [143] bitstream is given, illustrating the use of *gBSDUnit*s and *Parameter*s.

- *gBSDUnit* represents a section of the bitstream. It can be used to point to a group of syntax elements or a block of data. It includes zero or more *Parameter*s and *gBSDUnit*s. Hence, a hierarchy of *gBSDUnit*s can be created. For instance, in Listing 3.1 (line 8), a *gBSDUnit* is used to represent a Network Abstraction Layer Unit (NALU), which is a syntax structure that is part of the SVC specification.

- *Parameter* is used to describe syntax elements of the bitstream that may change when adapting the bitstream. Unlike the *gBSDUnit*, the *Parameter* provides the actual value and datatype of the bitstream element. This enables the adjustment of the numerical values of bitstream syntax elements. On line 3 in Listing 3.1, a *Parameter* is used to represent the syntax element `profile_idc`, which has a length of 8 bits and a value of 100.

Markers can occur as attributes within both the *gBSDUnit* and the *Parameter* language constructs. These markers typically contain semantically meaningful information that can be used by the transformation process. In Listing 3.1 (lines 8 and 9), markers containing information regarding the temporal

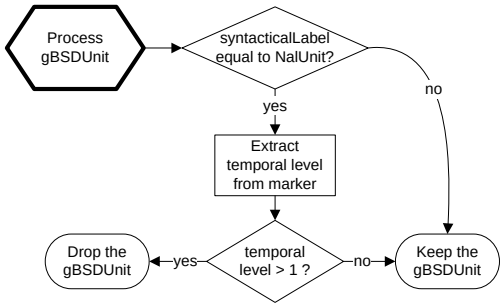


Figure 3.2: Workflow of a transformation filter which drops *gBSDUnit*s corresponding to a temporal level higher than 1.

level of a NALU are present in the gBSD. Figure 3.2 shows the workflow of a transformation filter that removes *gBSDUnit*s based on their markers. Such a transformation filter could be implemented by means of a STX stylesheet. In this example, applying the filter to the gBSD results in a transformed gBSD that corresponds to an SVC bitstream containing two temporal layers (i.e., the base and first layer).

3.2.2 gBS Schema in Practice

gBS Schema has already been used for a couple of use cases. For instance, European projects such as ISIS (Intelligent Scalability for Interoperable Services, [47]) and DANAE (Dynamic and distributed Adaptation of scalable multimedia coNtent in a context-Aware Environment, [101]) aimed at the design, implemention, and validation of a multimedia framework that allows to adapt audio-visual content to a wide range of service scenarios. Both projects used gBS Schema as adaptation technology. In [81], CAIN is introduced, which is a content adaptation manager targeted at the integration of different metadata-driven content adaptation approaches. gBS Schema is one of these approaches. Other examples of the use of gBS Schema are the adaptation and perceptual encryption of H.264/AVC Video [55], an event-driven video adaptation system [148], and the generic streaming of multimedia content [103].

3.2.3 Generation of gBSDs

As already mentioned in Section 3.2.1, the generation of gBSDs is not specified in the MPEG-21 DIA standard. Therefore, several approaches can be used to obtain a gBSD. In this section, we investigate the different methods to generate gBSDs. These are also summarized by Figure 3.3.

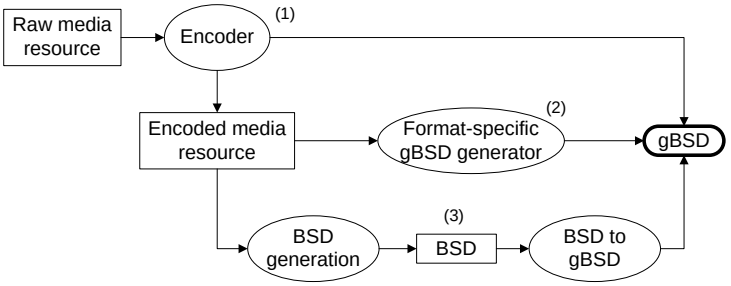


Figure 3.3: Techniques for generating gBSDs.

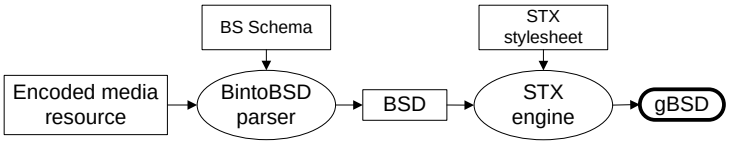


Figure 3.4: The two-step approach using BSDL's BintoBSD parser and STX.

3.2.3.1 Using Dedicated Software

Dedicated software can be developed that is able to automatically generate gBSDs. For instance, a specific encoder can be extended such that it is possible to generate a gBSD during the encoding process (option (1) in Figure 3.3). All the necessary information is available during the encoding of the media resource in order to produce a gBSD. However, the creation of a gBSD is application-specific. This implies that for different applications[2], different extensions have to be built on top of an encoder. Also, the logic is format-specific, requiring the development of new software when a new coding format has to be supported. Finally, this method is not applicable to already encoded media resources.

A second category within the dedicated software solutions consists of format-specific parsers that are able to create a gBSD, given an encoded media resource compliant with a specific coding format (option (2) in Figure 3.3). Note that this approach is also application- and format-specific, similar to the use of an encoder extended with gBSD generation functionality. As such, a new parser has to be developed in order to support a new coding format.

[2]Applications differ for example in terms of adaptation operations such as the exploitation of temporal scalability and the removal of violent scenes.

Listing 3.2: Excerpt of a resulting gBSD for an SVC encoded bitstream.

```
<gBSDUnit syntacticalLabel="Bit_stream" start="0">
  <!-- ... -->
  <gBSDUnit syntacticalLabel="byte_stream_nal_unit" start="
      53" length="1188"/>
  <!-- ... -->
</gBSDUnit>
```

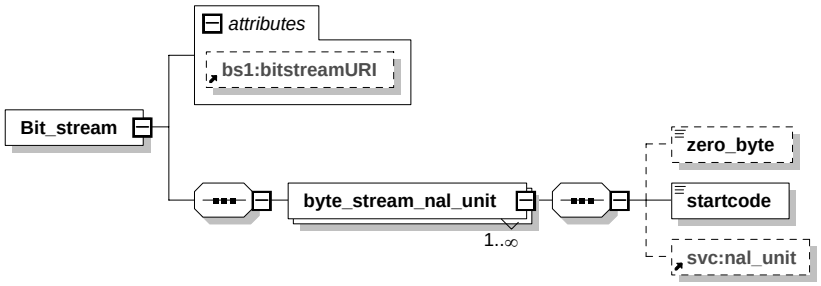


Figure 3.5: Simplified diagram of a BS Schema for SVC. An excerpt of the XML representation of this BS Schema can be found in Annex B, Listing B.1.

3.2.3.2 Using a Format-agnostic Approach

Next to the creation of dedicated software, it is also possible to generate gBSDs in a format-agnostic way. However, the production of gBSDs is application-specific due to the hierarchical structure of the description and the marking of *gBSDUnit*s and *Parameter*s for various adaptations. Therefore, a two-step approach (option (3) in Figure 3.3) was introduced by Panis *et al.* in [94]. First, a format-specific BSD is generated for a media resource. This can for example be achieved using BSDL's BintoBSD parser. The second step consists of transforming the format-specific BSD into a gBSD. XSLT or STX can be used as transformation technology, since both enable the use of a format-agnostic transformation engine. Figure 3.4 shows an example of the format-agnostic approach using BSDL's BintoBSD parser for the generation of BSDs and STX for the transformation of the BSDs into gBSDs.

To provide the reader with an idea regarding the implementation effort required by the approach proposed in Figure 3.4, the generation of gBSDs for SVC encoded bitstreams is discussed in more detail. An example of such a resulting gBSD is shown in Listing 3.2.

Listing 3.3: Excerpt of a BSD for an SVC encoded bitstream.

```
<Bit_stream bs1:bitstreamURI="example.264">
  <!-- ... -->
  <byte_stream_nal_unit>
    <zero_byte>0</zero_byte>
    <startcode>000001</startcode>
    <nal_unit>
      <!-- and so on -->
    </nal_unit>
  </byte_stream_nal_unit>
  <!-- ... -->
</Bit_stream>
```

The first step is to manually develop a BS Schema (a simplified diagram of such a BS Schema is shown in Figure 3.5). A more detailed explanation of the construction of a BS Schema for SVC can be found in [34]. Next, a BSD can be automatically generated using BSDL's standardized BintoBSD parser, taking as input the BS Schema for SVC. An example of such a resulting BSD is provided in Listing 3.3.

Next to a BS Schema, an XML transformation filter needs to be manually written in order to transform the BSD into a gBSD. The workflow of such an XML transformation filter is shown in Figure 3.6. The length of each syntax element needs to be determined in order to know the total length of a higher level structure (e.g., a NALU). The length is needed to create a corresponding *gBSDUnit* element which has a *start* and *length* attribute (line 3 in Listing 3.2).

Note that the author of the XML transformation filter determines the granularity of the resulting gBSD. In other words, the hierarchical structure of a gBSD is established during the transformation. In this example, the gBSD contains details up to the level of a NALU. Every NALU corresponds to a *gBSDUnit* (line 3 in Listing 3.2).

3.3 gBFlavor

In this section, gBFlavor [127, 128] is introduced. It is a novel solution for the creation of gBSDs in a format-agnostic way. In the next subsections, the motivation to develop gBFlavor and the architectural design of the gBFlavor adaptation chain are discussed. An in-depth specification of gBFlavor together with an elaboration on the transformation process is provided as well.

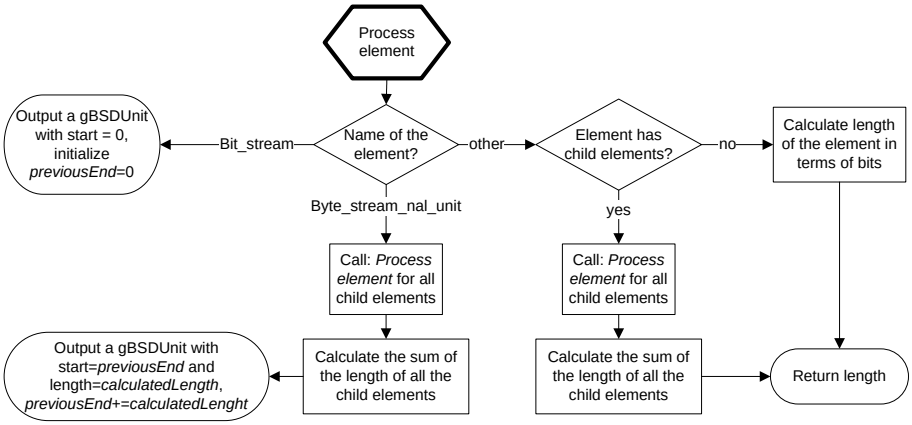


Figure 3.6: Workflow of an XML transformation filter which transforms BSDs for SVC into gBSDs. An implementation of this filter by means of a STX stylesheet can be found in Listing B.2.

3.3.1 Motivation

The automatic generation of gBSDs is a well-known problem. As indicated by Panis *et al.* in [94], the generation of a gBSD is application-specific. Thomas-Kerr *et al.* stated in [117] that "gBS Schema is generic (format-independent), but the generation process is not since this requires specific software which is able to parse the format in question". Both this application-dependent and format-dependent aspect of the generation process for gBS Schema makes it difficult to create a so-called 'BintogBSD parser', i.e., a format-agnostic parser that produces application-specific gBSDs.

Although the two-step approach discussed in Section 3.2.3.2 is format-agnostic, it has a number of disadvantages. First, two different technologies/languages (i.e., BSD generation and BSD-to-gBSD transformation) have to be used. For example, both BSDL and STX need to be used to obtain a gBSD. Second, the format-specific BSDs contain significantly more detail than the resulting gBSD. For instance, in case of H.264/AVC, adding markers to *gBSDUnit*s corresponding with a NALU based on their NALU type implies that the BSD has to contain the syntax element `nal_unit_type`. This is because the BSD-to-gBSD transformation needs access to this syntax element in order to set proper markers. However, there is no need to include these details in the gBSD because the transformation (based on the usage environment description) can make use of the markers. Hence, often more detail is needed in the BSD than in the gBSD, implying a decrease in execution speed of the

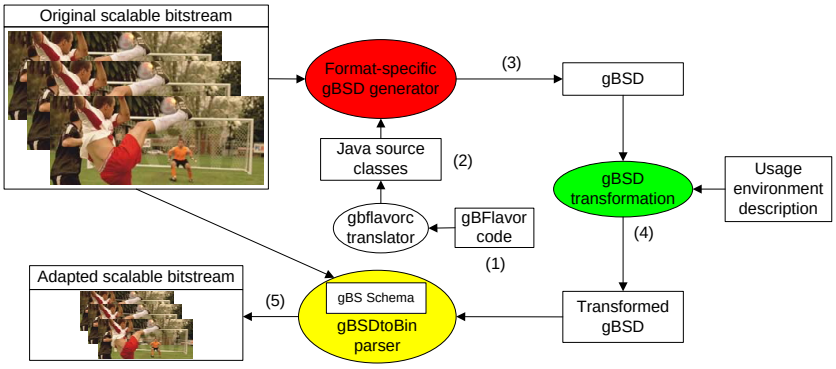


Figure 3.7: Adaptation framework based on gBFlavor.

format-specific BSD generation process (this will also be discussed in Section 3.4, where performance results are given). However, this amount of detail is needed for transforming the BSD into the resulting gBSD.

In order to offer a better solution for the creation of gBSDs in terms of execution time and implementation effort, we propose gBFlavor. It is a technology that enables the automatic generation of a format-specific parser. Subsequently, this parser is able to produce an application-specific gBSD for a given bitstream. Hence, gBFlavor provides an efficient format-agnostic solution for the generation of application-specific gBSDs.

3.3.2 Overall Functioning of gBFlavor

gBFlavor is built on top of BFlavor (BSDL + XFlavor, [126]). As discussed in Chapter 2, BFlavor is a description tool for the efficient creation of format-specific BSDs that are compliant with BSDL, using a modified version of XFlavor. Consequently, gBFlavor relies on the same principles as BFlavor, i.e., the automatic generation of a parser that is able to produce an XML description of the high-level structure of a given bitstream. The general workflow of a gBFlavor-enabled adaptation framework is illustrated in Figure 3.7. The following steps are needed to adapt a media resource using gBFlavor.

(1) The first step is the creation of a gBFlavor code. This code describes the high-level syntax or structure of a (scalable) coding format in an object-oriented manner (as discussed in the next section).

(2) The gbflavorc translator uses the gBFlavor code to automatically generate Java source classes, which are subsequently compiled to a format-

specific gBSD generator.

(3) The automatically generated format-specific gBSD generator takes as input a media resource and generates a gBSD. This gBSD is conformant to the MPEG-21 gBS Schema.

(4) According to the capabilities of a given usage environment, the gBSD is transformed by using an XML transformation tool such as XSLT or STX.

(5) The gBSDtoBin parser, the behavior of which is standardized in MPEG-21 DIA, takes as input the transformed gBSD and the original media resource, producing an adapted version of the original bitstream. The resulting bitstream is now suited for playback in a constrained usage environment.

Following the gBFlavor approach, the two-step operation (discussed in Section 3.2.3.2) is avoided for generating gBSDs, since the gBSD is directly generated by a format-specific parser. Note that, based on the different approaches to obtain gBSDs (discussed in Section 3.2.3), gBFlavor is located in the second option (i.e., using a format-specific gBSD generator). However, despite the fact that gBFlavor produces a format-specific parser, it is still a format-agnostic solution since the gbflavorc translator operates in a format-agnostic way. Every format-specific parser is thus generated in an automatic and format-independent manner.

3.3.3 gBFlavor versus BFlavor

As already mentioned, gBFlavor is built on top of BFlavor. Both serve as BSD generation tools, i.e., they provide a description tool for the automatic creation of efficient format-specific BSD generators. However, a number of key differences can be identified. BFlavor was introduced as a harmonized solution combining the strenghts of BSDL and XFlavor [31]. It serves as an efficient alternative for the format-agnostic generation of format-specific BSDs, i.e., it is an efficient replacement for BSDL's BintoBSD parser. In other words, BFlavor enables the creation of format-specific BSD generators compliant with BSDL. On the contrary, gBFlavor provides a solution for the efficient, automatic generation of gBSDs (i.e., format-agnostic BSDs compliant with MPEG-21 gBS Schema) by enabling the creation of format-specific gBSD generators. Note that, in contrast to BSDL's BintoBSD parser, gBS Schema does not offer a so-called 'BintogBSD parser'. In practice, BFlavor differs from gBFlavor in the following points.

- *Specification*: as will be discussed in Section 3.3.4, the gBFlavor specification is an extension of the BFlavor specification. This extension, i.e., the application-specific code part, is needed to enable the creation of application-specific gBSDs in a format-agnostic way (see Section 3.3.4.2). Note that the application-specific code part is not available in BFlavor because BSDL does not offer support for markers and hierarchical adaptations. Using BSDL, application-specific aspects can be exploited during the XML transformation.
- *Translation process*: although the gBFlavor specification is an extension of the BFlavor specification, their translation processes (i.e., the bflavorc and gbflavorc translators) are completely different. The reason is that BFlavor is compliant with BSDL and hence the translation process maps the constructs of the BFlavor code to constructs occurring in BSDL. On the other hand, gBFlavor is compliant with MPEG-21 gBS Schema which results in a translation process mapping constructs of gBFlavor to constructs occurring in MPEG-21 gBS Schema. Furthermore, additional mapping rules are necessary to provide a translation from the application-specific code to the corresponding gBS Schema hierarchy and marker occurrence. More information regarding the translation process of gBFlavor will be provided in Section 3.3.5.
- *Format-specific BSD generator*: the resulting BSD generators of BFlavor are able to generate BSDs compliant with BSDL, while the resulting BSD generators of gBFlavor are able to generate gBSDs compliant with MPEG-21 gBS Schema.

3.3.4 gBFlavor Specification

In this section, the syntax specification of gBFlavor is discussed in more detail. A gBFlavor code is divided into two sections: the high-level syntax code and the (optional) application-specific code. In Listing B.3, an excerpt of a gBFlavor code for SVC is provided [127]. The high-level syntax code is discussed in Section 3.3.4.1; the application-specific code is explained in Section 3.3.4.2.

The high-level syntax code corresponds to the BFlavor specification[3], which is in its turn derived from the XFlavor specification [51]. The latter has arisen from the Syntactic Description Language (SDL, [7]) which was developed to describe existing (and future) audiovisual coding standards. SDL

[3]This means that gBFlavor is backwards compatible with BFlavor. More specifically, based on a gBFlavor code, the gbflavorc translator is also able to generate a BS Schema (compliant to BSDL) and to produce a coding-format specific parser which is able to generate BSDs compliant to that BS Schema.

Listing 3.4: Example of a simple gBFlavor class.

```
%targetns{example%targetns}
%ns{ex%ns}
%root{Example%root}
%emulationBytes{(0002, 00);%emulationBytes}
%emulateBytes{(0001, 000201);%emulateBytes}

class Example (int param) {
  fixedByteRange(2) startcode = 0x0001;
  bit(2) parsable_var;
  if(param == 0){
    align();
    varByteRange() payload = 0x0001;
  }
}
```

addresses the need to disengage the definition of the bitstream syntax of audio-visual content from the decoding and rendering tools. More specifically, the description of the syntax should indicate *what* the bitstream contains, and not *how* to obtain it.

SDL is based on a set of well-defined elements with unambiguous semantics. Hence, the bitstream syntax definition features of SDL are described in the form of formal grammar rules. These rules form a context-free grammar and more specifically a LALR(1) (Look-Ahead Left to right, Rightmost derivation) grammar [2]. Therefore, compiler-compilers such as Yacc (Yet another compiler compiler, [69]) can be used to generate a parser for SDL, (X)Flavor or (g)BFlavor. In the next sections, the Extended Backus-Naur Form (EBNF, [57]) is used to describe the syntax constructs of gBFlavor in a formal way.

3.3.4.1 High-level Syntax Code

The high-level syntax code denotes how the generated parser has to parse the media resource. Hence, this high-level syntax code is based on the specification of the coding format used.

Classes

Syntax 1*:*
high_level_syntax_code = {hint_code | (**class**, class_name, ['(', parameter_list,

’)’,] ’{’, {statement}, ’}’) };

With the gBFlavor specification, it is possible to describe the (high-level) syntax of a coding format. Since this code is organized in a Java-like manner, the `class` structure is the building block of the gBFlavor code. Inside a class, the bitstream representation information is placed together with the data declarations. The high-level syntax code of gBFlavor is a declarative language, which implies that no user-defined methods are allowed in the gBFlavor code. However, in order to pass external information to a class, class parameters are allowed. This is illustrated on line 7 of Listing 3.4, where the class `Example` contains the parameter `param`.

Control Flow

Syntax 2*:*
if_else_expression = **if**, ’(’, condition, ’)’, ’{’, {statement}, ’}’, [**else**, ’{’, {statement}, ’}’];
while_expression = **while**, ’(’, condition, ’)’, ’{’, {statement}, ’}’;
for_expression = **for**, ’(’, [initialisation,] ’;’, [condition,] ’;’, [update,] ’)’, ’{’, {statement}, ’}’;
switch_case_expression = **switch**, ’(’, expression, ’)’, ’{’, {**case**, constant-expression, ’:’, {statement}, ’;’}, [**default**, ’:’, {statement}, ’;’,] ’}’;

It is possible to add control flow into classes by using `if-else`, `switch-case`, `for`, and `while` constructs. An example of the use of an `if`-statement is illustrated on line 10 of Listing 3.4.

Variables

Syntax 3*:*
variable_declaration = type, [’(’, length, ’)’,] element_name, [’=’, value,] ’;’ ;

A class may contain two different kinds of variables: parsable and non-parsable variables. A non-parsable variable can be seen as a regular variable used in Java. Parsable variables, illustrated on line 9 of Listing 3.4, include a parse length immediately specified after their type declaration. Parsable variables define the bitstream syntax elements because their value is fetched from the bitstream itself, while non-parsable variables are typically used for internal computations.

Variables can have built-in datatypes or user-defined datatypes. The following datatypes are supported by gBFlavor.

Table 3.1: Exponential Golomb bit strings and codewords.

bit string	ue()	se()
1	0	0
010	1	1
011	2	-1
00100	3	2
00101	4	-2
00110	5	3
00111	6	-3
...	...	...

- *bit*: the bit datatype is used to describe syntax elements having a fixed bit length.

- *short, int, long*: adopted from procedural languages such as Java, gBFlavor supports the signed and unsigned short (2 bytes), int (4 bytes), and long (8 bytes) datatypes.

- *ue, se*: signed and unsigned exponential Golomb datatypes, as used in H.264/AVC, are supported by gBFlavor. Note that these datatypes have a variable bit length, implying that no parse length can be specified. In Table 3.1, an overview is given of the first bit strings and corresponding codewords for the signed and unsigned exponential Golomb codes. Their use in SVC is illustrated on line 14 of Listing B.3.

- *varByteRange*: the varByteRange datatype is used for referring to a particular bitstream segment. For instance, this datatype is used when it is not relevant to describe a part of the bitstream syntax in the resulting gBSD. Only the start and the length of the bitstream segment are written into the gBSD in order to correctly reconstruct the bitstream. The end of the bitstream segment is indicated by a particular byte sequence (or range of byte sequences). This is illustrated on line 12 of Listing 3.4, where the element `payload` ends before the next occurrence of the bit string 0x0001. Note that such a bit string usually corresponds to a start code (e.g., the start code for a NALU in SVC corresponds to the byte sequence 0x000001).

- *fixedByteRange*: similar to the varByteRange datatype, the fixedByteRange datatype is used for referring to a particular bitstream segment. However, this datatype is used for segments of a fixed length in terms of bytes. The use of this datatype is illustrated on line 8 of List-

ing 3.4, where the element `startcode` contains a fixed length of 2 bytes.

- *Encoded base class*: in order to provide an extension mechanism for the creation of new datatypes, gBFlavor contains a built-in class, *Encoded*. This class provides a well-defined interface for user-defined datatypes. Hence, user-defined datatypes in gBFlavor are classes that extend the *Encoded* base class. More detailed information about this extension mechanism can be found in [31].

During gBSD generation, it is possible to check whether the values of particular syntax elements match with predefined values in the gBFlavor code. As illustrated on line 8 of Listing 3.4, this is done by simply assigning the predefined value to the parsable variable that needs to be checked. A warning is given during the gBSD generation when a validation check fails for a particular syntax element. Note that this feature is typically used to validate the value of syntax elements containing fixed values (e.g., start codes).

Built-in Functions

Syntax 4*:*
builtin_function = function_name, '(', [function_parameter_list,] ')', ';' ;

gBFlavor provides five built-in functions that can assist in describing the high-level syntax of a coding format.

- *align()*: this function reads bits from the bitstream until a byte-aligned position is reached. The value of the padding bits is added to the resulting gBSD. The use of this function is illustrated on line 11 of Listing 3.4.

- *getcontext(class, index, variable)*: classes can carry information that is needed for the parsing of future syntax elements or for future processing steps. Such information is called context information. gBFlavor offers two possibilities for dealing with context information: information can be passed through class parameters or by using context classes. A context class is a class that is kept in memory during the parsing process. A possible use case is for instance the Sequence Parameter Set (SPS) in SVC. How to declare a context class in gBFlavor is discussed further in this section. The *getcontext()* function is able to retrieve information from a class that is kept in the context (i.e., stored in the system memory). Its use is illustrated on line 3 of Listing 3.5, where the syntax element `element1` is fetched from the class `ContextClass`. The first parameter of the function is the name of a class that was previously

Listing 3.5: Using context classes in gBFlavor.

```
class RegularClass {
  bit(4) context_index;
  int var = getcontext("ContextClass", context_index,
      $element1);
  if(var > 0)
    // ...
}

%context{index%context}
class ContextClass{
  bit(4) index;
  bit(6) element1;
  //...
}
```

stored in the context. The second parameter denotes the value of the index element of the context class. Different instances of a context class can have different values for their index element (they are thus stored separately in the context). For instance, this is the case when different SPSs are present in an SVC bitstream. Note that this is discussed further in this section. The last parameter indicates the variable that must be accessed inside the selected context class. Its value will be returned by the *getcontext()* function [131].

- *nextbits(n)*: a look-ahead mechanism is provided by this function. It takes as parameter the number of bits to look forward into the bitstream. The value of these bits is returned by this function.
- *numbits()*: this function returns the number of bits that have already been parsed since the start of the parsing process.
- *skipbits(n)*: bit skipping is possible by using this function. It takes as parameter the number of bits that have to be skipped. Note that, in contrast to the use of the byteRange datatypes, nothing is written into the resulting gBSD when the *skipbits()* function is applied.

Hint codes

Syntax 5*:*
hint_code = '%', hint_code_type, '{', hint_content, '%', hint_code_type, '}';

Hint codes are used to signal additional information to the automatically generated parser. Note that we explicitly introduce a new term to indicate that the information provided by the code is used to provide hints for the gbflavorc translator, which is in contrast to the so-called verbatim codes of XFlavor. Six hint codes are defined within the gBFlavor specification.

- *ns, targetns*: these two hint codes are used to signal information about the classification scheme that has to be used in the resulting gBSD. The hint codes have to occur at the beginning of the gBFlavor code (shown on line 1 and 2 in Listing 3.4).

- *root*: the root hint code is used to indicate the root class of the gBFlavor code. The generated parser has to start with the parsing of this class when it receives a bitstream as input. In the example in Listing 3.4, the root class is `Example` (line 3). Note that this hint code is required to be present at the start of the gBFlavor code.

- *context*: the context hint code can appear at the start of a class declaration and will mark this class as a context class (discussed earlier in this section). Instances of this class will be kept in memory (context) during the parsing process. The hint code takes as parameter the name of the index element of the context class. Figure 3.8 illustrates the processing of context classes in gBFlavor. In this example, the SPS class is a context class with index element `seq_parameter_set_id`. Each context class has its own array in system memory. Instances of context classes are stored in this array based on the value of their index element. Storing a context class implies that every syntax element of the class, together with its value, is stored in the array. The use of this hint code is shown on line 8 in Listing 3.5. 0 is used as index element when no index element exists within the class (implying that only the last occurrence of this class is kept in memory). As already discussed in this section, the built-in *getcontext()* function can be used to retrieve values stored in the context classes.

- *emulationBytes* and *emulateBytes*: gBFlavor provides support for emulation prevention bytes by introducing the emulationBytes and emulateBytes hint codes. When a start code occurs coincidentally in the bitstream (i.e., false start code generation), emulation prevention bytes are inserted in order to prevent a start code emulation. Recent coding formats such as H.264/AVC and Video Codec-1 (VC-1, [113]) make use of emulation prevention bytes. The emulationBytes hint code is used

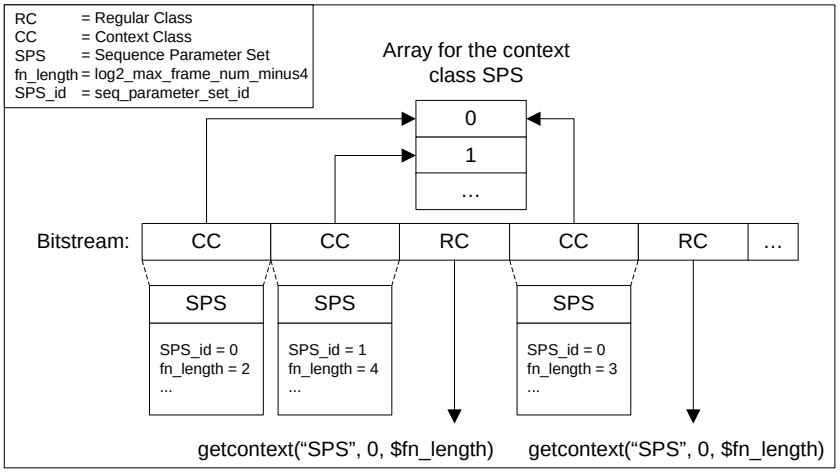


Figure 3.8: Processing context classes in gBFlavor. The SPS is used as context class. A regular class is a class which is not a context class.

by gBFlavor to detect and to ignore the emulation prevention bytes during the bitstream parsing. A list of couples *(byte sequence with emulation prevention byte, byte sequence without emulation prevention byte)* is given as an argument to this hint code (illustrated on line 4 in Listing 3.4).

The emulateBytes hint code is used to signal the occurrence of emulation prevention bytes in the resulting gBSD. This is needed because the gBSDtoBin parser needs to know if there are emulation prevention bytes or not. The argument of this hint code is a list of couples *(byte sequence that cannot appear in the bitstream, emulated version of the byte sequence)* (illustrated on line 5 in Listing 3.4) [131]. Note that the hint codes dealing with emulation prevention bytes and context information are optional, in contrast to the three hint codes discussed earlier.

3.3.4.2 Application-specific Code

Syntax 6*:*
app_spec_code = **gBSDApp**, application_name, '{', class_declaration, '}';

Thus far, we have only discussed the specification of the high-level syntax code. With this specification, it is already possible to automatically generate a format-specific parser that is able to generate gBSDs. These gBSDs contain the same level of detail as described in the high-level syntax code. Indeed,

Listing 3.6: Example of the application-specific part of a gBFlavor code.

```
1  gBSDApp Example_Application {
     class Example {
       if(parsable_var > 0)
         setmarker("Example", "", "Marked!");
5    }
   }
```

every parsed syntax element described in the high-level syntax code will also occur in the resulting gBSD. Given the high-level syntax code for SVC in Listing B.3, a parser can be generated that takes as input an SVC bitstream and produces the gBSD given in Listing B.4. This gBSD contains the same level of detail as described in the high-level syntax code, i.e., up to and including the slice header.

As discussed in Section 3.2, MPEG-21 gBS Schema provides support for hierarchical adaptations by describing hierarchies of *gBSDUnit*s. It also enables semantically meaningful marking of syntactical elements. However, the high-level syntax code of gBFlavor is not able to exploit these two features of gBS Schema. Indeed, it is not possible to produce different hierarchies of *gBSDUnit*s (i.e., the level of detail within gBSDs is equal to the level of detail within the gBFlavor code). Also, no possibility exists to add markers to the XML elements (i.e., *gBSDUnit*s and *Parameter*s). Note that these two features of gBS Schema make the resulting gBSDs application-specific.

In order to support different hierarchies and the insertion of markers in gBFlavor, application-specific code can be added to the gBFlavor code. A simple example of an application-specific code, belonging to the high-level syntax code of Listing 3.4, is given in Listing 3.6. Note that in Listing B.3, an example of application-specific code for SVC is included. In this example, markers are added to NALUs of the resulting gBSD in order to support the exploitation of scalability along the different axes of SVC. More detailed information regarding the functioning of sub-sequences and the exploitation of scalability in SVC can be found in [30] and [34] respectively.

Each target application within the application-specific code is denoted by using the keyword `gBSDApp`, followed by the name of the target application (illustrated on line 1 of Listing 3.6). Within the `gBSDApp` environment, a list of classes can be specified where a number of calculations have to occur in order to be able to set a marker at a specific syntax element (the formal syntax can be found in Syntax 6). Each class within a `gBSDApp` environment also has

to be present in the high-level syntax part of the gBFlavor code. The code of a particular class occurring in a `gBSDApp` environment is executed after having parsed the corresponding class in the high-level syntax part of the gBFlavor code (in case the application is selected by the parser as target application). Note that we have separated the application-specific code from the high-level syntax code because multiple target applications can be specified. The generated parser is in this case able to support multiple target applications.

Within each class inside the application-specific environment, a lightweight version of the high-level syntax code specification (Section 3.3.4.1) can be used. This lightweight version contains restrictions, due to the fact that it is not allowed to parse anything from the bitstream during the assignment of the markers. Hence, the use of the following constructs is prohibited in the application-specific code:

- parsable variables;
- the built-in functions *skipbits()* and *align()*;
- classes which extend the *Encoded* base class;
- hint codes.

The application-specific code also contains a new built-in function: *setmarker(class, variable, marker)*. This function makes it possible to assign a marker to a specific syntax element. Its use is illustrated on line 4 of Listing 3.6. This built-in function takes three parameters as input. The first parameter is the name of the class where the marker has to occur. The second parameter is the name of the variable where the marker has to be inserted. In case the second parameter is empty, the marker is inserted at the place where the class starts. In practice, this is considered as adding a marker to a *Parameter* (in case a variable is given in the second parameter) or to a *gBSDUnit*, which corresponds to the given class specified in the first parameter (in case the second parameter is empty). The third parameter is the actual value of the marker. In the SVC example (Listing B.3), a marker is assigned to a *gBSDUnit* that corresponds to the class `ByteStreamNalUnit`. This marker contains information regarding the temporal, spatial, and quality levels of the corresponding NALU.

3.3.5 Mapping between gBFlavor and MPEG-21 gBS Schema

Creating a gBFlavor code for a specific coding format enables the automatic creation of a parser for bitstreams compliant with that particular coding format.

Table 3.2: Mapping between the high-level syntax code of gBFlavor and MPEG-21 gBS Schema language constructs.

gBFlavor code		gBS Schema	
Element	Datatype	Element	Datatype
class		gBSDUnit	
	bit	Parameter	bx
	short	Parameter	xsd:short
	int	Parameter	xsd:int
	long	Parameter	xsd:long
Parsable	ue	Parameter	bs1:unsignedExpGolomb
variable	se	Parameter	bs1:signedExpGolomb
	varByteRange	gBSDUnit	
	fixedByteRange	gBSDUnit	
	Encoded	Parameter	bx
align()		Parameter	bs1:align8

The result of the parsing process is a gBSD. Hence, a mapping has to be established between the gBFlavor specification and the MPEG-21 gBS Schema specification.

3.3.5.1 Mapping of High-level Syntax Code to gBS Schema Constructs

An overview of the mapping of the high-level syntax code to gBS Schema is tabulated in Table 3.2. A class in the gBFlavor code is mapped to a *gBSDUnit* in the gBSD. Within a class, the parsable variables, which may have datatypes such as bit, short, int, long, ue, and se, are mapped to a *Parameter* containing the corresponding datatype of the gBS Schema. Note that a parsable variable defined by means of the *Encoded* base class (i.e., a user-defined datatype) also maps to a *Parameter*. However, since gBS Schema does not support an extension mechanism for user-defined datatypes, the length of the datatype becomes fixed under the form of a `bx`[4] datatype (the actual length will be known during the parsing process).

Furthermore, parsable variables of the type fixedByteRange and varByteRange are mapped to a *gBSDUnit*. This is straightforward since a byteRange datatype points to a bitstream segment, just like a *gBSDUnit* does. Finally, the *align()* built-in function is mapped to a *Parameter* having the datatype bs1:align8. Note that the bs1:align8, bs1:unsignedExpGolomb, and bs1:signedExpGolomb datatypes have been adopted in the second amendment of the MPEG-21 DIA specification [65].

[4] *bx* denotes the built-in integer datatypes of MPEG-21 gBS Schema. For instance, datatype *b2* denotes a length of two bits.

Algorithm 1 Marking of the full-detailed classes.

Require: T represents the full class tree, U contains all the user-marked classes: $(\forall u \in U)(setmarker(u, s_1, s_2)$ occurs in the gBFlavor code) with s_1 and s_2 string constants

Ensure: a list M containing the classes that must be described in full detail

```
repeat
  for all c ∈ T do
    if c ∉ M then
      for all d ∈ list of children of c do
        if (d ∈ M) ∨ (d ∈ U) then
          add c to M
        end if
      end for
    end if
  end for
until M has not changed
return M
```

3.3.5.2 Hierarchical Changes in an Application-specific gBSD

As discussed in Section 3.3.4.2, the application-specific part of the gBFlavor code enables support for the insertion of markers by using the *setmarker()* function. This function does not only lead to the addition of a marker to a syntax element, its use also results in a change in the hierarchical structure of the gBSD. More specifically, classes (and hence their corresponding *gBSDUnit*s) that do not include a marker, will not occur in the resulting gBSD. Listing 3.7 shows a gBSD after selecting the target application 'TemporalSpatialQuality_Scalability' (described in Listing B.3). As one can see, the gBSD only contains the corresponding *gBSDUnit*s of the classes `bitStream` and `byteStreamNalUnit`. Further, markers are added to *gBSDUnit*s that correspond to a `byteStreamNalUnit` class.

The logic for obtaining the list of classes whose corresponding *gBSDUnit*s have to occur in the resulting gBSD is given in Algorithm 1. It starts from the list U, which contains the list of classes that were marked by the author of the gBFlavor code (using the *setmarker()* function). The goal is to find the list M, which contains all the classes whose corresponding *gBSDUnit*s have to occur in the resulting gBSD. In each iteration, the classes containing a child class that occurs in U or in M, are added to M. The algorithm ends if M did not change anymore during an iteration.

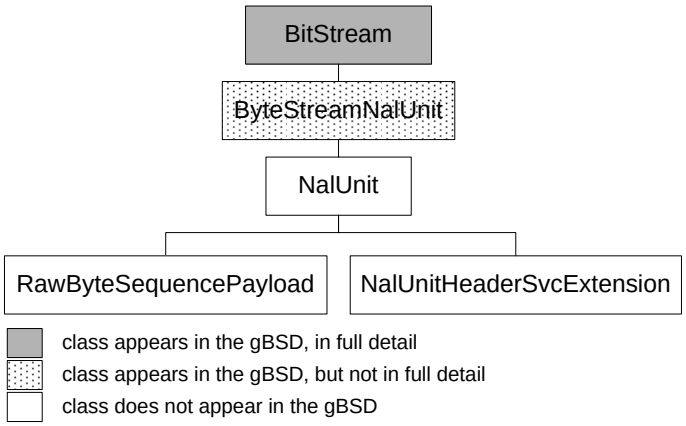


Figure 3.9: Part of the class tree, extracted from the high-level syntax code given in Listing B.3.

Listing 3.7: Excerpt of a gBSD, generated by gBFlavor and using the gBFlavor code given in Listing B.3 ('TemporalSpatialQuality_Scalability' is set as application).

```
<gBSDUnit syntacticalLabel=":jsvm:bitStream" start="0">
  <!-- ... -->
  <gBSDUnit syntacticalLabel=":jsvm:byteStreamNalUnit"
      start="53776"  length="112488" marker="TL=1:SL:0:QL=1"
      />
  <gBSDUnit syntacticalLabel=":jsvm:byteStreamNalUnit"
      start="166264" length="100432" marker="TL=1:SL:0:QL=2"
      />
  <gBSDUnit syntacticalLabel=":jsvm:byteStreamNalUnit"
      start="266696" length="92800"  marker="TL=1:SL:1:QL=0"
      />
  <gBSDUnit syntacticalLabel=":jsvm:byteStreamNalUnit"
      start="359496" length="105872" marker="TL=1:SL:1:QL=1"
      />
  <!-- ... -->
</gBSDUnit>
```

We will illustrate Algorithm 1 by applying it to the gBFlavor code given in Listing B.3. The corresponding gBSD is shown in Listing 3.7. A part of the class tree, extracted from the high-level syntax code, is depicted in Figure 3.9. All the classes within the class tree occur in the list T. The list U only contains the class `ByteStreamNalUnit`. The result of the algorithm in this example is the list $M = \{BitStream\}$, which contains the classes whose corresponding *gBSDUnit*s have to occur in the resulting gBSD (gray classes in Figure 3.9). The corresponding *gBSDUnit*s of classes that occur in list U but not in list M are present in the resulting gBSD, but not in full detail (dotted classes in Figure 3.9). Indeed, they are represented by a *gBSDUnit*, which does not have children (illustrated on line 3 of Listing 3.7). The *gBSDUnit*s corresponding to the remaining classes are not described in the gBSD (white classes in Figure 3.9).

It is important to remark that the classes whose corresponding *gBSDUnit*s do not occur in the resulting gBSD, are still parsed. These classes typically provide information to assign proper markers to syntax elements. For instance, the syntax element `nal_unit_type`, which is located in the class `NalUnit` (line 69 in Listing B.3), is used in the application-specific code to determine the value of a marker (line 97 in Listing B.3). However, since this syntax element is located in a class that does not occur in the list M or U, it is not present in the corresponding gBSD (Listing 3.7).

3.4 Performance Results

In order to evaluate the performance of our description-driven adaptation framework, we have tested gBFlavor in the context of two scalable coding formats, i.e., SVC and JPEG2000. An overview of the characteristics of the test sequences used is given in Table 3.3. Screenshots of these test sequences are provided in Figure 3.10. The target application of the gBSDs within the adaptation framework is the exploitation of scalability along the different axes of the coding formats.

For the gBSD generation process, two different approaches are compared. The first approach is the generation of a format-specific BSD, followed by the transformation of this BSD into a gBSD (as discussed in Section 3.2.3.2). BSDL is used to generate format-specific BSDs, while STX is used to transform the BSDs into gBSDs. In the second approach, gBFlavor is used. Note that we have only included format-agnostic solutions in our experiments.

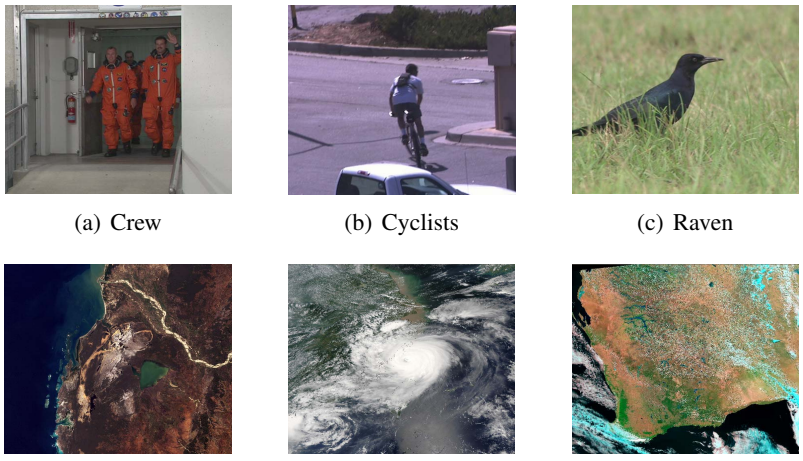
For the generation of the BSDs, we have relied on an optimized version of BSDL (as discussed in [34]). Joost (version 2006-10-05) is used as STX engine. In order to generate the adapted bitstreams from the (transformed) gB-

Table 3.3: Bitstream characteristics of the test sequences.

SVC									
Name	Resolution	# frames	Frame rate (Hz)	Size (MB)	Length (s)	Bit rate (MBit/s)	# temporal layers	# spatial layers	# quality layers
Crew	1280x720	1200	60	18.7	20	7.5	4	3	3
Cyclists	1280x720	1200	60	10.0	20	4.0	4	3	3
Raven	1280x720	1200	60	12.4	20	5.0	4	3	3

JPEG2000						
Name	Resolution	Size (MB)	Bit rate (bit/pixel)	# colour components	# spatial layers	# quality layers
Image1	4000x4000	5.4	2.8	3	10	23
Image2	5600x7200	10.8	2.3	3	10	23
Image3	6476x6224	19.6	4.1	3	10	23

SDs, we have used DIA's reference software (version 3.0.3 of the gBSDtoBin parser). In order to provide an estimate of the overhead of the XML descriptions in terms of file size, the (g)BSDs were compressed using XWRT [149], which is a high-performance XML compressor. Performance measurements were done on a PC having an Intel Pentium D 2,8 GHz CPU and 1 GB of system memory at its disposal. The operating system used was Windows XP Pro SP2, running Java 2 Runtime Environment (SE version 5). Every step was executed five times, whereupon the average was calculated (the average standard deviation for the different adaptation chains was 0.06 s).



(a) Crew (b) Cyclists (c) Raven

(d) Image1 (e) Image2 (f) Image3

Figure 3.10: Screenshots of the test sequences.

Table 3.4: Execution times and file sizes for the gBSD generation process (SVC).

BSDL + STX								
Name	BintoBSD (s)	BSD Size (MB)	compr. BSD Size (KB)	STX (s)	gBSD Size (KB)	compr. gBSD Size (KB)	Speed (MBit/s)	Total time (s)
Crew	49.2	13.0	70.6	8.7	1379.0	63.6	2.6	57.9
Cyclists	44.2	13.0	67.4	8.6	1371.3	58.6	1.5	52.8
Raven	45.4	13.0	68.2	8.7	1372.3	59.4	1.8	54.0

gBFlavor					
Name	gBSD generator[a] (s)	gBSD Size (KB)	compr. gBSD Size (KB)	Speed (MBit/s)	Total time (s)
Crew	4.6	1391.9	63.4	32.1	4.6
Cyclists	3.4	1384.2	58.7	23.6	3.4
Raven	3.7	1385.1	59.2	26.8	3.7

[a] 'gBSD generator' denotes the parser that is generated with the gbflavorc translator.

Table 3.5: Execution times and file sizes for the gBSD generation process (JPEG2000).

BSDL + STX							
Name	BintoBSD (s)	BSD Size (KB)	compr. BSD Size (KB)	STX (s)	gBSD Size (KB)	compr. gBSD Size (KB)	Total time (s)
Image1	3.3	174.2	8.4	1.5	554.4	17.9	4.8
Image2	5.3	174.5	8.5	1.6	555.4	17.8	6.8
Image3	8.7	174.9	8.8	1.6	556.6	18.2	10.2

gBFlavor				
Name	gBSD generator[a] (s)	gBSD Size (KB)	compr. gBSD Size (KB)	Total time (s)
Image1	1.5	624.1	18.3	1.5
Image2	2.3	625.0	18.2	2.3
Image3	3.5	626.3	18.5	3.5

[a] 'gBSD generator' denotes the parser that is generated with the gbflavorc translator.

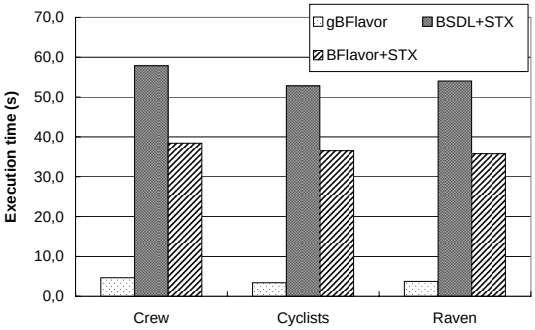


Figure 3.11: Execution times for gBSD generation (SVC).

3.4.1 gBSD Generation

3.4.1.1 General Observations

The execution times of the gBSD generation process for SVC and JPEG2000 are tabulated in Table 3.4 and Table 3.5, respectively. In these tables, gBFlavor and the combination of BSDL and STX are compared. Note that we did not include the time spent by the gbflavorc translator and the compilation of the java source classes to a format-specific parser. This is because these steps can be seen as pre-processing steps since both only need to be done once for each coding format. Therefore, the time spent on these first two steps can be ignored (and is negligibly small in practice). As one can see, gBFlavor outperforms the combination of BSDL and STX in terms of execution speed. The combination of BSDL and STX performs bad in comparison with gBFlavor due to the following reasons.

- **Performance of BSDL's BintoBSD parser.** The generation of a format-specific BSD takes 80% of the total time for generating an application-specific gBSD using the two-step approach. There are several reasons for the high execution times of the BintoBSD parser.
  - *XPath evaluation*: the BSDL specification offers XPath [22] as a mechanism to obtain context information (i.e., information of already parsed bits). On the contrary, gBFlavor uses context classes and class parameters to obtain context information. The performance of the context mechanism used by (g)BFlavor has proven to be superior to the evaluation of XPath expressions [131].
  - *Granularity of the format-specific BSD*: as already mentioned in Section 3.3.1, the format-specific BSD contains significantly more

detail than the resulting gBSD. This is because the BSD-to-gBSD transformation needs access to certain syntax elements in order to set proper markers. However, there is no need to include these details in the resulting gBSD because the transformation can make use of the markers. Hence, the format-specific BSDs will typically be larger in terms of file size than the resulting gBSDs because they have a finer granularity. This is illustrated in Table 3.4, where the file sizes of the BSDs and gBSDs are given for the different SVC test sequences. A BSD for the Crew sequence has a size of 13 MB, while the corresponding gBSD takes only 1.3 MB. Larger BSD sizes imply more I/O operations in order to write the BSDs to disk, introducing a decrease of execution time for the BintoBSD parser. Note that the granularity of BSDs and gBSDs does not always differ. This is in particular true for the JPEG2000 coding format, where the BSDs and corresponding gBSDs contain the same level of detail because it should be possible to modify syntax elements deep in the syntax structure (e.g., the modification of the resolution).

  - *Generic software module*: the BintoBSD parser is a generic parser which can take any bitstream as input, regardless of the coding format. This implies that format-specific information (i.e., the BS Schema) needs to be interpreted. Hence, genericity comes at a cost of execution time for loading the BS Schema in order to interpret it (approximately 1 s).

- **Performance of the BSD-to-gBSD transformation.** The transformation of a format-specific BSD to an application-specific gBSD is implemented by means of a STX stylesheet. Note that this transformation process was discussed in Section 3.2.3.2 for the SVC coding format. Despite the BintoBSD parser being the bottleneck in the two-step approach for generating gBSDs, the BSD-to-gBSD transformation also introduces an increase in execution time due to the following reasons.

  - *Granularity of the incoming format-specifc BSD*: as mentioned above, the granularity of the format-specific BSD is mostly finer than the granularity of the resulting gBSD. Hence, the STX processor needs to read and parse a fine-detailed BSD, resulting in a lot of I/O operations.
  - *Length calculations*: during the BSD-to-gBSD transformation, the length of the different *gBSDUnit*s needs to be calculated (as discussed in Section 3.2.3.2). This means that every detail of the

format-specific BSD needs to be catched and processed in order to determine the exact length in terms of bits of a *gBSDUnit*. On the contrary, gBFlavor has all the information regarding the length of a specific *gBSDUnit* at its disposal because the application-specific calculations are performed during the parsing of the bitstream.

– *Generic software module*: analogous to the BintoBSD parser, the STX processor is also a generic software module which has to interpret a STX stylesheet. Loading the STX stylesheet during the BSD-to-gBSD transformation takes 0.7 s on average.

Generation of application-specific gBSDs can be done very fast using gBFlavor thanks to the following reasons.

- *No XPath evaluation*: dealing with context information in gBFlavor is possible by using context classes or class parameters. As discussed in Section 3.3.4.1, context classes use arrays as underlying implementation to fetch context information. This is in contrast to BSDL where XPath evaluation is used to obtain context information.

- *Combination of parsing and application-specific gBSD generation*: where the combination of BSDL and STX introduces a lot of overhead in terms of I/O operations due to the two-step approach, gBFlavor does not suffer from this. Since gBFlavor allows to create a format-specific and application-specific parser at the same time, an application-specific gBSD can be obtained in one step. This is in particular illustrated in Figure 3.11, where the execution times for the gBSD generation (using the SVC format) are also illustrated for the combination of BFlavor and STX. BFlavor is an optimized solution for BSDL's BintoBSD parser; however, the execution speed of BFlavor followed by STX is still lower than gBFlavor's execution speed due to the need for more I/O operations (generation of an intermediate BSD).

- *Granularity of the resulting gBSD*: as already discussed above, the granularity of the resulting gBSD is often more coarse than its corresponding format-specific BSD. This results in a limited amount of I/O operations to write the resulting gBSD.

- *Format-specific software module*: since gBFlavor enables the creation of a format-specific parser in a format-agnostic way, there is no need to interpret format-specific information (like the BintoBSD parser) or application-specific information (like the STX processor).

Note that we have not compared gBFlavor with XFlavor due to the following reasons. First of all, XFlavor is not able to generate gBSDs. XFlavor is a description tool able to create an efficient format-specific parser. Such a parser creates for a given bitstream an XML description, compliant with a W3C XML Schema. However, this XML description can only be used by XFlavor's non-standardized bitgen tool, which takes as input the XML description and produces an adapted bitstream. A two-step approach, analogous to the BSDL+STX approach, could be used in order to solve this problem. In this case, an XML transformation filter (e.g., a STX stylesheet) needs to be written which is able to transform the XML description generated by the format-specific parser into a gBSD with a specific granularity and with the presence of particular markers. However, there is an important disadvantage using XFlavor for the generation of XML descriptions, making the two-step approach not feasible in practice. This is explicitly stated in [31]: "In XFlavor, the complete bitstream data are actually embedded in the BSD, resulting in potentially verbose descriptions, while BSDL uses a specific datatype to point to a data range in the original bitstream when it is too verbose to be included in the BSD". Hence, XFlavor can be seen as a representation language, rather than a description language, because it describes the bitstream at a low-level, bit-per-bit basis. This implies that using XFlavor for the adaptation of scalable coding formats is not feasible in practice.

Table 3.4 and Table 3.5 also show the file sizes of the different BSDs and gBSDs. Both the two-step approach BSDL+STX and gBFlavor generate the same output. First, it is obvious that both approaches have a gBSD (i.e., a BSD compliant with MPEG-21 gBS Schema) as output. Second, the granularity of the gBSDs is the same. In other words, the amount of detail of the gBSDs is the same for the BSDL+STX and the gBFlavor approach. Note that the granularity is determined during the STX transformation in case of the BSDL+STX approach; for the gBFlavor approach, this is specified in the application-specific code (which is discussed in Section 3.3.4.2). Third, the same markers occur in the gBSDs of both approaches. These three points make that the gBSDs are totally equivalent for both the BSDL+STX and the gBFlavor approach. An example of such a gBSD can be found in Listing 3.7, where an excerpt is listed of a gBSD having SVC as underlying coding format. Note that the little differences between the gBSD sizes of both approaches are caused by a different use of spaces, tabs, or new lines. However, they are syntactically completely equal.

Comparing the sizes of the format-specific BSDs and the gBSDs for the SVC format, the gBSDs are smaller than their corresponding BSDs because the gBSDs describe less syntax elements, i.e., syntax elements up to the NALU

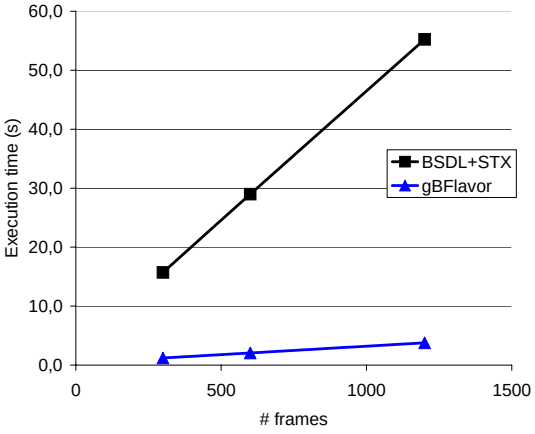


Figure 3.12: Impact of the # frames on the execution times for gBSD generation.

(see Listing 3.7). The corresponding BSDs contain a description up to and including the NALU header. Note that this detailed information is needed to set the proper markers in the resulting gBSD. For the JPEG2000 format, the gBSDs and corresponding BSDs contain the same level of detail because it should be possible to modify syntax elements deep in the syntax structure (e.g., the modification of the resolution). However, compressing the gBSDs results in a rather small overhead in comparison to the file size of the media resource (about 0.5 % – 5 %).

3.4.1.2 Impact Parameters for the gBSD Generation Process

In this section, a number of parameters are discussed that influence the performance of the gBSD generation process. These parameters apply to both the two-step approach and gBFlavor; however, a number of parameters will have a bigger impact on one of the approaches due to the different bottlenecks discussed in the previous section. The characteristics of the SVC test sequences used for this comparison are tabulated in Table 3.6. The execution times of the gBSD generation process for these test sequences can be found in Table 3.7. The following parameters are taken into account.

- *bit rate*: the number of I/O-operations is mostly determined by the bit rate of the bitstream (i.e., a higher bit rate results in a higher number of I/O-operations). The bit rate is dependent on other parameters such as the content of the sequence, the resolution, and the number of scalability axes.

- *content*: the content of a media resource has a big impact on its bit rate. Factors such as the amount of motion and the presence of complex textures, together with the coding algorithm used determine the bit rate of a media resource. Comparing the bit rate of the Cyclists sequence and the Crew1 sequence, one can see that the bit rate of Crew1 is much higher (see Table 3.6). This results in faster execution times for the Cyclists sequence (52.8 s in comparison with 57.9 s for Crew1 when using BSDL's BintoBSD parser and STX).

- *frame rate*: a higher frame rate results in a higher bit rate, which in its turn results in a lower execution time for the gBSD generation process.

- *resolution*: as mentioned above, the bit rate is dependent on the resolution. Hence, higher resolution bitstreams will have higher execution times for the gBSD generation process. Note that in case of the two-step approach, only the BintoBSD parser is dependent on the resolution, because the STX processor only has to deal with the format-specific BSD. The size of the latter is independent of the resolution of the underlying bitstream.

- *# frames*: varying the number of frames reveals that the gBSD generation has linear execution times. This is also illustrated in Figure 3.12, where the linear behavior of both the two-step approach and gBFlavor is shown.

- *# scalability layers*: the number of scalability layers is an important factor for the gBSD generation process. It does not only have an impact on the bit rate of the bitstream, but also on the size of the format-specific BSDs, as can be seen in Table 3.6. For the SVC coding format, the number of scalability layers influences the number of NALUs, which are the main parse units inside an SVC bitstream. Less scalability layers imply a decrease of the number of NALUs. The less NALUs occur in the bitstream, the lower the execution times for the gBSD generation process and the lower the sizes of the (g)BSDs will be (Table 3.7).

Table 3.6: Bitstream characteristics for the different SVC test sequences.

Name	Resolution	# frames	# NALUs	Frame rate (Hz)	Size (MB)	Length (s)	Bit rate (MBit/s)	# temporal layers	# spatial layers	# quality layers
Crew1	1280x720	1200	13208	60	18.7	20	7.5	4	3	3
Cyclists	1280x720	1200	13208	60	10.0	20	4.0	4	3	3
Crew2	704x576	1200	13208	60	10.6	20	4.2	4	3	3
Crew3	704x576	1200	13208	30	12.7	40	2.5	4	3	3
Crew4	704x576	600	6608	30	6.4	20	2.5	4	3	3
Crew5	704x576	300	3308	30	3.2	10	2.5	4	3	3
Crew6	704x576	1200	13208	30	12.3	40	2.5	6	3	3
Crew7	704x576	1200	6008	30	8.5	40	1.7	4	1	3
Crew8	704x576	1200	6008	30	11.3	40	2.3	4	3	1

Table 3.7: Execution times and file sizes for the gBSD generation process (SVC).

BSDL + STX								
Name	BintoBSD (s)	BSD Size (MB)	compr. BSD Size (KB)	STX (s)	gBSD Size (KB)	compr. gBSD Size (KB)	Speed (MBit/s)	Total time (s)
Crew1	49.2	13.0	70.6	8.7	1379.0	63.6	2.6	57.9
Cyclists	44.2	13.0	67.4	8.6	1371.3	58.6	1.5	52.8
Crew2	45.5	13.0	69.9	8.7	1372.4	61.3	1.6	54.2
Crew3	46.6	13.0	70.9	8.7	1375.5	63.1	1.8	55.2
Crew4	24.1	6.5	38.7	4.9	687.3	32.1	1.8	29.0
Crew5	12.7	3.3	22.4	3.0	343.6	16.4	1.6	15.7
Crew6	46.0	13.0	70.3	8.7	1374.9	62.8	1.8	54.8
Crew7	22.9	5.7	31.0	4.5	624.1	25.9	2.5	27.4
Crew8	24.3	5.7	28.2	4.5	625.0	24.6	3.1	28.8

gBFlavor					
Name	gBSD generator[a] (s)	gBSD Size (KB)	compr. gBSD Size (KB)	Speed (MBit/s)	Total time (s)
Crew1	4.6	1391.9	63.4	32.1	4.6
Cyclists	3.4	1384.2	58.7	23.6	3.4
Crew2	3.5	1385.3	61.2	24.3	3.5
Crew3	3.8	1388.3	62.9	26.9	3.8
Crew4	2.1	693.7	32.0	24.6	2.1
Crew5	1.2	346.8	16.4	21.1	1.2
Crew6	3.7	1387.7	62.6	26.4	3.7
Crew7	2.4	629.9	26.0	28.3	2.4
Crew8	2.7	630.8	24.7	32.7	2.7

[a] 'gBSD generator' denote the parser that is generated with the gbflavorc translator.

Table 3.8: Execution times and file sizes for the gBSD transformation and bitstream generation process. The last column denotes the total execution time of the gBFlavor-enabled adaptation framework.

SVC							
Name	Version T-S-Q[a]	STX (s)	gBSD Size (MB)	compr. gBSD Size (KB)	gBSDtoBin (s)	Bitstream Size (MB)	Tot. time (s)
Crew	2-1-1	1.0	0.4	17.3	0.5	2.6	6.2
	2-2-2	1.1	0.7	34.2	1.1	14.6	6.9
	3-1-2	1.1	1.0	44.0	0.8	6.2	6.6
	3-2-1	1.2	1.0	45.3	1.0	10.5	6.8
Cyclists	2-1-1	1.0	0.4	16.4	0.5	1.4	4.9
	2-2-2	1.1	0.7	32.6	0.8	8.6	5.3
	3-1-2	1.1	1.0	40.7	0.7	3.4	5.2
	3-2-1	1.2	1.0	41.9	0.8	5.5	5.4
Raven	2-1-1	1.0	0.4	16.7	0.5	1.8	5.2
	2-2-2	1.1	0.7	32.9	1.0	11.2	5.8
	3-1-2	1.1	1.0	41.3	0.7	4.2	5.6
	3-2-1	1.2	1.0	42.5	0.8	6.5	5.7

JPEG2000							
Name	Version C-S-Q[b]	STX (s)	gBSD Size (MB)	compr. gBSD Size (KB)	gBSDtoBin (s)	Bitstream Size (KB)	Tot. time (s)
Image1	0-4-4	0.8	27.5	2.2	0.2	6.3	2.5
	0-9-22	0.9	160.8	6.1	0.4	2727.3	2.9
	2-9-4	0.8	104.8	4.4	0.3	70.7	2.7
	2-4-22	0.9	251.4	7.9	0.4	41.7	2.9
Image2	0-4-4	0.8	27.6	2.2	0.2	19.4	3.3
	0-9-22	0.9	161.2	6.2	0.7	7993.8	3.8
	2-9-4	0.8	105.1	4.3	0.3	175.3	3.4
	2-4-22	1.0	251.8	8.2	0.4	96.9	3.6
Image3	0-4-4	0.8	27.6	2.2	0.2	15.3	2.5
	0-9-22	0.9	161.2	6.1	0.6	6690.2	5.0
	2-9-4	0.8	105.2	4.4	0.3	183.8	4.7
	2-4-22	1.0	251.9	8.2	0.4	105.5	4.9

[a] T, S, and Q denote the number of temporal, spatial, and quality layers in the bitstream.
[b] C, S, and Q denotes the number of colour, spatial, and quality layers in the bitstream.

3.4.2 Transformation and Adapted Bitstream Generation

Table 3.8 contains the execution times and file sizes for the adaptation and bitstream generation processes. The adaptation is implemented by means of STX stylesheets (one for each coding format). The adapted gBSD is then fed to MPEG-21 DIA's gBSDtoBin parser in order to obtain the desired adapted bitstream. As one can see, the actual adaptation process can be executed very fast (approximately one second). This is because the transformation is based on the markers inside the gBSD, which avoid complex computations during the transformation. The gBSDtoBin processor also performs very well and is highly dependent on the size of the resulting (adapted) bitstream. Indeed,

higher bitstream sizes imply more I/O-operations and hence higher execution times.

The last column of Table 3.8 shows that a gBFlavor-enabled adaptation framework is able to adapt media resources very efficiently (i.e., 5.8 s and 3.5 s on average for SVC and JPEG2000 respectively). Moreover, it is clear that gBFlavor enables the creation of a real-time XML-driven adaptation framework (less than 7 s for an SVC video of 20 s), in contrast to the two-step approach. Furthermore, the memory consumption is low and constant during the full adaptation chain for both gBFlavor and the combination of BSDL and STX (a maximum of 2.0 MB).

3.5 Conclusions and Original Contributions

Due to the lack of an efficient method for the generation of gBSDs in a format-independent manner, we proposed gBFlavor in this chapter. gBSDs are used in an MPEG-21 gBS Schema-based framework for content adaptation. Such a framework is able to adapt scalable bitstreams in a format-independent manner. The gBSD generation process is not standardized within MPEG-21 DIA, implying that a gBSD may be generated in a proprietary way. In this chapter, we discussed and evaluated existing methods for gBSD generation. The two-step approach as currently proposed in the scientific literature is an existing format-independent solution. More specifically, gBSDs are created by a format-specific BSD generation step, followed by a transformation into a gBSD. However, the latter solution requires knowledge of two different technologies (i.e., BSD generation and BSD-to-gBSD transformation). Furthermore, often more detail is needed in the BSD than in the resulting gBSD, implying a decrease in execution speed of the format-specific BSD generation process. Note that these detailed BSDs are necessary to set proper markers during the BSD-to-gBSD transformation.

gBFlavor makes it possible to automatically generate a format-specific parser that is able to produce an application-specific gBSD for a given bitstream. Such a parser is generated by the gbflavorc translator, which takes as input a gBFlavor code. We have proposed the specification for this code, which describes the high-level structure of a particular coding format. The gBFlavor specification, which is an extension of the BFlavor specification, provides support for the addition of semantically meaningful information to gBSDs (by means of markers). Hence, gBSDs targeting specific applications can be obtained.

In order to get an estimate of the expressive power and performance of a gBFlavor-enabled adaptation framework, we have compared the gBFlavor

approach with the existing two-step approach. The creation of gBSDs using gBFlavor avoids the two-step approach, since only one technology is needed to obtain gBSDs targeting a specific application. The exploitation of the scalable properties of two coding formats, i.e., SVC and JPEG2000, was used as the target application in our gBFlavor-enabled adaptation framework. Performance results have shown that gBFlavor outperforms the two-step approach in terms of execution speed.

The research that has led to this chapter is also described in the following publications.

1. W. De Neve, D. Van Deursen, D. De Schrijver, K. De Wolf, and R. Van de Walle. Using Bitstream Structure Descriptions for the Exploitation of Multi-layered Temporal Scalability in H.264/AVC's Base Specification. *Lecture Notes in Computer Science – Advances in Mulitmedia Information Processing*, volume 3768, pages 641–652, November 2005

2. D. Van Deursen, W. De Neve, D. De Schrijver, and R. Van de Walle. BFlavor: an Optimized XML-based Framework for Multimedia Content Customization. In *Proceedings of the 25th Picture Coding Symposium*, 6 pages on CD-ROM, April 2006, Beijing, China

3. D. Van Deursen, D. De Schrijver, W. De Neve, and R. Van de Walle. A Real-Time XML-Based Adaptation System for Scalable Video Formats. *Lecture Notes in Computer Science – Advances in Multimedia Information Processing*, volume 4261, pages 339–348, November 2006

4. W. De Neve, D. Van Deursen, D. De Schrijver, S. Lerouge, K. De Wolf, and R. Van de Walle. BFlavor: a Harmonized Approach to Media Resource Adaptation Inspired by MPEG-21 BSDL and XFlavor. *Signal Processing: Image Communication*, 21(10):862–889, November 2006

5. D. Van Deursen, W. De Neve, D. De Schrijver, and R. Van de Walle. Automatic Generation of generic Bitstream Syntax Descriptions Applied to H.264/AVC SVC Encoded Video Streams. In *Proceedings of the 14th International Conference on Image Analysis and Processing*, pages 382–387, September 2007, Modena, Italy

6. D. Van Deursen, W. De Neve, D. De Schrijver, and R. Van de Walle. gBFlavor: a New Tool for Fast and Automatic Generation of generic Bitstream Syntax Descriptions. *Multimedia Tools and Applications*, 40(3):453–494, December 2008

Chapter 4

Model-driven content adaptation

Enthusiasm is everything. It must be taut and vibrating like a guitar string.

Pelé (1940 -)

4.1 Introduction

In Chapter 2, we gave an overview of format-independent content adaptation. Adaptation systems operating independently of the underlying coding format are based on automatically generated XML-based descriptions of the high-level structure of the media resources [3]. By performing simple, high-level operations such as the removal of particular data blocks and the modification of the value of certain syntax elements, media resources can be customized in a format-independent way. However, XML-driven content adaptation still has a number of issues:

- creators of XML filters have to implement low-level, coding-format dependent algorithms that are needed to obtain format-independent adaptation operations;
- the integration of semantic adaptation operations and content metadata standards is realized in an ad-hoc way.

In this chapter, we tackle the problems of XML-driven content adaptation and present a new method for the adaptation of media resources in a format-

independent way, called model-driven content adaptation[1]. It is inspired by the principles of XML-driven content adaptation techniques such as BSDL or gBS Schema, while its final design is based on a model describing structural, content, and scalability information of media bitstreams. Existing coding and metadata formats are mapped to this model in order to be supported. Hence, adaptation operations (e.g., frame rate scaling or scene selection) can be expressed and implemented based on this model, in a way that is independent of the underlying coding format. Instead of using XML, Semantic Web technologies such as the Resource Description Framework (RDF, [72]), the Web Ontology Language (OWL, [83]), and the SPARQL Protocol And RDF Query Language (SPARQL, [99]) are used to implement our adaptation technique. This way, the interoperability between different metadata standards can be enhanced thanks to a more natural representation of objects and relationships, as reported in [6, 41, 122, 139].

The organization of this chapter is as follows. Section 4.2 elaborates on a number of problems of XML-driven content adaptation. A model for media bitstreams is presented in Section 4.3. Our new method for multimedia content adaptation, i.e., model-driven content adaptation, is discussed in Section 4.4. This new technique is compared to other approaches in Section 4.5. Performance results are provided in Section 4.6. Finally, conclusions are drawn in Section 4.7.

4.2 Problem Description

As discussed in the introduction, there are still a number of issues regarding current content adaptation systems based on XML descriptions of the high-level structure of media bitstreams. In this section, we discuss these issues in more detail by means of simple examples. Suppose we have video content encoded by making use of MPEG-2 Video and H.264/AVC. A BSD is created using BSDL's BintoBSD parser for both sequences, as shown in Figure 4.1. Now suppose that we want to express the adaptation operation 'remove the highest temporal layer', which will result in a decrease of the frame rate. In Figure 4.1, workflow diagrams are shown that correspond to XML filters expressing the adaptation operation. Note that for the H.264/AVC example, a simplified algorithm is given; more information on exploitation of temporal scalability in H.264/AVC can be found in [30].

[1] Model-driven content adaptation is equivalent to RDF-driven content adaptation, which is the term used in publications.

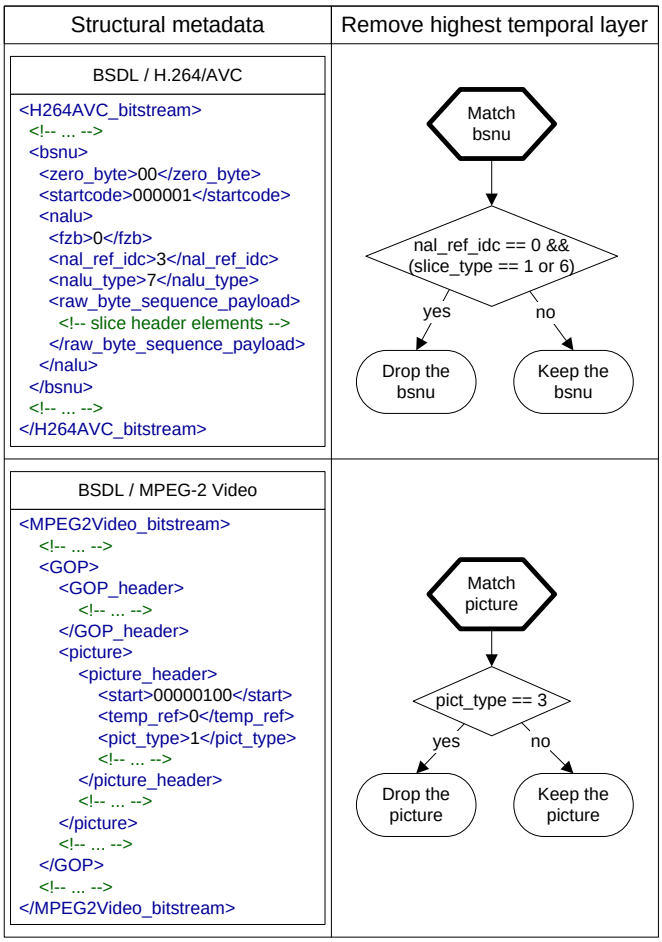


Figure 4.1: Implementing temporal scalability using XML-driven content adaptation.

High-level Adaptation Operations

The actual adaptation engines in an XML-driven content adaptation system are format-independent, while the descriptions themselves are format-specific. Hence, an XML filter implementing a particular adaptation operation (e.g., removal of temporal layers in a scalable video stream) needs to know the structure of these descriptions, and is thus dependent on the coding format. Therefore, multiple format-dependent XML filters need to be created to implement the same adaptation operation for different formats, as illustrated in Figure 4.1. The functionality of these XML filters is the same: calculate the temporal level for each video frame and decide whether the frame should be dropped or not.

However, the calculation of the temporal levels is format-specific and should be avoided during the BSD transformation step.

The same observation can be made for semantic adaptation operations, which typically express a selection of media fragments based on content metadata (e.g., selection of sport fragments in a news sequence). Content metadata are often linked to the multimedia content by means of timestamps, whereas structural metadata use bit offsets. Therefore, a mapping between the timestamp values of the content metadata and the bit offsets of the structural metadata needs to be implemented in the XML filter. Such a mapping is dependent on the underlying coding format and should be avoided during the BSD transformation step.

Hence, despite the use of an abstraction layer (i.e., a BSD) within XML-driven content adaptation, the adaptation operations themselves are not abstracted. More specifically, compared to format-specific adaptation software, the format-dependency is shifted from the binary to the XML domain. Creators of XML filters cannot think in terms of high-level and format-independent adaptations but have to be aware of the underlying coding formats. Hence, with the current XML-driven approach, format-independency is obtained in an ad-hoc manner.

Integration with Other Metadata Formats

The second problem regarding the integration of semantic adaptation operations and content metadata formats is related to the interoperability issues of XML [41,139]. Metadata formats typically consist of an XML Schema accompanied with plain text (e.g., MPEG-7). The text usually describes the semantic meaning of the XML tags specified in the XML Schema. The role of XML Schema is to define a grammar for XML documents. However, when it comes to semantic interoperability and making data understandable, XML has disadvantages. XML's major limitation is that it only describes grammars. It is impossible to recognize a semantic unit from a particular domain because XML aims at document structure and imposes no common interpretation of the data contained in the document [6,41,122]. For instance, taking into account different metadata standards, the same tags can have different meanings while tags with the same meaning can occur in different structures. Hence, expressing the semantic adaptation operation 'select sport fragments' in a scenario where two different content metadata standards are used (e.g., annotations are provided in both MPEG-7 and TV-Anytime), requires the development of two different XML filters (i.e., one that can interpret MPEG-7 and one that can interpret TV-Anytime). Note that the previous observation also holds true for different

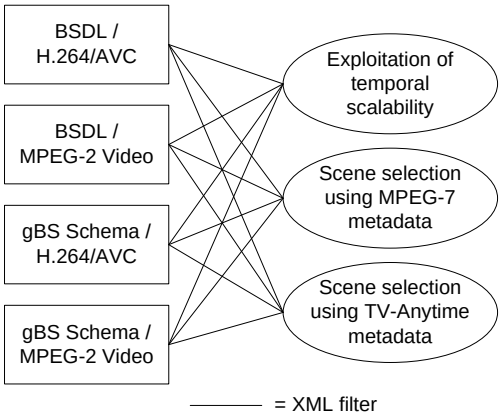


Figure 4.2: Problems with XML-driven content adaptation.

structural metadata standards. Examples of standardized solutions for expressing structural metadata are BSDL and gBS Schema; however, other representations such as XFlavor are also possible. It is obvious that XML filters are also dependent on the technology used for the representation of structural metadata.

Both problems are also illustrated in Figure 4.2, where video content is encoded by making use of MPEG-2 Video and H.264/AVC. Furthermore, content metadata regarding the different scenes in the video is available in both MPEG-7 and TV-Anytime. Finally, metadata regarding the high-level structure of the encoded video bitstreams is provided by descriptions compliant with both BSDL and gBS Schema. As can be seen in this figure, XML filters performing the BSD transformation step (discussed in Section 2.2.1) are depending on the coding format (i.e., H.264/AVC and MPEG-2 Video), the metadata format of the structural metadata (i.e., BSDL and gBS Schema), and the metadata format of the content metadata (i.e., MPEG-7 and TV-Anytime).

4.3 Modeling Media Bitstreams

In order to solve the problems discussed in Section 4.2 (i.e., format-specific XML filters and an ad-hoc integration of semantic adaptation operations and content metadata standards), we propose a novel format-independent adaptation technique.

As mentioned in Section 4.2, XML filters are dependent on the underlying coding format because the descriptions of the high-level structure of media bitstreams are format-dependent. As such, despite the fact that the underlying

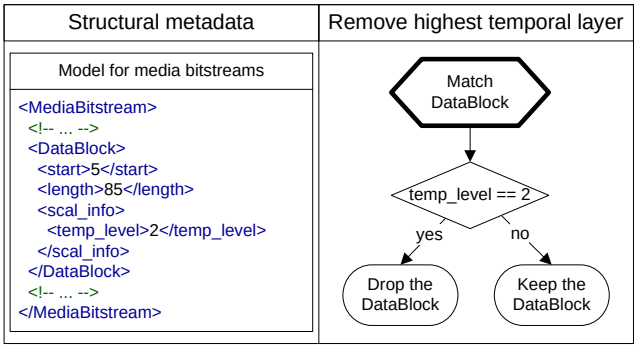


Figure 4.3: Expressing adaptation operations on top of format-independent BSDs.

adaptation engines are independent of the coding format, the actual transformation logic is not. Therefore, the transformations of the structural metadata should be shifted to a higher level. This can only be realized if format-specific calculations (e.g., calculation of temporal levels or calculation of a timestamp for a particular frame) can be avoided during the BSD transformation step. Hence, we propose to shift these format-specific calculations from the BSD transformation step to the BSD generation step. As a result, we are able to obtain a fully format-independent BSD that enables the expression of high-level adaptation operations (an example can be found in Figure 4.3). We have defined the structure and semantics of such a fully format-independent BSD in the form of a model for media bitstreams. That way, the adaptation operations can be formulated in terms of transformations based on the model; hence they are shifted to a higher level, independent of the coding format. Our model for media bitstreams is based on previous work regarding the definition and implementation of a model for media bitstreams [76,89,130]. However, the latter only provides support for high-level structural adaptations and uses XML as underlying technology.

To obtain a seamless integration between semantic adaptation operations and content metadata standards, our model for media bitstreams needs to support semantic adaptation operations. More specifically, it has to enable the extraction of specific media fragments from a media bitstream, independent of the coding format. Furthermore, the model has to provide hooks to connect to existing metadata standards. That way, semantic adaptation operations can be expressed by using concepts defined in existing content metadata standards.

We have avoided XML Schema to describe our model because the use of XML as underlying technology causes interoperability problems between dif-

ferent metadata standards, as discussed in Section 4.2. In contrast to XML, Semantic Web technologies such as RDF and OWL enhance the interoperability among metadata standards for multimedia content [139] (see also Section 1.1.2). Therefore, our model for media bitstreams is implemented by using OWL[2]. The instances of the model (i.e., the structural metadata or BSDs) are expressed in RDF. The transformation of the structural metadata is implemented by using SPARQL queries, which are independent of the coding format.

To summarize, the requirements for our model for media bitstreams can be listed as follows:

- fully independent of coding formats;
- support for expressing adaptation operations (both structural and semantic) independent of coding formats;
- description of the link between low-level coding structures and high-level media segments;
- seamless integration between adaptation operations and other metadata models.

4.3.1 Model for Media Bitstreams

As elaborated on above, a model for media bitstreams is needed in order to abstract the transformation of the structural metadata. In this section, we present such a model covering structural, content, and scalability information. As discussed above, we implemented the model by using OWL (see Listing C.1). In Figure 4.4, an overview is given of the model. More detailed views on this model are provided in the next subsections. Note that for figures visualizing (parts of) the multimedia model, ellipses, rectangles, and arrows represent OWL classes, literals, and properties respectively.

4.3.1.1 Structural Metadata

The modeling of the high-level structure of a media bitstream is shown in Figure 4.5. The *MediaBitstream* class corresponds to a particular compressed, elementary media bitstream. It contains a description of the underlying format (e.g., H.264/AVC) (i.e., the *format* property) and a reference to the location of the media bitstream by means of the *bitstreamSource* property.

[2]Using RDFS to implement our model is not sufficient, because we need OWL-specific constructs such as the transitivity relationship.

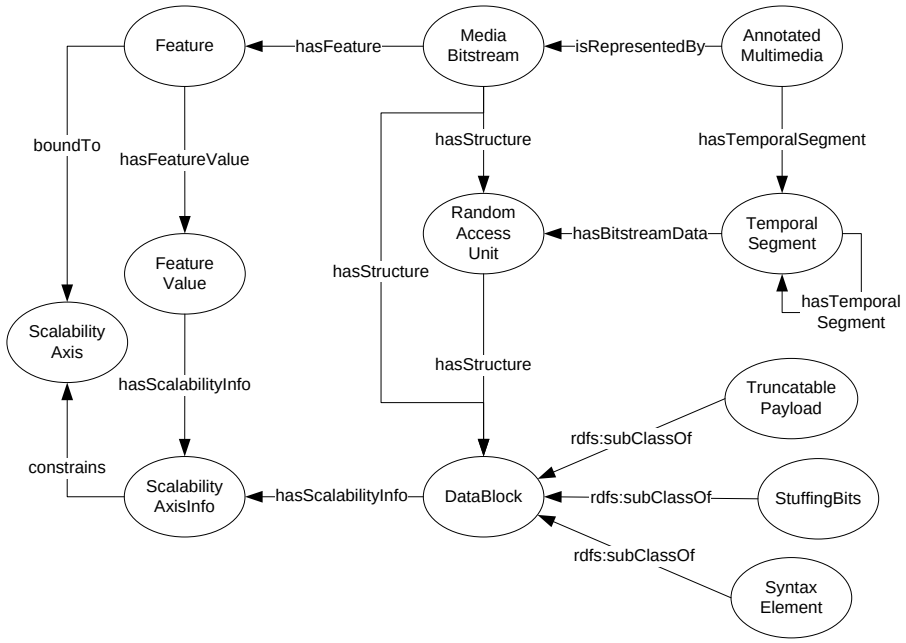


Figure 4.4: Overview of the multimedia model.

A *MediaBitstream* consists of a number of *RandomAccessUnit*s. Random access refers to the ability of the decoder to start decoding at a point in a compressed media bitstream other than at the beginning and to recover an exact representation of the decoded bitstream [25, 49]. A *RandomAccessUnit* contains a number of successive *DataBlock*s, pointing to particular segments in the media bitstream. More detailed information regarding the modeling of *DataBlock*s is provided in Section 4.3.1.3.

The property *hasStructure*, which is used to connect the classes in the structural metadata, is a transitive property. This means that if a pair (x, y) is an instance of *hasStructure*, and the pair (y, z) is also an instance of *hasStructure*, then we can infer that the pair (x, z) is also an instance of *hasStructure*. We introduce this transitivity relationship to express that *DataBlock*s are always contained in *MediaBitstream*s (even if they are located within a *RandomAccessUnit*). Additionally, it is possible that a *MediaBitstream* has *DataBlock*s not occurring in *RandomAccessUnit*s (e.g., a Sequence Parameter Set (SPS) in H.264/AVC).

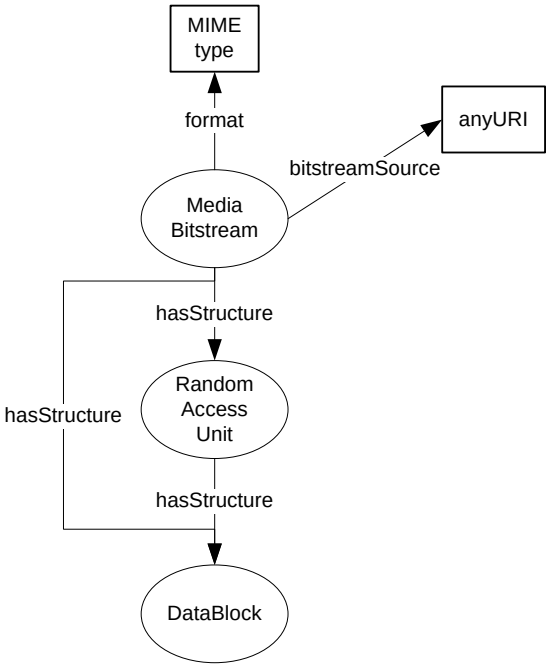


Figure 4.5: Model for the structural metadata.

4.3.1.2 Scalability Information

The scalability information part of the model provides information regarding properties of the media bitstream that are related to possible adaptation operations. It enables the declaration of what types of adaptations may or should be applied to the media bitstream in order to optimally fit a given context [87]. The modeling of the scalability information is depicted in Figure 4.6. A *MediaBitstream* contains a number of *Features*, each containing a specific *unit*. An example of a feature is 'frame rate', having as unit 'fps'. Each *Feature* contains one or more *FeatureValue*s. Note that the data type of the *hasValue* property is not specified since this is dependent on the feature. Furthermore, a *Feature* can be bound to one or more *ScalabilityAxes* containing an amount of levels (e.g., a temporal scalability axis containing four levels). A *FeatureValue* can be linked to a *ScalabilityAxisInfo* class, which provides information regarding the relationship between the scalability axes and the feature values. This is established by constraining a *ScalabilityAxis* by means of a level (e.g., 15 fps corresponds to the second level of the temporal scalability axis).

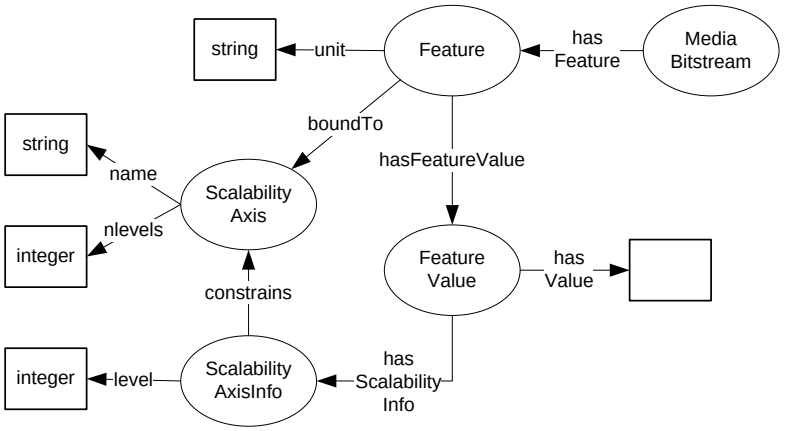


Figure 4.6: Model for the scalability information.

4.3.1.3 Data Blocks

As already mentioned above, *DataBlock*s point to particular segments in the media bitstream. In Figure 4.7, the modeling of a data block is shown in detail. A *DataBlock* is always characterized by two properties: the *start* and *length* of the datablock in terms of bits. Additionally, a datablock can contain *ScalabilityAxisInfo* (see Section 4.3.1.2), indicating to which scalability layers the data block belongs.

Three possible subclasses exist for a *DataBlock*.

- *TruncatablePayload*: points to a bitstream segment that can be truncated by a number of bytes, something that typically occurs within bitstreams encoded with Fine Granularity Scalability (FGS) techniques.

- *StuffingBits*: allow to write padding bits to the output bitstream until it is byte-aligned. The property *nbytes* determines on how many bytes the bitstream is aligned. E.g., if *nbytes* is equal to four, then padding bits are added until the output bitstream is aligned on four bytes.

- *SyntaxElement*: represents a specific syntax element of the media bitstream. The *value* property indicates the value of the syntax element, i.e., the decimal representation of the bits covered by the syntax element (specified by the *start* and *length* property).

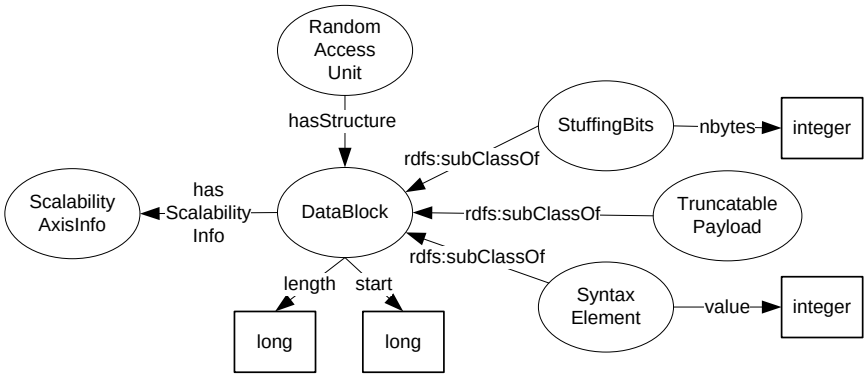


Figure 4.7: Model for a data block.

4.3.1.4 Content Metadata

A *MediaBitstream* can be annotated by means of *AnnotatedMultimedia*, which contains a description of the content of the *MediaBitstream*. The content metadata model is depicted in Figure 4.8. *AnnotatedMultimedia* consists of a number of *TemporalSegment*s, each pointing to a specific segment of the multimedia content by means of a *start* and *duration* property. A *TemporalSegment* can in its turn contain a number of other *TemporalSegment*s, allowing to model a hierarchy of *TemporalSegment*s. Furthermore, a *TemporalSegment* has a *keyword* property, allowing a simple annotation of the multimedia content by means of keywords (i.e., a form of tagging). More complicated descriptions of *TemporalSegments* can be obtained by linking existing ontologies to our content metadata model. More information regarding the linking of other ontologies to our multimedia model is provided in Section 4.3.2.2.

The connection between the media bitstream and the *TemporalSegment*s is established by means of the *hasBitstreamData* property. Each *RandomAccessUnit* of a media bitstream can belong to one or more *TemporalSegment*s. This way, the *TemporalSegment*s are connected to the bits of particular media bitstreams. Note that *AnnotatedMultimedia* points to the multimedia content by means of timestamps, independently of the coding format, while a *MediaBitstream* points to specific bitstream segments in terms of bits.

Furthermore, since *AnnotatedMultimedia* can be linked to different *MediaBitstream*s, the model for media bitstreams provides support for media bitstream selection (see Chapter 1). Indeed, it is possible to relate multiple *MediaBitstream* instances (each corresponding to a different version in terms of coding format, bit rate, etc.) to one *AnnotatedMultimedia* instance.

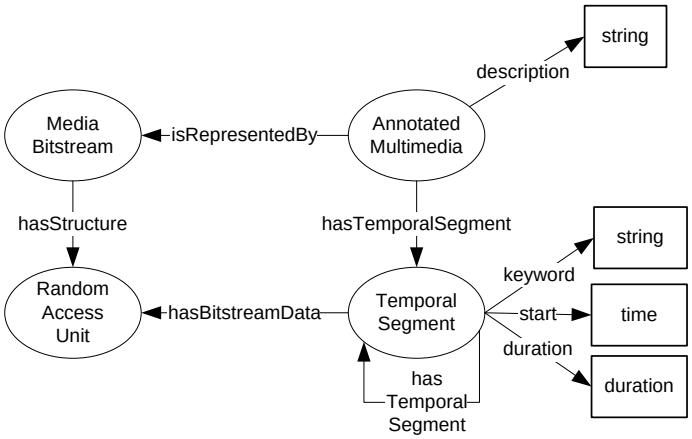


Figure 4.8: Model for the content metadata.

4.3.2 The Multimedia Model in Practice

In this section, the relationship between the multimedia model presented in Section 4.3.1 and existing coding and metadata formats is illustrated.

4.3.2.1 Mapping H.264/AVC to the Multimedia Model

Existing multimedia coding formats need to be mapped to the model defined in Section 4.3.1. As an example, we map the H.264/AVC coding format to the model. Figure 4.9 provides a visual representation of the mapping of an H.264/AVC encoded bitstream to the multimedia model. Note that an excerpt of the resulting RDF triples is shown in Listing C.2.

An H.264/AVC encoded bitstream is a succession of Network Abstraction Layer Units (NALUs). Different NALU types exist: a Sequence Parameter Set (SPS), which contains information related to the whole sequence; a Picture Parameter Set (PPS), which contains information related to a set of pictures; and a slice layer, which contains the actual encoded data such as the motion vectors and the residual data.

As shown in Figure 4.9, when mapping an H.264/AVC encoded bitstream to the multimedia model, each NALU is mapped to a *DataBlock*. The SPS and PPS *DataBlock*s are directly connected to the *MediaBitstream*. Furthermore, the Instantaneous Decoding Refresh (IDR) slices indicate the start of a new *RandomAccessUnit*.

Slice *DataBlock*s are provided with *ScalabilityInfo* indicating in which scalability layer the data block is located. In this case, only one *ScalabilityAxis*

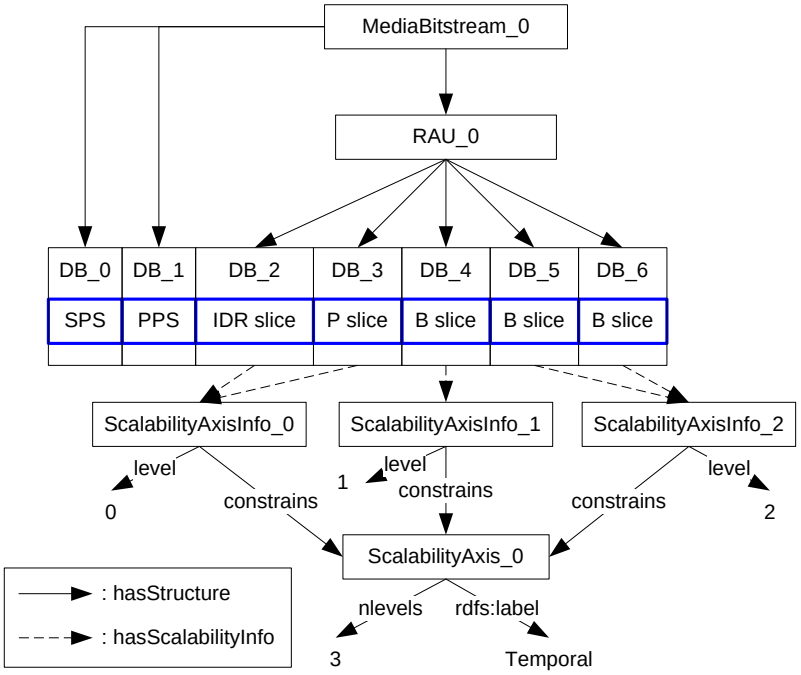


Figure 4.9: Describing an H.264/AVC bitstream with the multimedia model. DB and RAU denote DataBlock and RandomAccessUnit respectively.

is present, i.e., the temporal scalability axis containing three levels. Since the example H.264/AVC bitstream is encoded using hierarchical B-pictures, the first B-picture is located in the second temporal layer, while the other two B-pictures are located in the third temporal layer. I- and P-pictures are located in the first temporal layer [30].

Figure 4.10 illustrates the description of the features of the bitstream. For instance, a feature in this example is 'frame rate' with possible *FeatureValue*s 6.25, 12.5, and 25 fps. These values correspond to the first, second, and third temporal layer respectively.

4.3.2.2 Linking the Content Metadata Model to Existing Ontologies

As discussed in Section 4.3.1.4, the multimedia model allows to create simple annotations of the multimedia content by adding keywords to *AnnotatedMultimedia* or *TemporalSegment* instances. However, more sophisticated content descriptions are often needed. Therefore, the content metadata model provides hooks for existing ontologies. An example is shown in Figure 4.11,

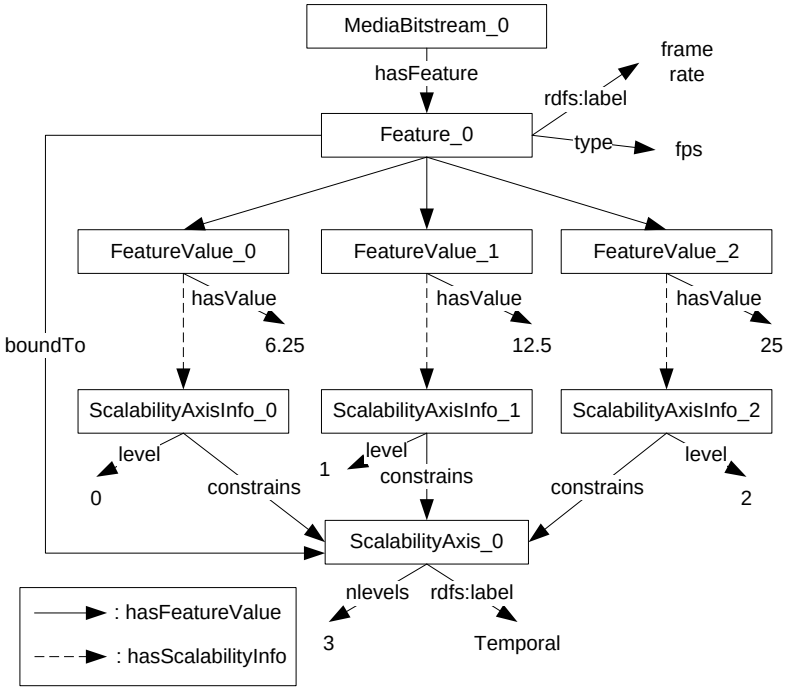


Figure 4.10: Describing scalability information of an H.264/AVC bitstream with the multimedia model.

where a class *Person*, defined in the Friend-Of-A-Friend (FOAF[3]) ontology, is connected with *TemporalSegment*s and *AnnotatedMultimedia* by means of the *foaf:depicts* property. A second example is shown in Figure 4.12, where concepts described in a soccer ontology are linked to *AnnotatedMultimedia* and *TemporalSegment*. This way, media bitstreams that are annotated as illustrated above can be adapted based on the concepts defined in the linked ontologies. For example, fragments can be selected that are related with a specific person corresponding to a URI that is an instance of the class *Person* defined in the FOAF ontology.

The content metadata model discussed in Section 4.3.1.4 can also be linked to existing ontologies specifying how to connect content descriptions to parts of a media asset. Examples are MPEG-7 [53], DIG35[4], and Exif[5].

[3] `http://xmlns.com/foaf/spec/`

[4] `http://www.w3.org/2005/Incubator/mmsem/XGR-vocabularies/#formal-DIG35`

[5] `http://www.w3.org/2005/Incubator/mmsem/XGR-vocabularies/#formal-Exif`

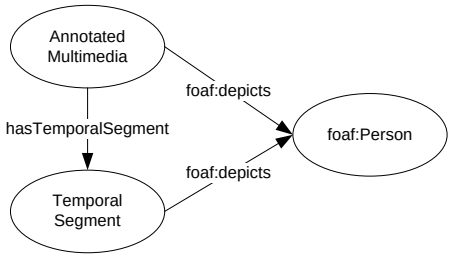


Figure 4.11: Linking domain-specific ontologies with the content metadata model.

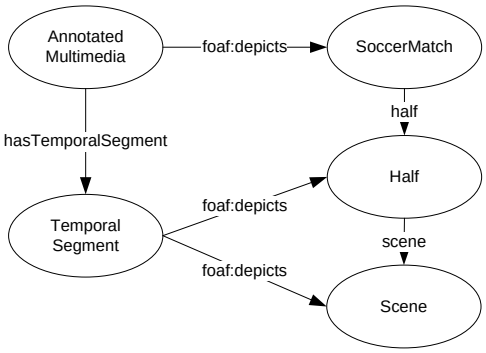


Figure 4.12: Extending the content metadata model with domain-specific ontologies.

4.3.2.3 The Model for Media Bitstreams versus COMM

Arndt *et al.* have proposed the Core Ontology for MultiMedia (COMM, [6]), which is a formal description of a high quality multimedia ontology that is compatible with existing Semantic Web technologies. COMM is based on both MPEG-7 [78] and the DOLCE[6] foundational ontology [82]. More specifically, it is composed of *multimedia patterns* specializing the DOLCE design patterns for Descriptions & Situations and Information Objects and covers a very large part of the MPEG-7 standard. COMM defines four multimedia patterns, based on the two most important functionalities provided by MPEG-7 (i.e., decomposition of a media asset and the annotation of its parts):

- *media annotation pattern*: description of the physical instances of multimedia content;
- *decomposition pattern*: description of spatial, temporal, spatiotemporal, and media source decompositions of multimedia content into segments;

[6]DOLCE stands for Descriptive Ontology for Linguistic and Cognitive Engineering.

- *content annotation pattern*: description of the attachment of MPEG-7-specific metadata to the multimedia content (e.g., dominant colors);
- *semantic annotation pattern*: a specialization of the content annotation pattern which describes the connection of multimedia content with domain-specific ontologies.

Although the goals of COMM and our multimedia model are different (i.e., formally describing multimedia content versus multimedia content adaptation), they share the following functionalities.

- *Media bitstream description*: COMM uses the media annotation pattern to describe media bitstreams. Within our model, the *MediaBitsteam* class is used to represent a media bitstream.
- *Temporal and media source decomposition description*: COMM uses the decomposition pattern to describe, amongst others, temporal and media source decompositions of multimedia content. Within our model, the *TemporalSegment* class is used to represent temporal decompositions of multimedia content. Further, the *AnnotatedMultimedia* class is used to describe media source decompositions. More specifically, using the *isRepresentedBy* property, multimedia content (i.e., *AnnotatedMultimedia*) can be decomposed into different media sources (i.e., *MediaBitstream*s).

Next to these similarities, there are also a number of differences between COMM and our multimedia model. The following functionalities are supported by our multimedia model, but are not available within COMM (mainly because COMM is not focussed on multimedia content adaptation):

- *structural decomposition*: decomposition of a *MediaBitstream* into *RandomAccessUnit*s and *DataBlock*s;
- *link between structural and temporal decomposition*: *RandomAccessUnit*s are connected to *TemporalSegment*s by using the *hasBitstreamData* propery;
- *link between scalability axes and media bitstream features*: our model for scalability information (see Figure 4.6) is able to link scalability axes and features of media bitstreams (e.g., link the feature 'frame rate' to the temporal scalability axis);
- *link between scalability axes and structural decomposition*: *DataBlock*s are connected to scalability axes through the *ScalabilityAxisInfo* class.

We can conclude that COMM and our multimedia model partly overlap with each other. Therefore, it would be interesting to investigate how both models can be linked to each other. Connecting our multimedia model to COMM would result in a number of benefits such as a much more extensive content metadata model, MPEG-7 compliance, a formal definition for linking to domain-specific ontologies, and an enhanced interoperability within the Linked Data cloud[7]. The following approaches could be followed to link both models.

- *Definition of a mapping between the overlapping parts of both models*: the relationship between both models could be expressed by using common ontology matching techniques [45]. For instance, the OWL constructs *sameAs*, *equivalentClass*, and *equivalentProperty* or more sophisticated mapping rules could be used.
- *Extending COMM*: since COMM is extensible through the use of multimedia patterns, we could start from COMM and extend it with the same functionalities as our current multimedia model. This could for instance be realized by extending COMM's media annotation pattern or by defining new multimedia patterns to express the structural decomposition and scalability information. This way, our scalability and structural models will be rebuilt with a well designed foundational ontology as a modeling basis.

4.4 Model-driven Content Adaptation

In order to cope with the problems of XML-driven content adaptation mentioned in Section 4.2, we present a new technique for multimedia content adaptation in a format-independent way, i.e., model-driven content adaptation [133]. On the one hand, Semantic Web technologies are used to represent the metadata for multimedia content. On the other hand, a model for media bitstreams covering structural, content, and scalability information (defined in Section 4.3) is used to abstract the transformation process.

4.4.1 Metadata Generation

Several possibilities exist to generate metadata compatible with the multimedia model defined in Section 4.3.1. Figure 4.13 shows an example of an architecture which takes as input raw media bitstreams. The first step is to extract

[7]For more information, visit `http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`.

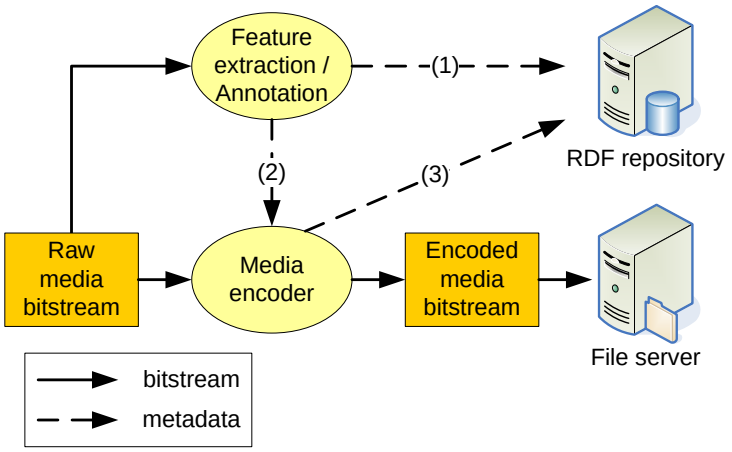


Figure 4.13: Example architecture for the generation of metadata compatible with the multimedia model.

features (e.g., shot segmentation) and/or to manually add annotations regarding the multimedia content. The resulting content metadata will consist of instances of *AnnotatedMultimedia* and *TemporalSegment*s and is stored in an RDF repository (arrow (1) in Figure 4.13). Next, the raw media bitstream is encoded. The encoded media bitstream is sent to a file server. During the encoding process, the structural metadata is generated (arrow (3) in Figure 4.13). More specifically, a *MediaBitstream* instance is created accompanied by instances of *RandomAccessUnit*s, *DataBlock*s, *Feature*s, etc. The content metadata, obtained during the feature extraction and/or annotation, is used by the multimedia encoder (arrow (2) in Figure 4.13) in order to connect the structural metadata with the content metadata (i.e., the bits of the encoded bitstream are linked to the timestamps available in the content metadata). As discussed in Section 4.3.1.4, this is realized by linking *RandomAccessUnit*s to *TemporalSegment*s by using the *hasBitstreamData* property. Finally, the structural metadata is also stored in an RDF repository.

It is important to remark that the scenario described above is not applicable for already encoded media bitstreams, possibly described by a specific metadata format (e.g., H.264/AVC encoded bitstreams annotated with MPEG-7 metadata [26] and described by BSDL). In this case, software is needed to translate the available metadata into metadata compliant with the multimedia model [132]. In case no structural metadata is present, coding-format specific parsers have to be created, taking as input an encoded media bitstream and producing structural metadata compliant with the multimedia model. Note that

such coding-format specific parsers could be automatically generated by using a technique similar to XFlavor [51], which is able to automatically generate a coding-format specific parser producing XML descriptions of the high-level structure of the media bitstreams. The same approach could be followed in the context of model-driven content adaptation. More specifically, instead of producing XML descriptions, such parsers could generate RDF triples compliant with the multimedia model.

4.4.2 General Workflow

Having generated all the necessary metadata, it is now possible to retrieve and adapt a specific media bitstream. The general workflow is depicted in Figure 4.14. There are three main steps during model-driven content adaptation: data block selection, transformation, and binarization. RDF graphs describing data blocks are queried during the data block selection step. These RDF graphs can be adapted during the data block transformation step. Finally, each selected and adapted RDF graph is used to generate the resulting media bitstream. More detailed information regarding the different steps in the workflow is provided in the next subsections.

A comparison of the workflow of XML-driven content adaptation (discussed in Section 2.2.1) and model-driven content adaptation can be made as follows. The generation of an XML description, the transformation of the XML description, and the creation of an adapted bitstream using the transformed XML description correspond to RDF metadata generation, data block selection and transformation, and data block binarization respectively.

4.4.2.1 Data Block Selection

Selecting data blocks is done by taking into account terminal and network characteristics together with user preferences. The preferred data blocks can be obtained by performing the following sequence of steps (see also Figure 4.15).

(1) Content selection: an instance of *AnnotatedMultimedia* is selected based on the user preferences (e.g., select a soccer match of a specific team). Furthermore, the content selection can be refined by selecting only specific fragments that are desired by the user (e.g., only select fragments where a specific soccer player occurs). Moreover, fragments from different instances of *AnnotatedMultimedia* can be selected. The result of the content selection is a list of *TemporalSegment*s corresponding to the fragments selected based on the usage environment.

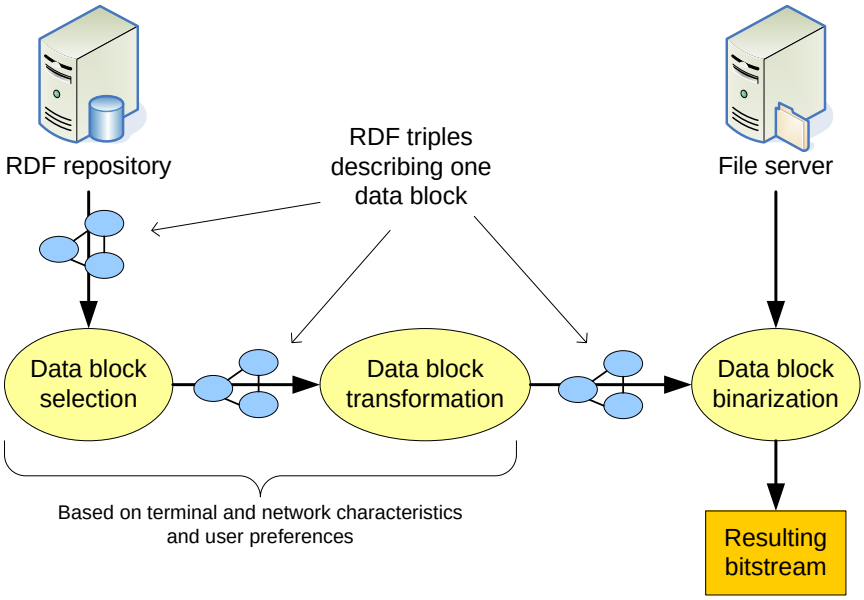


Figure 4.14: The general workflow of model-driven content adaptation.

(2) Bitstream selection: given an instance of *AnnotatedMultimedia*, a *MediaBitstream* representing this multimedia content is selected. Note that this selection can be based on the usage environment (e.g., the device of the end-user only supports a limited amount of coding formats).

(3) Content to structural mapping: the selected *TemporalSegment*s are used to obtain the *DataBlock*s contained in the selected *MediaBitstream*. This is possible by following the links between the *TemporalSegment*s and the *RandomAccessUnit*s.

(4) Structural selection: the subset of *DataBlock*s obtained by the content selection is further restricted by selecting only the *DataBlocks* occurring in specific scalability layers. For instance, if a video stream with a frame rate of 15 fps is requested, only the *DataBlocks* having *ScalabilityInfo* meeting this condition are selected.

Note that it is not necessary to execute each step every time. One could for instance avoid the structural selection if there is no scalability information present.

The selection of data blocks can be performed by using the SPARQL Protocol and RDF Query Language (SPARQL, [99]). This is a query language

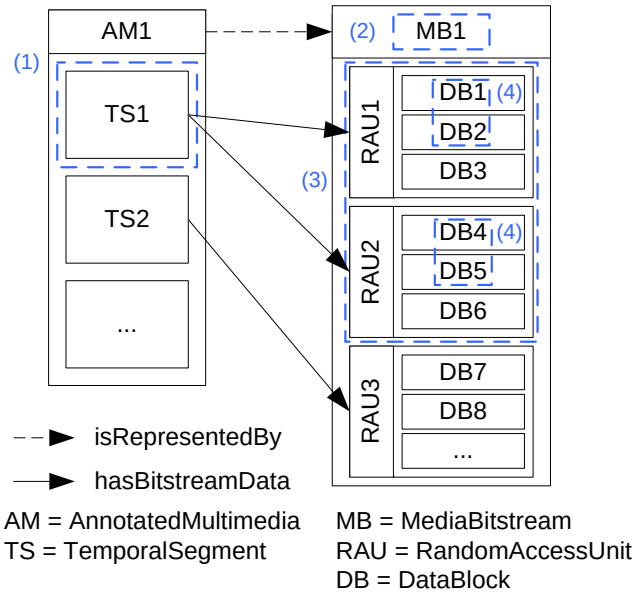


Figure 4.15: Different steps to select data blocks based on the usage environment.

and data access protocol for the Semantic Web, standardized by the RDF Data Access Working Group (DAWG) of the World Wide Web Consortium (W3C). SPARQL offers querying based on triple patterns, conjunctions, disjunctions, and optional patterns; results of SPARQL queries can be results sets or RDF graphs.

In order to demonstrate the data block selection process, a number of SPARQL queries are discussed. Since we want to obtain RDF graphs describing data blocks, we need SPARQL CONSTRUCT queries, i.e., queries resulting in RDF graphs. Listing 4.1 shows a query which selects data blocks occurring in the temporal base layer. Lines 2–8 contain the RDF triples needed to describe one data block. The WHERE clause (lines 9–29) determines which data blocks are selected. In this example, no content metadata is used to select the *MediaBitstream*, i.e., the bitstream is selected based on its label (line 11). Lines 12-21 bind the variables defined in the CONSTRUCT clause. Lines 22–28 specify that only data blocks which occur in the temporal base layer or which do not have scalability information (e.g., SPS or PPS in an H.264/AVC bitstream) are selected. For example, executing this SPARQL query over the RDF triples listed in Listing C.2 (Annex C) results in the RDF triples listed in Listing C.3 (Annex C).

An example of selecting datablocks based on content metadata is provided in Listing 4.2. In this example, data blocks belonging to the first half of a specific soccer match are selected. Lines 3–8 contain the RDF triples needed to describe one data block, analogous to the previous example. In the WHERE clause, the *TemporalSegment* is selected which depicts the first half of the soccer match (lines 13–17). Next, the data blocks are determined by the *RandomAccessUnit*s related to this temporal segment (lines 18–20).

Listing 4.1: Format-independent SPARQL query which selects data blocks occurring in the temporal base layer.

```
1  PREFIX mmo: <multimedia_model.owl#>
   CONSTRUCT {
     ?db rdf:type ?types.
     ?db mmo:start ?start.
5    ?db mmo:length ?length.
     ?db mmo:nBytes ?sb.
     ?db mmo:value ?value.
   }
   WHERE {
10   ?bitstream rdf:type mmo:MediaBitstream.
     ?bitstream rdfs:label 'nieuws_avc'.
     ?bitstream mmo:hasStructure ?db.
     ?db rdf:type ?types.
     ?db mmo:start ?start.
15   ?db mmo:length ?length.
     OPTIONAL {
       ?db mmo:syntaxElementValue ?value.
     }
     OPTIONAL {
20     ?db mmo:nStuffingBytes ?sb.
     }
     OPTIONAL {
       ?db mmo:hasScalabilityInfo ?si_temp.
       ?si_temp mmo:hasScalabilityAxis ?sa_temp.
25     ?sa_temp rdfs:label 'temporal'.
       ?si_temp mmo:level ?temp_level.
     }
     FILTER(!bound(?temp_level) || ?temp_level = 0)
   }
```

4.4.2.2 Data Block Transformation

The data block transformation step is in the first place meant to make changes inside the selected RDF graphs describing a data block. For instance, the value

Listing 4.2: Format-independent SPARQL query to obtain data blocks belonging to the first half of a specific soccer match.

```
PREFIX mmo: <multimedia_model.owl#>
PREFIX so: <soccer.owl#>
CONSTRUCT {
  ?db rdf:type ?types.
  ?db mmo:start ?start.
  ?db mmo:length ?length.
  # ...
}
WHERE {
  ?bitstream rdf:type mmo:MediaBitstream.
  ?bitstream rdfs:label 'soccer_avc'.
  ?annoMM mmo:isRepresentedBy ?bitstream.
  ?annoMM rdf:type mmo:AnnotatedMultimedia.
  ?annoMM mmo:hasTemporalSegment ?segment.
  ?segment foaf:depicts ?half.
  ?half rdf:type so:Half.
  ?half so:number 1^^xsd:integer.
  ?segment mmo:hasBitstreamData ?rau.
  ?bitstream mmo:hasStructure ?rau.
  ?rau mmo:hasStructure ?db.
  ?db rdf:type ?types.
  ?db mmo:start ?start.
  ?db mmo:length ?length.
  # ...
}
```

of a *SyntaxElement* can be changed or the length of a *TruncatablePayload* can be shortened. In practice, data block transformation is needed when a particular coding format requires, next to the selection of datablocks, certain syntax element modifications in order to deliver a compliant adapted bitstream. An example of such a coding format is Scalable Video Coding (SVC). For instance, the following approach needs to be followed to adapt the SVC syntax element *num_layers_minus1*, which denotes the amount of scalability layers available in an SVC bitstream. The structural part of the multimedia model needs to be extended with a coding-format specific syntax element as illustrated in Figure 4.16. In this figure, the class *Num_layers_minus1* is created as a subclass of *SyntaxElement*. When a data block of type *Num_layers_minus1* is detected during the data block transformation step, the value of this syntax element is changed according to the requested scalability properties.

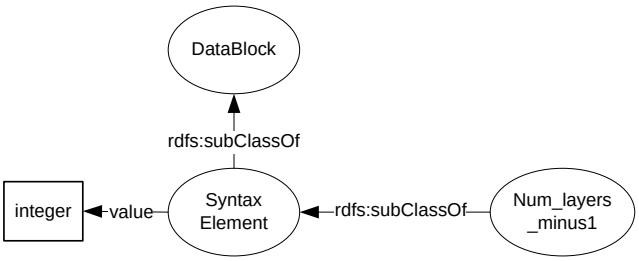


Figure 4.16: Extending the multimedia model for the scalability information located in an SVC bitstream.

A second use case for data block transformation is the support for dynamic adaptations, i.e., when the multimedia content is delivered during varying usage environment conditions. In order to avoid the initialization and evaluation of a new query each time a structural adaptation property (e.g., amount of temporal layers) changes, the structural selection is omitted during the data block selection step. More specifically, the content selection is established during the data block selection step and the structural selection is performed during the data block transformation step. For example, during data block selection, all data blocks can be selected belonging to the first half of a soccer match; during data block transformation, the frame rate of the video fragment can be scaled to 15 fps by dropping the necessary temporal scalability layers. Hence, when structural adaptation parameters change during the adaptation process, the query does not need to be re-executed.

Currently, no standardized solution exists to transform RDF graphs. Next to ontology-specific software (i.e., write an own RDF transformer based on a specific ontology), the following solutions are available.

- *XML transformation technologies*: XPath and XSLT can be used to access and transform an XML serialization of RDF data. However, the main problem with this approach is that the standard RDF/XML serialization is non-deterministic (i.e., there are many possible serializations for a given RDF model). Furthermore, XPath expressions are not aware of the semantics of the RDF model. Several approaches such as Twig [141] and RxPath [114] define a set of XPath/XSLT extension functions and/or provide a mapping of RDF to the XPath data model, in order to cope with the RDF/XML serialization problem.

- *RDF transformation technologies*: the non-deterministic character of RDF/XML serialization is caused by the fact that XML is tree-based

while RDF is graph-based. Converting a graph-based model such as RDF to a tree-based model such as XML is not trivial. Therefore, a graph-oriented RDF access mechanism is needed. RDF Path [93] is one example of an attempt to come to an RDF Path language. It is triple oriented, tries as far as possible to mimic XPath, and treats a graph as an extended tree with no root.

- *SPARQL/Update*: in addition to SPARQL, which provides a retrieval language for RDF, SPARQL/Update (a.k.a. SPARUL, [108]) is proposed as an update language for RDF graphs. It is a language to express updates to an RDF store and is intended to be a standard mechanism by which updates to a (remote) RDF Store can be described, communicated, and stored.

4.4.2.3 Data Block Binarization

The final step in the workflow of model-driven content adaptation is the binarization of the data blocks. More specifically, based on the original media bitstream (present in the file server as shown in Figure 4.14) and the selected and adapted data block graphs, an adapted media bitstream is created. The *start* and *length* properties of the data block are used to copy a part of the original bitstream into the adapted bitstream. If the data block is a *SyntaxElement*, the syntax element value is written to the adapted bitstream. In case the data block has as type *StuffingBits*, stuffing bits are added to the adapted bitstream until it is byte-aligned.

4.5 Model-driven Content Adaptation vs. Other Techniques

In this section, a comparison is made between the proposed model-driven content adaptation technique and other techniques for multimedia content adaptation, i.e., XML-driven content adaptation and dedicated software approaches. The comparison is based on a number of criteria that are listed below.

Criterion 1: *Format-independency of the software*. The software modules used between the original compressed bitstream and the adapted compressed bitstream are investigated in terms of their independency of underlying coding and/or metadata formats. The more software modules are format-agnostic in the adaptation chain, the more extensible the adaptation framework and the better the support for new formats will be.

Criterion 2: *Knowledge needed for adaptation operations*. This criterion examines the specific knowledge needed to define adaptations. More specifically, to what extent is knowledge needed regarding a specific coding format in order to define a specific adaptation. Within format-independent content adaptation systems, it is important to avoid coding-format specific calculations when defining adaptation operations. Otherwise, format-independency is obtained in an ad-hoc way, as discussed in Section 4.2.

Criterion 3: *Content metadata integration*. In order to perform adaptations based on content metadata (e.g., indication of violent video scenes), a straightforward integration between the adaptation logic and the content metadata is desired.

Criterion 4: *Adaptation possibilities*. The different kinds of adaptations that are possible with a specific content adaptation technique are examined in this criterion. Furthermore, prerequisites of the compressed bitstreams are investigated in order to enable certain kinds of adaptations.

Criterion 5: *Performance*. The performance of adaptation techniques is measured in terms of execution speed, memory consumption, and metadata overhead in terms of disk usage.

The first four criteria are discussed below, while the performance measurements are provided in Section 4.6. A summarizing table is provided in Section 4.7. The first four criteria are applied to model-driven content adaptation, XML-driven content adaptation, and dedicated software approaches.

Criterion 1: Format-independency of the software

It is clear that dedicated software approaches do not provide format-agnostic software modules. Hence, support for new coding and/or metadata formats requires the development of new software modules.

Both XML- and model-driven content adaptation are able to deliver format-agnostic software modules for the content adaptation chain. The transformation and binarization steps are driven by coding-format and metadata agnostic software modules (i.e., modules for transformation and binarization). The generation of structural metadata can also be performed by using format-agnostic software modules; BSDL even provides a standardized solution taking the form of the BintoBSD parser.

Criterion 2: Knowledge needed for adaptation operations

Developing dedicated software for particular multimedia content adaptation operations requires knowledge of the high-level structure of the coding formats that will be supported by the dedicated software (to be able to parse the compressed bitstream), knowledge of the supported content metadata formats, and the coupling (i.e., the mapping) of the structure of the compressed bitstream and the content metadata.

XML-driven content adaptation requires the same knowledge as the dedicated software approach. The technique claims to rely on high-level descriptions of the media bitstreams. However, as discussed in Section 4.2, these descriptions are just a textual representation of the coding-format specific structures and syntax elements. Coding-format specific algorithms to implement coding-format independent adaptation operations such as 'lower the frame rate' have to be used within the BSD transformation step. Therefore, an XML filter is dependent on the coding format and the metadata formats used. Furthermore, the mapping between the structure of the compressed bitstream and the content metadata is also defined inside the XML filter. In comparison to the dedicated software approach, XML-driven content adaptation requires less implementation effort since I/O operations are abstracted by the format-agnostic software modules.

Model-driven content adaptation also requires knowledge regarding the high-level structure of the coding formats, metadata formats, and mapping between structural and content metadata; however, this knowledge is obtained during the metadata generation and is separated from the actual adaptation operations. Indeed, the high-level structure of coding formats is mapped to the structural multimedia model (see Section 4.3.1.1) and is independent of possible adaptations. The same holds true for the content metadata model (see Section 4.3.1.4), which contains a description of the multimedia content. Hence, defining adaptations within an model-driven content adaptation system is purely based on the multimedia model.

Criterion 3: Content metadata integration

As discussed in Section 4.2, integrating content and structural metadata is obtained in an ad-hoc way in case of XML-driven content adaptation. This is due to the lack of semantic interoperability of XML, resulting in a metadata-format dependency of the XML filters. Furthermore, coding-format specific mappings between structural and content metadata need to be created during the adaptation process. However, as discussed in Chapter 3, markers containing content metadata can be inserted in gBSDs. Hence, no coding-format specific map-

pings need to be made during the adaptation process. The problem with these markers (and XML-based metadata in general) is a lack of a formal description of their semantic meaning. Therefore, even XML filters for gBSDs containing markers are dependent on the metadata format used. The dedicated software approach is similar to XML-driven content adaptation, since the content metadata needs to be mapped to the coding-format specific bitstream structures.

By the definition of a multimedia model, model-driven content adaptation provides a seamless integration of structural and content metadata. More specifically, the structural part of the model can be seen as a layer on top of media bitstreams, providing support for easy access to the high-level structures and syntax elements of the media resource. Based on a subset of the structural metadata (i.e., a subset of the data blocks), an adapted media resource is generated. Because we are working with an additional layer on top of the media resource (i.e., the structural metadata), it is possible to link these metadata to content descriptions of the media resource (i.e., the content metadata). Moreover, Semantic Web technologies, which solve the semantic interoperability problem of XML, are inherently present since the multimedia model is implemented by making use of Semantic Web technologies.

Criterion 4: Adaptation possibilities

All adaptation operations are possible by using dedicated software. Both semantic and structural adaptations are possible and do not require any prerequisites of the compressed bitstream, except that the bitstream needs to be encoded in a coding format that is supported by the software. In particular cases, the compressed bitstream can be completely decoded and re-encoded in order to perform the necessary adaptations.

XML-driven content adaptation is mainly focussed on the adaptation of scalable bitstreams as discussed in Section 2.3. Hence, when structural adaptations need to be established, the compressed bitstreams need to be encoded in such a way that it is possible to perform the adaptations without the need of a complete recode process. For example, efficiently exploiting temporal scalability using the H.264/AVC coding format requires the presence of a hierarchical coding structure [30]. The same observation can be made for semantic adaptations: the encoded bitstreams need to provide several random access points in order to extract specific segments of the bitstream.

The adaptation possibilities of model-driven content adaptations are equal to the ones of XML-driven content adaptation. They both operate on compressed bitstreams and do not perform any decoding and/or re-encoding of the bitstream.

Table 4.1: Overview of the bitstream characteristics.

Coding format	Size (MB)	Bit rate (kbit/s)	Parse unit	# Parse units	# Temp. levels	# Spat. levels	# Qual. levels	XML descr. size (MB)	# RDF triples	RDF gen. (s)
H.264/AVC	35.9	2942	NALU	5004	3	1	1	4.4	47330	54.6
SVC	40.8	3343	NALU	15008	3	2	4	18.5	147506	282.5
H.263+	45.2	3705	Picture	5000	3	2	1	7.7	54492	121.6
MPEG-2 Video	77.3	6332	Picture	2500	3	1	1	9.6	29830	67.6
AAC	1.5	128	Frame	4691	-	-	-	3.0	60729	54.8
MP3	2.3	192	Frame	4170	-	-	-	1.9	54136	50.9

4.6 Performance Measurements

In this section, the fifth criterion is discussed in the context of XML- and model-driven content adaptation. We do not discuss the dedicated software approach since it is clear that the performance of dedicated software in terms of execution speed and memory consumption will generally be better or equal to the performance of format-independent approaches such as XML- and model-driven content adaptation. Also, no overhead in terms of disk usage is present in case of dedicated software solutions since no structural metadata is present.

We assume that the metadata generation step is done in advance, i.e., the XML descriptions and RDF triples are present. Previous work has shown that the generation of XML descriptions of the high-level structure of a media bitstream can be performed in real time by making use of BFlavor [31], which is able to automatically generate a coding-format specific parser producing XML descriptions compliant to BSDL. In our research, the same approach was followed in the context of model-driven content adaptation, i.e., automatically generated parsers can produce RDF triples compliant with the multimedia model presented in Section 4.3.1. The content metadata was obtained by manually annotating the multimedia content.

4.6.1 Application Scenario

In order to evaluate and compare XML- and model-driven content adaptation, the following application scenario is used. A video fragment of a part of a soccer game is present as test sequence, together with the appropriate audio fragment. Furthermore, information regarding the level of importance of specific scenes is present. Each scene was annotated by a number equal to 0 (everything except game interruptions), 1 (goals, chances, and faults), or 2 (only goals).

The video fragment is encoded using four different video codecs (i.e., H.264/AVC, the scalable extension of H.264/AVC (SVC), H.263+, and MPEG-2 Video); the appropriate audio fragment is encoded with two audio codecs (i.e., Advanced Audio Coding (AAC) and MPEG-1 Audio Layer 3 (MP3)). The user selects specific parts of the test sequence based on the level of importance (i.e., semantic adaptation). Furthermore, exploitation of scalability properties of the encoded bitstreams is performed (i.e., structural adaptation). The latter can vary dynamically during the adaptation, implying that the adaptation parameters regarding the scalability properties need to be adjustable in an on-the-fly fashion.

4.6.2 Experimental Results

4.6.2.1 Bitstream Characteristics

The video fragment contains a length of 100 s, a frame rate of 25 fps, and a resolution of 720x576 pixels. Selecting scenes with a level of importance of 0, 1, or 2 results in an adapted bitstream of 2484, 1503, or 855 frames respectivily. An overview of the properties of the encoded bitstreams can be found in Table 4.1. The sizes of the XML descriptions and the number of RDF triples corresponding to the structural metadata[8] are also shown in this table. Note that the overhead of format-agnostic content adaptation (i.e., the XML descriptions and RDF triples) is dependent on the number of parse units present in the bitstream. The number of parse units will also have an impact on the execution times regarding the adaptation of the bitstreams (see Section 4.6.2.3). However, it is clear that both XML- and model-driven content adaptation introduce a significant amount of overhead in terms of disk usage due to the occurrence of the structural metadata.

The last column of Table 4.1 shows the execution times for the generation of RDF triples describing structural metadata and scalability information. These RDF triples can be generated in the same way as XML descriptions suitable for XML-driven content adaptation. More specifically, a modified version of BFlavor [31] was used to automatically create format-specific parsers that are capable of outputting RDF triples (in its original form, BFlavor was only able to automatically create format-specific parsers producing XML descriptions compliant to the BSDL standard). As can be seen in Table 4.1, the execution times depend on the number of parse units and the bit rate. Note that each parse unit corresponds to an RDF data block graph. SVC and H.263+ perform less than real-time as the length of the test sequences used is 100 seconds. This is due to the use of multiple scalability layers (resulting in a higher amount of parse units) and a higher bit rate.

4.6.2.2 Implementation Details

In our experiments, BSDL and STX[9] were used to perform XML-driven content adaptation. Bitstream Syntax Schemas (BS Schemas) for the six codecs were designed, together with ten STX stylesheets (six stylesheets to implement

[8] The number of RDF triples to represent the scalability information and content metadata is in this case negligible in comparison with the number of RDF triples needed for the structural metadata. Note that in our configuration, one RDF triple took approximately 90 bytes of disk usage.

[9] Version 1.3.1 of the BSDL reference software and version 2008-03-09 of the STX engine Joost were used.

the scene extraction (one for each codec) and four stylesheets to implement the exploitation of scalability (one for each video codec)). The STX engine transforms the XML descriptions based on a STX stylesheet. The BSDtoBin parser is a standardized parser from BSDL taking as input the transformed XML description and the original bitstream and producing the adapted bitstream. The STX engine and the BSDtoBin parser are connected through SAX events. More specifically, no intermediate (transformed) XML description is generated, since the transformed SAX events are immediately transferred to the BSDtoBin parser. This will significantly speed up the XML-driven adaptation process because I/O operations regarding intermediate XML descriptions are avoided.

The data block selection, transformation, and binarization modules of our model-driven content adaptation framework are built on top of Sesame[10] (version 2.0.1), which is an open source RDF database with support for RDF Schema inferencing and querying. The built-in SPARQL engine of Sesame is used for the evaluation of queries during the data block selection step. The native store facilities of Sesame were used for our RDF repository, implying that the RDF data is retrieved directly from disk. Using a native store provides better scalability since it is independent of the available system memory. However, when generic RDF storage solutions such as the native store facilities of Sesame become insufficient in terms of scalability due to a high number of RDF triples (i.e., the structural metadata), other solutions should be considered to store the structural metadata.

For example, one solution for this problem is to store the structural metadata and scalability information in an RDF store which is specifically designed for the model for media bitstreams. More specifically, the structural metadata and scalability information can be stored in a highly scalable Relational Database Management System (RDBMS), using a database scheme based on the structural and scalability part of the model for media bitstreams. Hence, such a RDBMS can be seen as an efficient RDF store specifically designed for our model for media bitstreams. This way, SPARQL queries can be translated into SQL queries. The results of these SQL queries can then be converted back to RDF graphs corresponding to the selected data blocks. RDBMSs should be capable of dealing with a large amount of structural metadata since they are mature, stable, and scalable, while also providing a high performance in terms of query execution speed.

In contrast to the combination of BSDL and STX where a lot of BS Schemas and STX stylesheets need to be developed, only one SPARQL state-

[10] Available on `http://www.openrdf.org/`.

ment and a domain-specific ontology[11] was needed for model-driven content adaptation. In order to build this SPARQL statement, knowledge of the multimedia model is needed together with the concepts of the domain-specific ontology which is coupled with the content metadata model. Note that the exploitation of scalability was included during the data block selection step (i.e., scalability options are static during the adaptation). Additionally, an implementation where the exploitation of scalability was performed during the data block transformation step (i.e. scalability options can dynamically vary during the adaptation) was used to compare the static and dynamic exploitation of scalability.

Performance measurements were done on a PC having an Intel Pentium D 2,8 GHz CPU and 1 GB of system memory at its disposal. The operating system used was Windows XP Pro SP2, running Java 2 Runtime Environment (SE version 1.5.0_09). The memory consumption of the Java programs was measured by relying on JProfiler 4.2.1. All time measurements were executed six times, whereupon an average was calculated over the last five runs to avoid startup effects (the standard deviation was 0.05 s).

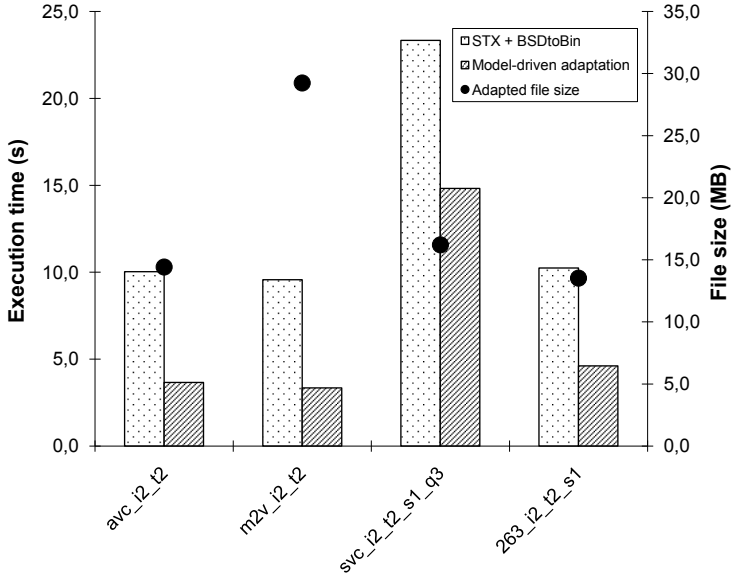


Figure 4.17: Execution times and file sizes for the video coding formats.

[11] In our case, a simple soccer ontology was used, including concepts such as *SoccerMatch* and *SoccerScene*, as well as properties such as *levelOfImportance*.

Table 4.2: Overview of the execution times and resulting file sizes.

Sequence	STX + BSDtoBin (s)	Model-driven adaptation (s)	Adapted file size (MB)
avc_i0_t2	18.0	9.3	35.5
avc_i0_t0	12.8	7.7	20.3
avc_i1_t2	13.2	5.9	22.8
avc_i1_t0	9.9	4.9	12.6
avc_i2_t2	10.0	3.7	14.4
avc_i2_t0	8.1	3.1	7.8
svc_i0_t2_s1_q3	47.1	42.3	40.4
svc_i0_t2_s1_q0	36.7	31.1	38.7
svc_i0_t2_s0_q0	28.9	27.1	3.7
svc_i0_t0_s1_q3	31.7	39.0	23.3
svc_i0_t0_s1_q0	28.9	28.9	22.8
svc_i0_t0_s0_q0	25.5	26.2	2.6
svc_i1_t2_s1_q3	32.7	25.7	25.8
svc_i1_t2_s1_q0	26.5	18.9	24.8
svc_i1_t2_s0_q0	21.6	16.7	2.3
svc_i1_t0_s1_q3	23.4	23.5	14.5
svc_i1_t0_s1_q0	21.7	17.6	14.1
svc_i1_t0_s0_q0	19.6	16.1	1.6
svc_i2_t2_s1_q3	23.3	14.8	16.2
svc_i2_t2_s1_q0	19.8	11.0	15.6
svc_i2_t2_s0_q0	16.8	9.7	1.4
svc_i2_t0_s1_q3	17.9	13.6	8.9
svc_i2_t0_s1_q0	16.9	10.9	8.7
svc_i2_t0_s0_q0	15.6	9.3	1.0
m2v_i0_t2	23.1	7.8	76.6
m2v_i0_t0	13.5	5.3	25.0
m2v_i1_t2	14.7	5.3	48.4
m2v_i1_t0	8.8	3.6	15.6
m2v_i2_t2	9.6	3.3	29.2
m2v_i2_t0	6.1	2.4	9.4
263_i0_t2_s1	20.2	12.4	44.8
263_i0_t2_s0	13.9	8.6	9.6
263_i0_t0_s1	13.6	10.7	22.8
263_i0_t0_s0	11.9	8.1	4.9
263_i1_t2_s1	14.1	7.6	25.1
263_i1_t2_s0	10.5	5.5	5.4
263_i1_t0_s1	10.3	6.8	12.6
263_i1_t0_s0	9.4	5.2	2.7
263_i2_t2_s1	10.2	4.6	13.5
263_i2_t2_s0	8.2	3.4	2.9
263_i2_t0_s1	8.1	4.1	6.7
263_i2_t0_s0	7.6	3.3	1.4
aac_i0	4.5	3.9	1.5
aac_i1	3.4	2.6	0.9
aac_i2	2.7	1.8	0.5
mp3_i0	3.9	3.6	2.3
mp3_i1	3.0	2.5	1.4
mp3_i2	2.4	1.4	0.6

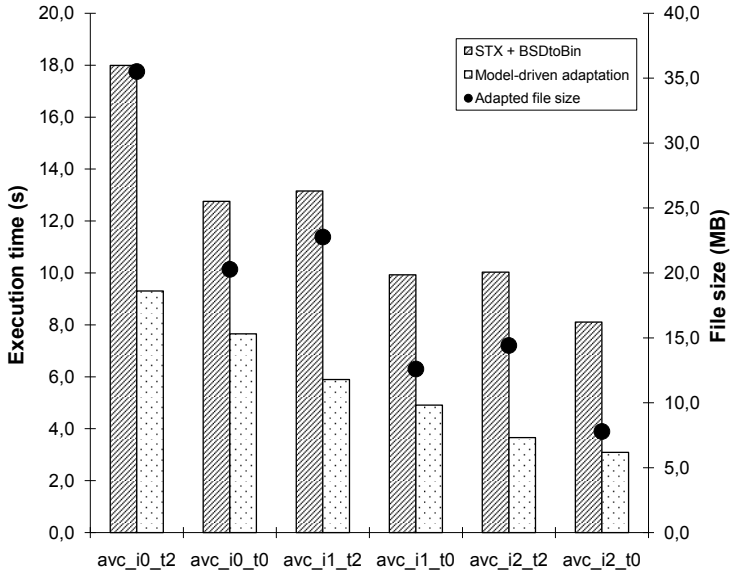


Figure 4.18: Execution times for the H.264/AVC coding format.

4.6.2.3 Results

An overview of the execution times and resulting file sizes of the adapted bitstreams is given in Table 4.2. The names of the resulting sequences correspond to `CCC_iI_tT_sS_qQ`, with `CCC` equal to the coding format; `I` corresponds to the level of importance; and `T`, `S`, and `Q` respectively correspond to the number of temporal, spatial, and quality scalability layers present in the adapted bitstream.

In general, both XML- and model-driven content adaptation perform well and are able to adapt bitstreams of various coding formats in real time. As can be seen from Table 4.2, the coding format (more specifically the number of parse units per frame and the bit rate) has a significant impact on the execution times. For instance, in case of model-driven content adaptation, the higher the number of parse units per frame, the higher the number of data blocks that need to be selected, optionally transformed, and serialized. Furthermore, a high bit rate of the adapted bitstream results in a high number of I/O operations during the binarization of the data blocks. The same observations can be made for XML-driven content adaptation. In Figure 4.17, this is illustrated by plotting the execution times together with the adapted file sizes for the video coding formats.

In most of the cases, model-driven content adaptation has lower execution

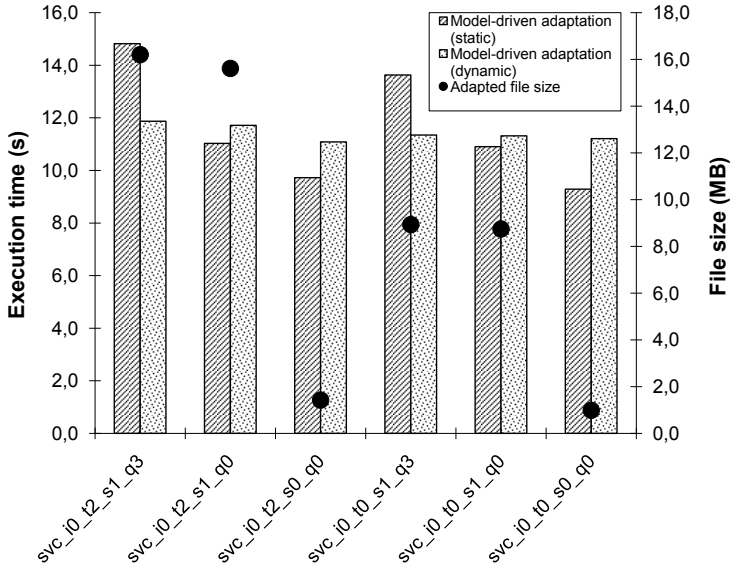


Figure 4.19: Execution times for the SVC coding format, comparing static versus dynamic exploitation of scalability.

times than XML-driven content adaptation, but both techniques have a comparable performance. This is for instance illustrated in Figure 4.18, where the execution times for the H.264/AVC coding format are plotted. There are two reasons why model-driven content adaptation performs slightly better than XML-driven content adaptation. First, coding-format specific algorithms need to be executed during the transformation step in case of XML-driven content adaptation. As discussed in Section 4.3, these coding-format specific algorithms are already executed during the structural metadata generation step in case of model-driven content adaptation. Second, descriptions compliant to the model for media bitstreams only contain information that is really necessary for the adaptation operation. On the contrary, XML-driven content adaptation needs to process BSDs containing coding-format specific structures and syntax elements necessary to execute these coding-format specific algorithms. Hence, the processing of these BSDs will take longer than the processing of a description compliant with the model for media bitstreams. Further, both XML- and model-driven content adaptation have a low and constant memory consumption (approximately 2 MB).

A comparison between static and dynamic exploitation of scalability is shown in Figure 4.19. In case of static adaptation, the proper data blocks are already present after the data block selection step. On the contrary, dynamic

Table 4.3: Overview of the discussed criteria.

Criterion	Dedicated software	XML-driven adaptation	Model-driven adaptation
Format-independency	No	Yes	Yes
Knowledge needed for adaptation	High	High	Low
Content metadata integration	Low	Low	High
Adaptation possibilities	All	Coding-format dependent	Coding-format dependent
Execution speed	Real time	Real time	Real time
Memory consumption	Low	Low	Low
Metadata overhead	None	Structural metadata	Structural metadata

adaptation implies that during the data block selection step, only a selection is made based on content preferences (and not regarding scalability options). Hence, more data blocks are selected during the data block selection step. The actual structural adaptation is then performed during the data block transformation step. This declares why the execution times for the dynamic adaptation process are almost constant (between 11 and 12 s in Figure 4.19), since the same amount of data blocks are selected, regardless of the scalability preferences. The little variations in execution times are due to the varying file sizes of the adapted bitstreams.

4.7 Conclusions and Original Contributions

In this chapter, we have introduced the concept of model-driven content adaptation, a novel multimedia content adaptation technique that operates independently of coding and metadata formats. It is based on the definition of a multimedia model describing and coupling structural metadata, content metadata,

and scalability information. Existing coding and metadata formats are mapped to this multimedia model. An implementation of the model is realized using Semantic Web technologies such as RDF, OWL, and SPARQL. The adaptation of media bitstreams is performed by selection, transformation, and serialization of the structural metadata, based on the content metadata and scalability information.

We have compared model-driven content adaptation with dedicated multimedia content adaptation software and XML-driven content adaptation (discussed in Chapter 2). A summary of the evaluation of different criteria is provided in Table 4.3. The most significant advantages of model-driven content adaptation are the low amount of knowledge needed to define adaptation operations and the seamless integration with content metadata. Furthermore, performance results have shown that model-driven content adaptation is characterized by real-time execution times and a low and constant memory consumption.

The research that has led to this chapter is also described in the following publications.

1. S. De Bruyne, D. De Schrijver, W. De Neve, D. Van Deursen, and R. Van de Walle. Enhanced Shot-Based Video Adaptation using MPEG-21 generic Bitstream Syntax Schema. In *Proceedings of the 2007 IEEE Symposium Series on Computational Intelligence*, 6 pages on CD-ROM, April 2007, Honolulu, Hawaii

2. D. Van Deursen, D. De Schrijver, S. De Bruyne, and R. Van de Walle. Fully Format Agnostic Media Resource Adaptation Using an Abstract Model for Scalable Bitstreams. In *Proceedings of the 2007 IEEE International Conference on Multimedia and Expo*, pages 240–243, July 2007, Beijing, China

3. S. De Bruyne, D. Van Deursen, J. De Cock, W. De Neve, P. Lambert, and R. Van de Walle. A Compressed-domain Approach for Shot Boundary Detection on H.264/AVC Bit Streams. *Signal Processing: Image Communication – Special Issue on Semantic Analysis for Interactive Multimedia Services*, 23(7):473–498, August 2008

4. D. Van Deursen, C. Poppe, G. Martens, E. Mannens, and R. Van de Walle. XML to RDF Conversion: a Generic Approach. In *Proceedings of the 4th International Conference on Automated Production of Cross Media Content for Multi-channel Distribution*, pages 138–143, November 2008, Florence, Italy

5. D. Van Deursen, W. Van Lancker, S. De Bruyne, W. De Neve, E. Mannens, and R. Van de Walle. Format-independent and Metadata-driven Media Resource Adaptation using Semantic Web Technologies. Submitted to *Multimedia Systems Journal*

Chapter 5

Fully integrated multimedia delivery platforms

One man alone can be pretty dumb sometimes, but for real bona fide stupidity, there ain't nothin' can beat teamwork.

Edward Abbey (1927 - 1989)

5.1 Introduction

As already mentioned in this dissertation, recent years have witnessed an increasing heterogeneity in the multimedia landscape on different fronts such as end-user devices, network technologies, and coding formats. Next to coding formats, there also exists a wide variety of delivery formats, i.e., formats encapsulating encoded media bitstreams (e.g., the MP4 file format [59] or the Real-time Transport Protocol (RTP, [105])). Finally, end-users with specific preferences often want to obtain a personalized version of multimedia content (e.g., an end-user only requesting scenes satisfying his/her interests).

As discussed in the previous chapters, format-independent adaptation techniques can be used to tackle the above described multimedia heterogeneity and to obtain Universal Multimedia Access (UMA, [136]). In order to investigate the feasibility of these techniques in practice, a fully integrated adaptation platform based on format-independent adaptation techniques needs to be designed. The platform must be able to support both structural and semantic adaptation operations. Further, it needs to be deployable in streaming environments, requiring real-time processing of the media bitstream using format-independent adaptation techniques.

In this chapter, we address the design and functioning of two fully integrated platforms for multimedia adaptation and delivery. The implementation of both platforms was largely performed by Wim Van Lancker. In Section 5.2, the first platform, which is called MuMiVA, is introduced. It relies on two standardized, XML-driven content adaptation tools, i.e., MPEG-B BSDL and MPEG-21 gBS Schema (see Chapter 2) and is able to deliver multimedia content using the RTP/RTSP protocol. Further, in Section 5.3, we introduce NinSuna, our second multimedia adaptation and delivery platform which solves a number of problems of the MuMiVA platform, such as the problems with XML-driven content adaptation and the lack of other delivery formats (next to RTP/RTSP). Therefore, the adaptation engines used within the NinSuna platform rely on model-driven content adaptation, which was discussed in Chapter 4. Furthermore, NinSuna also provides support for the packaging of adapted multimedia content in a format-independent way. The latter allows NinSuna to deliver the multimedia content using various delivery formats. In Section 5.4, synchronization issues within our platforms are discussed which are caused by semantic adaptations of synchronized, compressed media bitstreams. Finally, related work and conclusions are discussed in Section 5.5 and Section 5.6, respectively.

5.2 MuMiVA

In this section, we address the design and performance evaluation of a multimedia delivery platform that relies on XML-driven adaptation engines. Our platform is called MuMiVA[1]; it is a fully integrated, extensible platform for multimedia delivery in heterogeneous usage environments, using streaming technologies [125]. Thanks to the use of XML-driven adaptation technology, format-independent adaptation logic can be used to adapt (scalable) media resources. To demonstrate the flexibility of our multimedia delivery platform, we discuss the functioning of two different applications (i.e., exploitation of temporal scalability and shot selection) applied to two different coding formats (i.e., MPEG-4 Visual and H.264/AVC).

5.2.1 MuMiVA Architecture

As discussed above, MuMiVA is a multimedia delivery platform that is able to transparently deliver multimedia content for heterogeneous usage environments. Moreover, this transparent approach reflects both a format-agnostic

[1] MuMiVA is short for "Mutare, Mittere, Videre, Audire", which is Latin for "to adapt, to send, to watch, to hear".

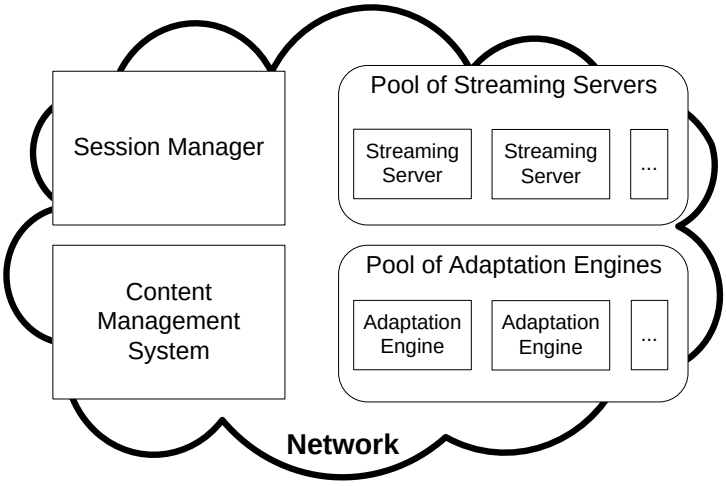


Figure 5.1: A global view on MuMiVA.

(i.e., independent of the underlying coding format) and application-agnostic characteristic (i.e., independent of the adaptations applied to the media resources). In this section, an overview is given of the MuMiVA architecture, together with its strengths.

5.2.1.1 Distributed Architecture: a Global View on MuMiVA

Our multimedia delivery platform contains multiple components that can be distributed across a managed network. Distributing the components of MuMiVA across the network makes it possible to achieve a scalable multimedia delivery platform. More specifically, a distributed multimedia system makes it easy to expand or contract its pool of servers to accommodate increasing or decreasing loads on the platform. Figure 5.1 gives an overview of the components present in our MuMiVA platform. Explanatory notes for this figure are given below.

- *Content Management System (CMS)* is a multimedia archive that contains multimedia content encoded with (scalable) coding formats, as well as metadata about the content. Both structural (e.g., a BSD of a particular bitstream) and content metadata (e.g., scene information for a particular video sequence) are present in the CMS.
- *Pool of Adaptation Engines* is a collection of distributed adaptation engines, which allows to select adaptation engines based on the current

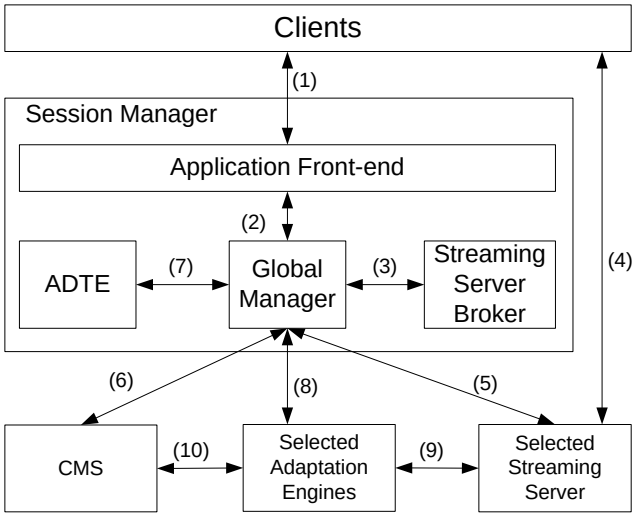


Figure 5.2: Functioning of MuMiVA.

adaptation engine load. An adaptation engine may deploy multiple adaptation techniques. The current implementation only supports format-agnostic adaptation techniques, i.e., BSDL and gBS Schema. However, MuMiVA allows the adoption of other (format-specific) adaptation tools. Examples of such adaptation tools are bitstream extractors and transcoders.

- *Pool of Streaming Servers* is a collection of distributed streaming servers, which allows to select the proper streaming server based on an assessment of the current server load.

- *Session Manager* couples the CMS, adaptation engines, and streaming servers. It provides an access point for clients to the multimedia delivery platform by means of an application front-end (e.g., web service). Furthermore, this component selects the proper streaming server, manages the different client sessions, and is able to take decisions regarding the adaptation of the requested content, based on information about the usage environment of the clients.

5.2.1.2 Functioning of MuMiVA

The MuMiVA architecture is shown in Figure 5.2, as well as the communication between its different components. Explanatory notes for the interactions

between the different components of the MuMiVA platform are provided below.

(1) A client requests multimedia content by contacting the MuMiVA platform. Besides the content request, a client also sends information about its usage environment.

(2) The application front-end provides the information about the client to the global manager. The manager initializes a new session for each particular client. Consequently, the client receives a session ID.

(3) The global manager contacts the broker for the streaming server in order to select a proper streaming server based on the current server load. The selected streaming server will be announced to the client by the global manager through the application front-end.

(4) The client connects to the selected streaming server, and provides its session ID in order to receive the desired content.

(5) The selected streaming server contacts the global manager, in order to receive information about the existing request of the client, based on its session ID.

(6) The global manager fetches metadata about the requested content (e.g., bit rate and resolution), which are stored in the CMS.

(7) Once the global manager has all the necessary information at its disposal (i.e., metadata about the requested content and information about the usage environment), it contacts the Adaptation Decision Taking Engine (ADTE). The ADTE first decides which adaptation technique to use, based on the available adaptation engines. Once an adaptation technique is chosen (e.g., gBS Schema), the ADTE might take additional decisions related to the adaptation technique (e.g., the XML transformation tool in case of an XML-driven adaptation technique). Next, it calculates the adaptation parameters by matching the metadata about the requested content and the information about the usage environment (e.g., comparison of the resolution of a video sequence with the size of the screen of the end-user device). When the requested content includes both audio and video, the ADTE will select two adaptation engines: one for the video stream and one for the audio stream. In the latter case, the synchronization between the output of the adaptation engines is done by the streaming server.

(8) The global manager initializes the selected adaptation engines with the collected adaptation parameters.

Once the necessary negotiation between the client and MuMiVA is done, the client can start consuming the requested multimedia content. The streaming server, adaptation engines, and CMS form a pipeline system, where the communication is based on a pull-system. More specifically, the streaming server pulls content from the adaptation engine, which subsequently pulls content from the CMS. The following steps are performed during the streaming of the multimedia content to the client.

(9) The streaming server reads the adapted media resources from the adaptation engines. Subsequently, these adapted resources are sent to the client by using the Real-time Transport Protocol (RTP) [105]. The Real Time Streaming Protocol (RTSP) [106] is used for the exchange of control operations between client and streaming server (e.g., the client can decide to pause the streaming of the content).

(10) The adaptation engines read the original media resources from the CMS. Furthermore, an adaptation engine customizes a given media resource according to the adaptation parameters received from the global manager. More detailed information about the internal working of an adaptation engine within MuMiVA is provided in Section 5.2.1.3.

5.2.1.3 XML-driven Adaptation Engine

One of the main features of our MuMiVA platform is the use of XML-driven content adaptation engines, based on MPEG-B BSDL and MPEG-21 gBS Schema. As discussed in Chapter 2, XML-driven content adaptation consists of three steps: BSD generation, BSD transformation, and adapted bitstream generation. Within the MuMiVA platform, BSD generation is seen as a preprocessing step. Consequently, the BSDs of the corresponding media resources are already available in the CMS. Hence, the adaptation engines only need to perform the last two steps of the XML-driven content adaptation chain, i.e., BSD transformation and adapted bitstream generation.

As discussed in Chapter 2, different XML transformation tools exist to transform a BSD. There are two approaches to interpret and to transform XML documents. Firstly, traditional procedural programming languages such as Java or C++, together with a parser, can be used to consume XML data. Secondly, transformations can be implemented by using stylesheets together

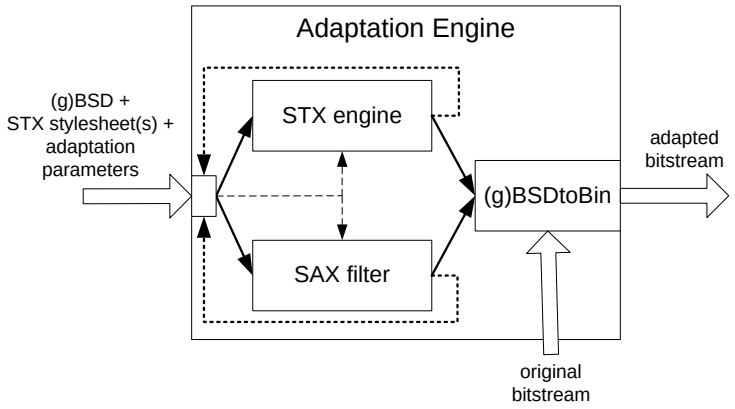


Figure 5.3: Functioning of a MuMiVA adaptation engine, based on XML-driven content adaptation. The dashed arrows denote the adaptation parameters and STX stylesheets. The bold arrows denote the multimedia data flow of the (g)BSD. The dotted arrows illustrate the possibility for multiple transformations of a (g)BSD during the adaptation.

with a generic engine for interpreting these stylesheets (e.g., XSLT and STX). The main difference between the two approaches is the possibility to make use of generic software modules (transformation engines) in the latter case. Moreover, two main types of XML parsers exist: one built on top of tree-based models (e.g., Document Object Model (DOM)) and one on event-based models (e.g., Simple API for XML (SAX)). MuMiVA only uses SAX-based XML transformation tools, since these can be used in streaming environments (in contrast to DOM-based XML transformation tools) [42]. Therefore, two different transformation tools can be used within an adaptation engine of MuMiVA: a SAX filter (i.e., a Java program for XML transformations using a SAX-based parser) and a STX engine.

The functioning of an adaptation engine is depicted in Figure 5.3. As discussed above, the adaptation engine receives adaptation parameters from the global manager. Furthermore, it reads the original bitstream from the CMS, together with its corresponding (g)BSD. When STX is used as XML transformation technology, the necessary STX stylesheets are also fetched from the CMS.

Based on the adaptation parameters, the adaptation engine selects the proper transformation tool (i.e., SAX filter or STX) and transforms the (g)BSD. Different transformations can be applied to the (g)BSD during the adaptation (see the dotted arrows in Figure 5.3). For example, the first trans-

formation may consist of temporal rescaling by using a SAX filter; the second transformation may consist of scene extraction by using a STX engine. This will be further explained in Section 5.2.2.

After the transformation of the (g)BSD, the adapted content is generated by a (g)BSDtoBin parser, taking as input the transformed (g)BSD and the original bitstream.

It is important to notice that the underlying software modules of the adaptation engine are fully format-agnostic when STX is used, since STX uses a format-agnostic engine for executing a particular stylesheet. This is in contrast with the use of a SAX filter, which does not rely on format-agnostic logic.

5.2.1.4 Strengths of the MuMiVA Platform

In the previous subsections, we have discussed the overall architecture of our multimedia delivery platform, together with the internal functioning of its different components. Delivering multimedia content by using MuMiVA has the following benefits:

- *Support of format-agnostic adaptation*: the use of XML-driven content adaptation implies that the adaptation engines can operate independently of the underlying coding format (i.e., the software modules are format-agnostic).

- *Support of application-agnostic adaptation*: the use of XML-driven content adaptation also implies that the adaptation engines support application-agnostic adaptations, i.e., the software modules are independent of the application. In this context, an application corresponds to the kind of adaptation that is executed on the media resource (e.g., temporal scalability or scene selection).

- *Extensibility*: since our multimedia delivery platform is format-agnostic and application-agnostic, it can be considered straightforward to extend MuMiVA with new coding formats and applications.

- *Support for streaming*: the MuMiVA platform offers (adapted) multimedia content in a streaming environment. As discussed above, it implements the RTSP protocol, implying that media players such as VideoLan Client (VLC [138]), Osmo [48], and QuickTime [100] can play the content streamed by the MuMiVA platform. The adaptation of the content also occurs in a streaming fashion, since the XML-driven content adaptation engine is fully SAX-based [42].

- *Interoperability*: it is important to make use of standardized and open technologies to obtain interoperability. The following technologies are used within the MuMiVA platform: MPEG-21 DIA, SAX, STX, RTSP, and RTP. Note that STX is not standardized yet; however, this technology is currently under consideration for standardization by W3C.
- *Full integration*: to the best of our knowledge, the MuMiVA platform is the first multimedia content delivery platform that offers a fully integrated solution regarding format-agnostic and application-agnostic delivery of multimedia content in a streaming environment.

To illustrate the extensibility of our platform, we will discuss the necessary steps that need to be taken to extend the MuMiVA platform with the H.264/AVC Scalable Video Coding (SVC) format. We did not yet provide support for SVC within MuMiVA due to the lack of real-time SVC decoders. A straightforward application for SVC is the exploitation of scalability along its three scalability axes (i.e., temporal, spatial, and Signal-to-Noise Ratio (SNR)) [107]. The following steps need to be taken so that MuMiVA can provide support for the delivery of SVC bitstreams, adapted according to a certain usage environment:

- The SVC-compliant bitstreams, located in the CMS, need to be accompanied by their structural metadata (i.e., XML descriptions of the high-level structure of the bitstreams). These metadata will be used to adapt the bitstreams.
- One or more STX stylesheets or SAX filters need to be written, so that the adaptation engine is able to apply the necessary XML transformations to the XML descriptions. More specifically, these transformations correspond to the exploitation of different types of scalability in SVC bitstreams.
- Finally, the streaming server needs to be extended in order to enable the streaming of SVC-compliant bitstreams.

Note that, besides SAX filters, only new software has to be written for the streaming server. All other steps have no impact on the software of the MuMiVA platform. Consequently, the streaming server is not format-agnostic.

5.2.2 MuMiVA Applications

Two applications (i.e., video frame rate reduction and shot selection), which are deployed on our MuMiVA platform, are discussed in more detail in this

section. Both applications are applied to two coding formats, i.e., MPEG-4 Visual [60] and H.264/AVC [68]. Since MuMiVA supports both coding formats, multimedia content compliant with these two coding formats is present in the CMS, together with its metadata. MPEG-21 gBS Schema is used as adaptation technology[2]. Note that multiple gBSDs can be present for one resource. Indeed, as discussed in Section 3.2, gBSDs can be application-specific due to the occurrence of markers. For simplicity, we assume that the gBSDs, present in the CMS, support both applications. More specifically, markers contain information regarding the different shots of the corresponding bitstream (see Section 5.2.2.1) and the frame rate (see Section 5.2.2.2). An example of such a gBSD is given in Listing 5.1. Note that the gBSD contains details up to frame level, which is sufficient for both applications.

5.2.2.1 Shot Selection

Shot selection enables the personalization of video content according to the preferences of a user. More specifically, a user can select individual shots out of a video sequence (e.g., goals in a soccer match). However, special attention needs to be payed to the extraction of the desired shots as the adapted bitstream needs to remain compliant with the corresponding specification. Therefore, we use the algorithm proposed in [25], where the gBSDs contain a description of the Random Access Units (RAUs). Each RAU contains a list of one or more frames, which in their turn belong to a particular shot. This is also illustrated in Listing 5.1, which contains two RAUs (lines 3 and 9).

As depicted in Listing 5.1, each *gBSDUnit* corresponding to a RAU contains a marker. This marker denotes, among other things, the shots that are located within the RAU. During the transformation, only the RAUs containing frames which belong to the requested shot are kept. Subsequently, within each selected RAU, frames which do not belong to the requested shot and which are located *after* the frames belonging to the requested shot (in decoding order), are dropped. Given the gBSD in Listing 5.1, selecting shot 4 will keep only the two RAUs depicted in this figure. Furthermore, the last frame of the second RAU will be dropped, since it does not belong to shot 4. Note that the above discussed algorithm for shot selection can be applied to both H.264/AVC and MPEG-4 Visual.

[2]Similar results would be obtained using MPEG-B BSDL.

Listing 5.1: Example of a gBSD for an H.264/AVC-encoded bitstream, containing information about the frame rate and the different shots.

```
<gBSDUnit syntacticalLabel="bitstream" start="0">
  <!-- ... -->
  <gBSDUnit syntacticalLabel="RAU" start="0" marker=":shot
      =3|shot=4">
    <gBSDUnit syntacticalLabel="FRAME" start="0" length="
        876" marker=":shot=4:fps=6"/>
    <gBSDUnit syntacticalLabel="FRAME" start="876" length="
        604" marker=":shot=4:fps=12"/>
    <gBSDUnit syntacticalLabel="FRAME" start="1480" length=
        "597" marker=":shot=3:fps=24"/>
    <gBSDUnit syntacticalLabel="FRAME" start="2077" length=
        "595" marker=":shot=4:fps=24"/>
  </gBSDUnit>
  <gBSDUnit syntacticalLabel="RAU" start="2672" marker="
      :shot=4|shot=5">
    <gBSDUnit syntacticalLabel="FRAME" start="2672" length=
        "945" marker=":shot=5:fps=6"/>
    <gBSDUnit syntacticalLabel="FRAME" start="3617" length=
        "545" marker=":shot=4:fps=12"/>
    <gBSDUnit syntacticalLabel="FRAME" start="4162" length=
        "675" marker=":shot=4:fps=24"/>
    <gBSDUnit syntacticalLabel="FRAME" start="4837" length=
        "611" marker=":shot=5:fps=24"/>
  </gBSDUnit>
  <!-- ... -->
</gBSDUnit>
```

5.2.2.2 Video Frame Rate Reduction

Reduction of the frame rate in a video sequence is obtained by dropping frames. This form of adaptation is also known as the exploitation of temporal scalability. Since the gBSDs already contain information regarding the frame rate, the transformation of the gBSDs can be considered straightforward for both H.264/AVC and MPEG-4 Visual. Indeed, as illustrated in Listing 5.1, every *gBSDUnit* corresponding to a frame contains a frame rate. When the user requests a frame rate equal to 12 fps, each frame containing a higher frame rate than 12 is dropped. This is also illustrated by the simplified STX stylesheet that is depicted in Listing 5.2.

Listing 5.2: Simplified STX stylesheet for the exploitation of temporal scalability. *gBSDUnit*s are dropped based on the frame rate located in their marker.

```
<stx:transform pass-through="all" output-method="xml">
  <stx:param name="frame_rate" select="12"/>
  <stx:template match="gBSDUnit[@syntacticalLabel='FRAME']"
      >
    <stx:if test="number(substring-after(@marker,':fps='))
       &lt;= $frame_rate">
      <stx:process-self/>
    </stx:if>
    <!-- else: drop the gBSDUnit -->
  </stx:template>
</stx:transform>
```

The logic for obtaining the mapping between the frames and their frame rate information is located in the gBSD generation step. This mapping is different for the two coding formats, i.e., MPEG-4 Visual and H.264/AVC. For the MPEG-4 Visual coding format, exploitation of temporal scalability is relatively easy since this can be done by simply dropping Bi-directional frames (B frames). This is possible since B frames are not used as reference for the coding of other frames in the video sequences.

Exploitation of temporal scalability is more complicated for H.264/AVC. This is because B slice-encoded frames can be used as reference for the encoding of other slices. As described in [30], using hierarchical coding patterns allows the dropping of frames in H.264/AVC.

5.2.2.3 Combining Shot Selection and Frame Rate Reduction

If the gBSDs support both shot selection and frame rate reduction, the two applications can be combined (i.e., selection of a particular shot, at a particular frame rate). Two approaches are possible for this combination within the MuMiVA framework. A first option is the creation of a single STX stylesheet (or SAX filter) that can deal with both applications. This stylesheet would take two parameters as input: the requested shot and the frame rate. A second option is the development of two separate STX stylesheets (and/or SAX filters), one for each application. This is possible since an adaptation engine allows multiple XML transformations during the adaptation of media resources (as discussed in Section 5.2.1.3). Note that the second approach provides more flexibility since it allows the reuse of the two transformation filters as stand-alone filters (e.g., when only shot selection is needed as an application).

5.2.3 Implementation

MuMiVA is implemented using Java's Standard Edition version 1.5. The adaptation engines rely on the reference software implementations of the BSDtoBin and gBSDtoBin parsers. The Joost[3] (version 2006-10-5) STX transformation engine is used to interpret the STX stylesheets. The streaming server is based on the C++ library of Live555 Streaming Media[4].

5.2.4 Performance Results

In order to provide an estimate of the necessary computational power for our MuMiVA platform, we have evaluated the MuMiVA applications described in Section 5.2.2, i.e., exploitation of temporal scalability, shot selection, and the combination thereof, applied to MPEG-4 Visual and H.264/AVC. These applications were implemented by means of STX stylesheets. The gBSDs describe the bitstream syntax up to the level of a frame; however, in order to estimate the impact of the level of detail of a gBSD, we have also created an additional gBSD for the H.264/AVC format which is detailed up to the RAU level only. Note that this coarse-grained gBSD is only suitable for a simplified version of the shot selection application (i.e., unnecessary frames are not removed inside the selected RAUs). Characteristics of the used test sequences (having a resolution of 1280x512) can be found in Table 5.1. Note that the gBSD for MPEG-4 Visual is more verbose than the gBSD for H.264/AVC due to the inclusion of parameters for each Video Object Sequence (VOS).

The experiments were done on a PC having an Intel Pentium D 2.8 GHz CPU and 1 GB of system memory at its disposal. The operating system used was Windows XP Pro SP2, running Sun Microsystems's Java 2 Runtime Environment (Standard Edition version 1.5). JProfiler version 4.2.1 and AQTime version 4 were used for profiling the adaptation engine and the streaming server, respectively. All time measurements were executed six times, whereupon an average was calculated over the last five runs to avoid startup effects.

Table 5.2 tabulates the CPU usage of the adaptation engine and the streaming server (both executed on a separate core), together with the file sizes of the transformed gBSDs and the resulting bitstreams. A comparison between the adaptation engine and the streaming server in terms of CPU usage reveals that the adaptation engine operates very efficiently (from 3 % to 10 % average CPU usage), while the streaming server requires two times more CPU power than the adaptation engine (average CPU usage varying from 19 % to 25 %). The latter is due to the fact that the streaming server needs to parse the incoming

[3] `http://joost.sourceforge.net/`

[4] Available on `http://www.live555.com/liveMedia/`.

Table 5.1: Bitstream and gBSD characteristics of the test sequences.

Format	Bitstream size (MB)	Frame rate (fps)	Length (s)	Bit rate (Mbit/s)	gBSD size (KB)	Coarse gBSD size (KB)
H.264/AVC	36.3	24	83.3	3.5	231	13
MPEG-4 Visual	72.1	24	83.3	6.9	392	n/a

bitstreams in order to assign proper timestamps to the RTP packets. This is in contrast with the adaptation engine, which can rely on the information located in the gBSD to perform the adaptations.

The following factors influence the CPU usage of the adaptation engine and/or the streaming server: application, coding format, and the level of detail of the gBSD (see Table 5.2). It is important to remark that the STX engine takes the most CPU usage of the adaptation engine, since it includes the transformation logic (the gBSDtoBin parser mainly performs byte copy operations).

- *Application*: the CPU usage of the adaptation engine is application-dependent. Shot selection is more complex than temporal scalability because large parts of the sequence might be skipped (in case they do not belong to the wanted shots), which implies that the transformation needs to be faster than real time during the skipping of particular shots. Combining two applications takes more CPU time; however, the second XML transformation (i.e., temporal scalability) only has to transform the gBSD fragments selected by the first transformation (i.e., shot selection). The CPU usage of the streaming server is application-independent as shown in Table 5.2.

- *Coding format*: the CPU usage of both the adaptation engine and the streaming server is format-dependent. The adaptation engine takes more CPU usage for MPEG-4 Visual than for H.264/AVC due to the more verbose gBSD for the MPEG-4 Visual bitstream. This larger gBSD demands more CPU time during the XML transformation. The streaming server takes more CPU usage for H.264/AVC than for MPEG-4 Visual because the parsing process is more complex for H.264/AVC in order to assign proper timestamps to video frames.

- *The level of detail of the gBSD*: this factor only influences the CPU usage of the adaptation engine. This can be deduced from Table 5.2 by comparing the CPU usage of the shot selection application once for a gBSD with detail up to frame level and once for a gBSD with detail up

Table 5.2: File sizes and CPU usage for the different MuMiVA scenarios.

	H.264/AVC				MPEG-4 Visual		
	TS[a]	SS	SS+TS	SS_c	TS	SS	SS+TS
Transformed gBSD size (KB)	122	96	53	6	280	156	122
Adapted bitstream size (MB)	21.8	16.2	9.9	18.0	43.5	31.4	19.1
Adaptation Engine CPU_A[b] (%)	3	5	7	4	8	9	10
Adaptation Engine CPU_P (%)	5	8	10	6	15	20	25
Streaming Server CPU_A (%)	25	24	24	23	19	18	19
Streaming Server CPU_P (%)	50	50	50	50	45	45	40

[a] TS, SS, and SS_c denote Temporal Scalability (obtain a frame rate of 12 fps), Shot Selection (select 25 shots out of 60 shots), and Shot Selection with a coarse-grained gBSD, respectively.
[b] CPU_A and CPU_P denote average and peak CPU usage, respectively.

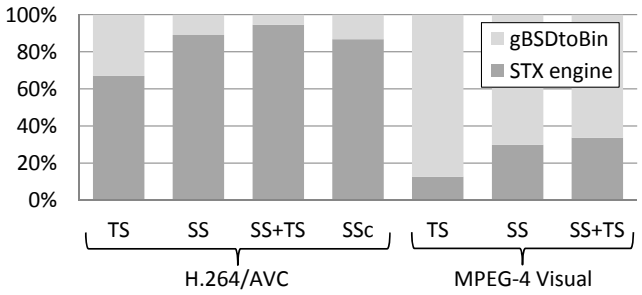


Figure 5.4: Partition of the execution times (including I/O operations) of the STX engine and the gBSDtoBin parser.

to RAU level. The CPU usage of the adaptation engine increases when the level of detail of the gBSD increases. Obviously, the CPU usage of the streaming server is independent of the level of detail of the gBSD.

The memory usage is constant for both the adaptation engine and the streaming server (a maximum of 5 MB of memory is used). Moreover, applications, coding formats, and the level of detail of a gBSD have a negligible impact on the memory usage of these components.

To examine the adaptation engine in more detail, the proportion in terms of execution times between the STX engine and the gBSDtoBin parser is depicted in Figure 5.4. In case of the MPEG-4 Visual format, the gBSDtoBin parser takes most of the execution time (63 % to 83 %) because the bit rate of the MPEG-4 Visual sequence is twice as high as the bit rate of the H.264/AVC sequence. Hence, more I/O operations need to be performed by the gBSDtoBin parser for the MPEG-4 Visual bitstream. Furthermore, the application also

influences the proportion between the STX engine and the gBSDtoBin parser, i.e., the STX engine will take more time when more complex transformations are executed.

5.2.5 Shortcomings of MuMiVA

As discussed in Chapter 4, XML-driven content adaptation has a number of disadvantages, despite its format-independent nature. The XML filters are dependent on the structure of the metadata and underlying coding formats. Furthermore, due to interoperability problems between XML-based metadata standards [41], integration with content metadata occurs in an ad-hoc manner. Mappings between different XML-based metadata formats need to be implemented in different XML filters. Hence, creators of these XML filters cannot think in terms of high-level adaptation operations but have to be aware of the underlying coding and metadata formats.

Another problem is the multimedia delivery within MuMiVA, which is implemented by means of a dedicated and coding-format specific streaming server. Hence, an adapted media bitstream that is sent to the streaming server will be parsed again in order to determine the different fragments and their corresponding timestamps. Also, when a new coding format needs to be supported, the streaming server needs to be extended for being able to correctly parse bitstreams compliant with the new coding format. Hence, no generic solution for multimedia delivery is present.

5.3 NinSuna

In order to solve the shortcomings of our MuMiVA platform, we present the design and functioning of NinSuna[5], which is a fully integrated and format-independent platform for the purpose of multimedia adaptation and delivery [134, 135]. Its basic design is inspired by the principles of XML-driven content adaptation techniques, while its final design consists of a hybrid architecture using both XML and Semantic Web technologies such as the Resource Description Framework (RDF, [72]), the Web Ontology Language (OWL, [83]), and the SPARQL Protocol And RDF Query Language (SPARQL, [99]). Furthermore, a tight coupling exists between NinSuna's design and the model for media bitstreams for describing the structural, content, and scalability properties of media bitstreams (as discussed in Chapter 4). This

[5] NinSuna is short for "The NinSuna INtelligent Search framework for UNiversal multimedia Access". A website containing information regarding the NinSuna platform and an online demo is available on `http://multimedialab.elis.ugent.be/NinSuna`.

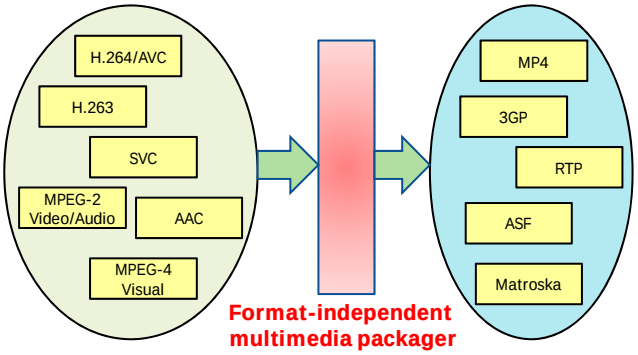


Figure 5.5: Obtaining a format-independent multimedia packager.

model, implemented using OWL, provides support for a seamless integration of adaptation operations and content metadata. Furthermore, we extend this model so that it allows format-independent packaging and delivery of multimedia content.

5.3.1 Format-independent Multimedia Content Packaging

A logical step after the adaptation of multimedia content is multimedia delivery. Multimedia content is usually not delivered as elementary bitstreams but packed in a particular delivery format. Today, a significant number of delivery or packaging formats exists; examples are MPEG-4 Part 14 (MP4 file format, [59]), Material eXchange Format (MXF, [111]), and Real-time Transport Protocol (RTP, [105]). As illustrated in Figure 5.5, we have to deal with different coding formats on the one hand, and different delivery formats on the other hand. Our goal is to develop a format-independent multimedia packager, i.e., a generic software module that is independent of the incoming coding format and the outgoing delivery format. An additional challenge is the coupling of format-independent multimedia adaptation with format-independent multimedia packaging.

Encapsulating multimedia content in a particular delivery format typically consists of two main processes: fragmentation and packetization. The fragmentation process divides the input media bitstream into portions, each representing one fragment. A simple example of a fragment in the context of video streams is a single frame. The packetization process selects (and possibly aggregates) the obtained fragments and maps them to the output delivery format. This mapping includes the assignment of timestamps and the addition of syntactical structures such as packet headers.

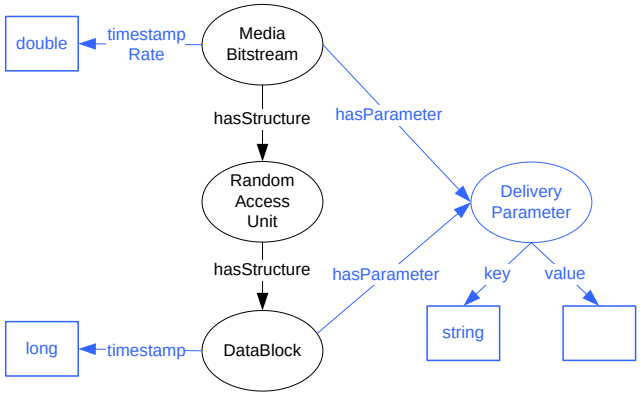


Figure 5.6: Extending the model for media bitstreams to support format-independent multimedia packaging.

In this section, we present a new method for format-independent multimedia content packaging. We use MPEG-B BSDL to abstract the packed media bitstream and to enable the use of format-agnostic software modules. It is based on an extension of our model for media bitstreams, as previously outlined in Chapter 4. Furthermore, a seamless integration is obtained between multimedia content adaptation and packaging in a format-independent way.

5.3.1.1 Extension of the Model for Media Bitstreams

In Figure 5.6, an overview is given of the extensions added to the model for media bitstreams in order to support format-independent packaging. Two categories can be distinguished: support for timestamps and support for delivery parameters.

A *timestamp* property is added to the *DataBlock* class, which represents a number related to the display time of the data block. In order to actually calculate the display time, the *timestampRate* property is added to the *MediaBitstream* class. The latter contains a number indicating the amount of timestamps that are contained in one second. Note that a data block does not necessarily correspond to a fragment (i.e., the outcome of the fragmentation process as described above). Multiple data blocks can make one fragment (e.g., data blocks representing H.264/AVC slices can be grouped in one fragment). However, one data block cannot be split up into several fragments.

Next, in order to assist in the fragmentation and packetization process, it is possible to define *DeliveryParameter*s. This can be done at the level of a media bitstream, as well as at the level of a data block. An example of

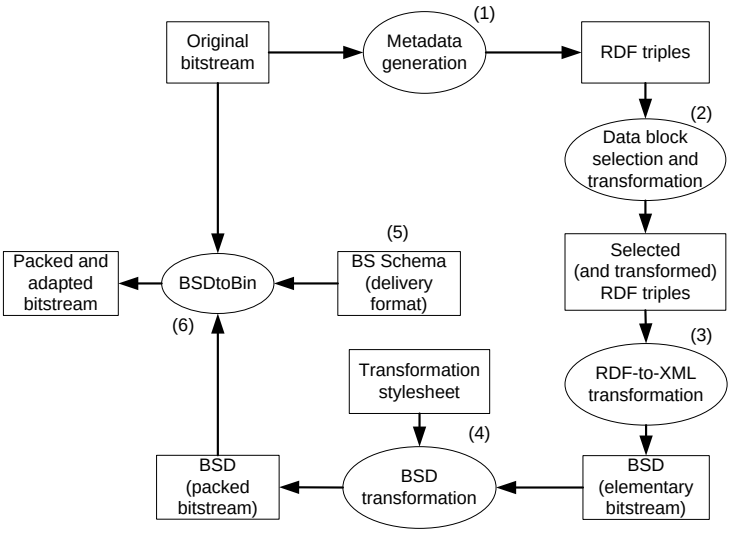


Figure 5.7: The general workflow of model-driven content adaptation and packaging.

a delivery parameter added at the level of a media bitstream is the sampling frequency when the underlying coding format is AAC (i.e., *key* property is equal to 'samplingFrequency', while the *value* property is for example equal to '48000'). The sampling frequency is an example of a delivery parameter that is needed by the packetization process, since the value of this parameter may be needed in the headers of a particular delivery format.

5.3.1.2 Coupling Model-driven Content Adaptation with Multimedia Packaging

The extension of our model for media bitstreams enables the creation of metadata (compliant with the model) that can assist in the format-independent adaptation and packaging of media bitstreams. The general workflow of model-driven content adaptation and packaging is depicted in Figure 5.7. Explanatory notes are given below.

(1) *Metadata generation*: media bitstreams that need to be adapted and packaged in our framework have to be equipped with metadata compliant with our (extended) multimedia model. As discussed in Chapter 4, the metadata generation can occur during the encoding process or by means of a (generic or format-specific) software module. Note that such a generic solution could rely on the principles of techniques such

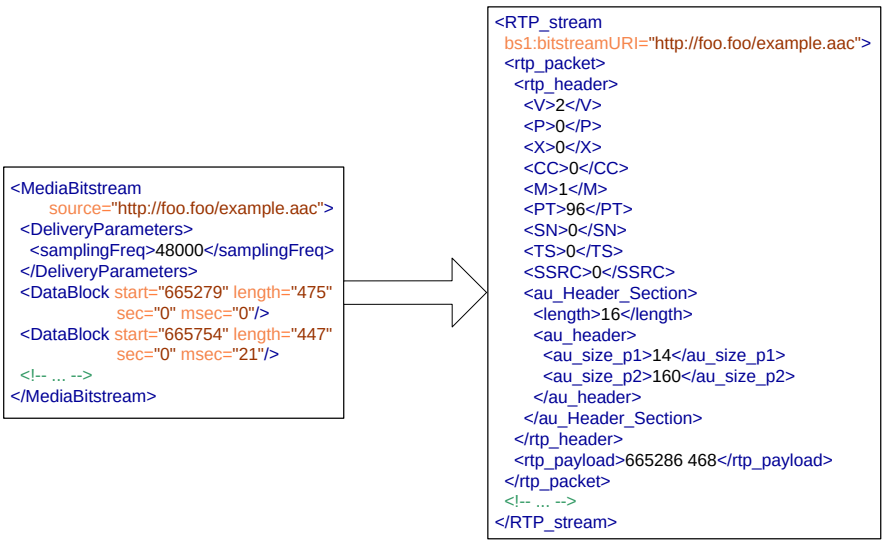


Figure 5.8: BSD-driven RTP packaging of AAC media bitstreams.

as BSDL or BFlavor, where structural metadata is generated based on a description of the high-level structures and syntax elements of a particular coding format. The result of the metadata generation process is a collection of RDF triples compliant with the model for media bitstreams.

(2) *Data block selection and transformation*: as discussed in Chapter 4, adaptation of media bitstreams is performed by selecting the proper data blocks and by possibly transforming the selected data blocks.

(3) *RDF-to-XML transformation*: instead of creating an adapted, elementary media bitstream based on the selected (and adapted) data blocks, we perform a simple RDF-to-XML transformation. The result of this transformation is a BSD (see Chapter 2) which can be used to create a packaged version of the adapted media bitstream. An example of such a BSD is shown on the left-hand side of Figure 5.8. The classes and properties defined in our model, needed for the packaging process, are mapped to XML elements and attributes respectively. Note that the timestamps are represented in terms of seconds and milliseconds.

(4) *BSD transformation*: the actual packaging process starts with the transformation of the BSD representing the adapted, elementary media bitstream. The resulting BSD represents an adapted and packaged media bitstream. Figure 5.8 illustrates the BSD transformation for the RTP

packaging of an AAC bitstream. The obtained BSD is compliant with MPEG-B BSDL, which implies that the BSDL framework can be used for further processing. The BSD transformation can be implemented using XSLT or STX, which enables the use of a format-independent transformation engine. However, it is important to note that the transformation stylesheets are not only dependent on the target delivery format, but also on the incoming coding format since each coding format requires a different packaging (i.e., fragmentation and packetization) strategy.

(5) *BS Schema creation*: as discussed in Chapter 2, a BS Schema describes the high-level structures and syntax elements of a particular format. In this case, a BS Schema for the target delivery format needs to be created. The BSD obtained in the previous step needs to be compliant with this BS Schema.

(6) *Adapted and packed bitstream generation*: finally, an adapted and packaged media bitstream can be created using BSDL's format-independent BSDtoBin parser, based on the BSD representing the adapted and packaged media bitstream, the BS Schema for the target delivery format, and the original media bitstream.

As discussed above, packaging media bitstreams typically consists of two main processes: fragmentation and packetization. It is not trivial to see where these two processes actually occur in the above discussed workflow for model-driven content adaptation and packaging. Fragmentation is realized during the BSD transformation process, where the data blocks are mapped to fragments. Packetization is spread across multiple steps. One aspect is the assignment of timestamps to fragments. During the metadata generation step, the data blocks are labeled with initial timestamps (i.e., timestamps of the original media bitstreams). However, since the adaptation process can cause gaps in the initial timestamps (e.g., a particular scene is deleted during the adaptation), these timestamps need to be recalculated (i.e., the gaps need to be detected and corrected). The latter is performed during the RDF-to-XML transformation. The packetization process also includes the addition of syntactical structures such as packet headers. This is done during the BSD transformation.

The choice to go back from RDF to XML during the packaging process can be justified as follows. We introduced Semantic Web technologies such as RDF to enhance the interoperability between different metadata standards. However, the latter is mainly an adaptation issue. More specifically, semantic adaptations such as scene selection based on content metadata are suffering from these interoperability problems (in case the adaptation is performed

in the XML domain). In order to obtain packaging of media bitstreams in a format-independent way, a description of headers and syntax elements of the target delivery format is needed, together with pointers to data segments in the original bitstream, which is exactly what already existing technologies such as MPEG-B BSDL support.

5.3.2 The NinSuna Platform

NinSuna is a fully integrated platform for multimedia adaptation and delivery in heterogeneous usage environments, relying on both XML and Semantic Web technologies for the implementation of format-independent adaptation and packaging engines. Furthermore, it aims at being deployable in streaming environments. Its multimedia content adaptation and packaging techniques rely on our model for media bitstreams. Note that these adaptation and packaging techniques were previously discussed in Chapter 4 and Section 5.3.1 respectively. Next to the delivery of multimedia content, NinSuna also provides support for uploading content with corresponding metadata.

5.3.2.1 Architecture

The NinSuna architecture is shown in Figure 5.9. Three layers can be distinguished: the storage layer, the processing layer, and the front-end layer. The storage layer consists of a multimedia content server, containing the media bitstreams, and an RDF repository, containing all the necessary metadata (i.e., RDF triples compliant with our model for media bitstreams). The processing layer contains an Adaptation and Packaging Engine (APE) (the working of which has been discussed in Section 5.3.1.2), a Structural Metadata Generator (SMG) that enables the creation of RDF triples compliant with the structural part of the model, and an Adaptation Decision-Taking Engine (ADTE). The ADTE calculates adaptation parameters, initializes the APE, and manages the different client sessions [87]. The front-end layer provides access points for clients of the NinSuna platform. New media bitstreams can be uploaded with their corresponding metadata using the authoring service. Information regarding media bitstreams can be obtained using the retrieval service. The adapted and packaged multimedia content is retrieved through the download or streaming service. The streaming service implements the RTSP protocol [106], while the download service makes multimedia content available through (progressive) download-and-play scenarios (e.g., using MP4 as delivery format).

Workflow Explanatory notes for the workflow within the NinSuna platform (see Figure 5.9) are given below. The NinSuna platform allows authoring

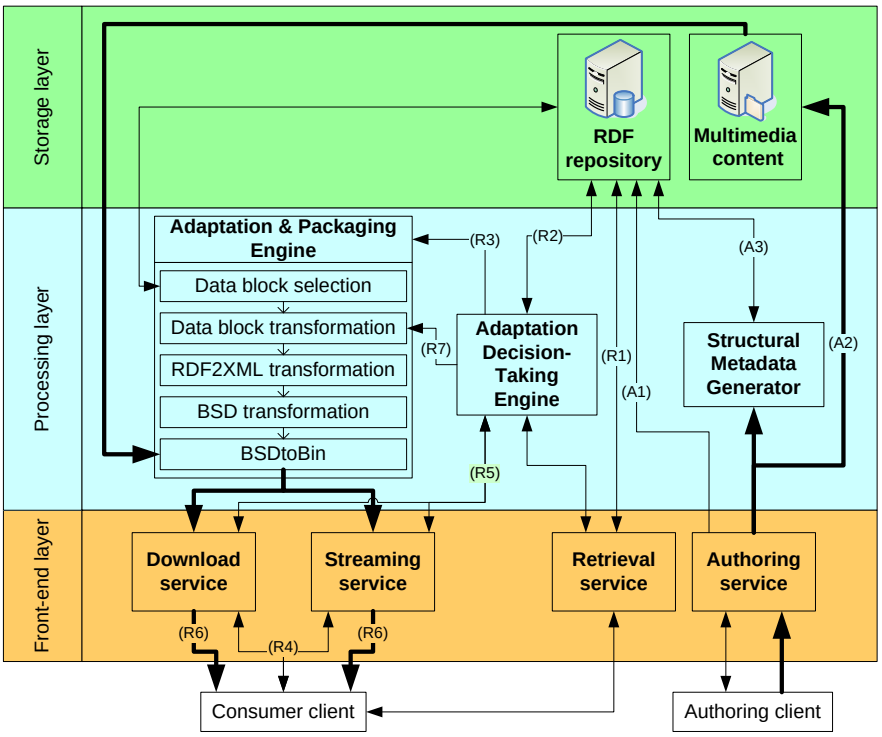


Figure 5.9: Architecture of the NinSuna platform.

clients to communicate with the authoring service in order to extend the multimedia database. The workflow for uploading new multimedia content to the NinSuna platform is as follows.

(A1) The authoring client sends content annotations for a particular media bitstream to the authoring service. These annotations, stored in the RDF repository, consist of RDF triples compliant with the content metadata part of the model for media bitstreams.

(A2) The authoring client uploads the actual multimedia content, encoded with a specific coding format, to the multimedia content repository.

(A3) The SMG is used to generate the structural part of the metadata belonging to the uploaded media bitstream. It takes as input the encoded media bitstream and its content annotations and produces RDF triples compliant with the structural part of the multimedia model. The RDF triples are subsequently stored in the RDF repository. Note that the content

metadata are needed by the SMG to create a mapping between the structural and content metadata (i.e., to connect random access units with temporal segments using the *hasBitstreamData* property, as discussed in Chapter 4).

Consumer clients of the NinSuna platform can make use of three services: retrieval, download, and streaming. The workflow for retrieving multimedia content is given below.

(R1) The retrieval service makes it possible to query the RDF repository (by making use of SPARQL) in order to browse through the multimedia content, based on the available content metadata. Hence, the retrieval service can be compared to a SPARQL endpoint [23] with a number of additional shortcuts to retrieve media bitstreams. Note that only the content metadata part can be browsed: the structural metadata are unavailable through the retrieval service since these metadata are irrelevant for the consumer client.

(R2) The consumer client sends a SPARQL query in order to request a particular media bitstream (for an example of such a query, see Listing 4.1), together with a description of its usage environment, to the retrieval service which passes this information to the ADTE. The latter creates a new session and selects coding and delivery formats based on the given usage environment characteristics. When the client desires both audio and video, two appropriate coding formats are selected. Next, the ADTE calculates the adaptation parameters for the selected coding formats by matching the scalability information of the requested content with the information about the usage environment (e.g., by comparing the video resolution with the screen size of the end-user device).

(R3) The ADTE initializes one or more APEs (see further) with the received SPARQL query and the calculated adaptation parameters. Furthermore, it creates an URL for the consumer client which indicates where the requested multimedia content can be found. This URL, which contains a session ID, is sent back to the consumer client.

The number of APEs that is initialized depends on the number of requested media bitstreams and the delivery service used. When only one media bitstream is requested, one APE is initialized. When two or more media bitstreams are requested (e.g., a video stream with a corresponding audio stream), the number of APEs that needs to be initialized depends on the requested delivery service (i.e., streaming or download). This is illustrated in Figure 5.10 for a video and corresponding audio

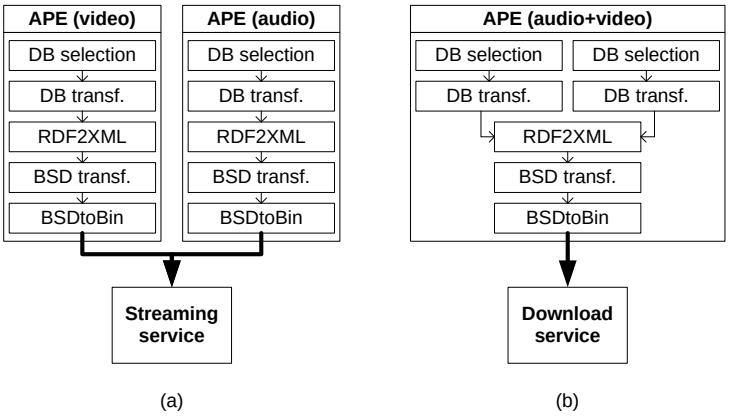


Figure 5.10: Structural variations of APEs during the adaptation and delivery of corresponding audio and video streams.

stream. If the media bitstreams need to be delivered using the streaming service, two APEs are initialized (Figure 5.10(a)). Two separate streams of RTP packets are created for the streaming service, where each stream corresponds to a different RTP session. When the media bitstreams are delivered through the download service, merging of the two streams occurs earlier. More specifically, the two streams are associated with different adaptation processes (i.e., data block selection and adaptation), but these are merged at the beginning of the packaging process (i.e., the RDF-to-XML transformation) (Figure 5.10(b)). This way, the media bitstreams can be packed together in a delivery format targeted for (progressive) download-and-play scenarios (e.g., MP4).

(R4) Dependent on the received URL, the consumer client contacts the progressive download or streaming service to retrieve the desired content. When the streaming service was selected, the consumer client starts an RTP/RTSP session with the NinSuna platform.

(R5) By using the session ID (included in the URL), the download or streaming service contacts the ADTE in order to obtain the proper session. Hence, based on the answer of the ADTE, the download or streaming service can determine which APE(s) will provide the requested content.

(R6) The APEs start working (as described in Section 5.3.1.2) and provide the adapted and packaged media bitstream to the download or streaming service. When the streaming service is selected, the APE provides a

stream of RTP packets. The latter are sent out by the streaming service to the consumer client according to the RTSP protocol.

(R7) When the usage environment conditions change (e.g., the network connection changes from broadband to smallband), adaptation parameters can be dynamically adjusted. The consumer client uses the retrieval service to announce its new usage environment. The ADTE recalculates and changes the adaptation parameters. As discussed in Chapter 4, to avoid the initialization and evaluation of a new query each time an adaptation property changes, support for dynamic adaptations is provided in the data block transformation step.

Distributing NinSuna across the Network Communication within the NinSuna platform is realized using the HTTP protocol [46]. Hence, the different components can be distributed across a network. First of all, the different layers (storage, processing, and front-end) can be divided across different machines. Furthermore, a pool of APEs and SMGs can be created to serve a large number of clients. These pools can also be divided across multiple machines. The management of these pools can be done by brokers in the network deciding which APE and/or SMG to select based on the current load. Note that such a distributed architecture increases the scalability of the platform, since it allows extending the platform with additional components in order to anticipate an increasing load.

Extensibility One of the main features of the NinSuna platform is its format-independency. Both the adaptation and packaging processes are format-independent. Hence, it is rather straightforward to extend the platform with support for new coding, metadata, and delivery formats. Adding support for a new coding format consists of the following steps.

- A mapping needs to be created between the coding format and the model for media bitstreams. This mapping is actually implemented and performed during the metadata generation step.
- XML transformation filters (implemented in STX or XSLT) need to be created for use during the BSD transformation step. More specifically, delivery formats that support the encapsulation of the new coding format and that are already available in the platform (i.e., a BS Schema exists for the delivery format) need to be taken into account. Hence, for each delivery format that needs to be supported for the new coding format, an XML transformation filter needs to be created.

New metadata formats are inherently supported thanks to the use of our OWL-based model for media bitstreams: it is only necessary to align the new metadata format (i.e., ontology) with the content metadata part of the model. Finally, a new delivery format is supported by taking the following steps.

- A BS Schema needs to be created, which describes the high-level syntax structures and syntax elements of the delivery format (as discussed in Section 5.3.1.2).

- XML transformation filters (implemented in STX or XSLT) need to be created for use during the BSD transformation step. More specifically, coding formats that are already supported by the platform and that are allowed to be encapsulated in the new delivery format need to be taken into account. Hence, for each coding format that needs to be supported for the new delivery format, an XML transformation filter needs to be created.

5.3.2.2 Implementation

The Java Platform is used to implement NinSuna. Sesame[6] (version 2.1), which is an open source RDF database with support for RDF Schema inferencing and querying, is used as RDF repository. The Sesame RDF API is used to access the repository and to evaluate the SPARQL queries. Within the APE, Saxon 6.5.5 and Joost v.2008-05-28 are used as XSLT and STX transformation engine respectively. Also, an own Java implementation of the BSDtoBin parser is used. Note that we did not use the BSDL reference software due to a lack of support for multithreading. The SMG consists of parsers generated by Flavor [44], enhanced with support for the generation of RDF triples compliant with the structural part of the model for media bitstreams. Finally, the streaming service uses the RTSP implementation available in the C++ library of Live555 Streaming Media.

5.3.2.3 Performance Measurements

In this section, a number of performance measurements are presented to provide the reader with an impression of the performance of the NinSuna platform. First, a use case scenario is discussed, which is then followed by the experimental results.

[6] Available on `http://www.openrdf.org/`.

Use Case Scenario A number of news sequences were used to test our adaptation and delivery platform [79]. Semi-automatic annotation was used for each news sequence, i.e., shots were automatically detected after which each detected shot was manually annotated by a number of keywords. When mapping these metadata to our model for media bitstreams, each shot corresponds to a *TemporalSegment* which contains a *keyword* property (values for this property correspond to the keywords).

The scenario to obtain (parts of) a news sequence is as follows. The user searches, based on keywords, for news sequences containing news topics that are of his/her particular interest. Next, the user requests the selected news scenes and provides a description of the usage environment to the NinSuna platform. The latter selects the requested audio and video scenes, performs structural adaptations if needed (i.e., exploitation of scalability such as frame rate scaling), and packages the selected streams.

The multimedia content archive of NinSuna contained seven news sequences, each having a resolution of 720x432, a frame rate of 25 fps, and a length of approximately 22 minutes. The video streams of the news sequences were encoded using H.264/AVC. A hierarchical coding structure was used to obtain three layers of temporal scalability (i.e., the videos can be rescaled from 25 fps to 12.5 fps and 6.25 fps) [30]. Instantaneous Decoding Refresh (IDR) frames were inserted every 16 frames to obtain feasible random access. Further, the audio streams of the news sequences were encoded using AAC (with a sampling frequency equal to 48000). Additional bitstream characteristics can be found in Table 5.3.

Two delivery formats are available in our scenario: RTP (streaming service) and MP4 (download service). Hence, three XML transformation stylesheets were developed: one STX stylesheet to guide the packaging of an H.264/AVC stream into RTP packets, one STX stylesheet to guide the packaging of an AAC stream into RTP packets, and one XSLT stylesheet to guide the packaging of an H.264/AVC and AAC stream into an MP4 container. We use STX stylesheets to guide the RTP packetization because of STX's streaming capabilities. An XSLT stylesheet is used for the packaging into an MP4 container because the format defines header values containing information related to the whole media bitstream, and these values need to be calculated at runtime (e.g., length of the resulting MP4 file and random access points in the media bitstreams). Note that we could also used STX to implement the packaging process for MP4, but that would have introduced a significant overhead in terms of implementation effort because of the streaming character of STX.

Table 5.3: Overview of the bitstream characteristics.

		Video		Audio	
Name	Length (s)	Size (MB)	Bit rate (MBit/s)	Size (MB)	Bit rate (Kbit/s)
news1	1302	217.5	1.34	19.7	124
news2	1301	198.7	1.22	19.4	122
news3	1274	184.7	1.16	19.9	128
news4	1284	196.9	1.23	20.0	128
news5	1460	198.2	1.09	22.3	125
news6	1269	187.3	1.18	19.2	124
news7	1305	174.4	1.07	20.4	128

Experimental Results Performance measurements were done on a PC having an Intel Pentium D 2.8 GHz CPU and 1 GB of system memory at its disposal. The operating system used was Windows XP Pro SP2, running Java 2 Runtime Environment (SE version 1.5.0_09). JProfiler 5.1.4 was used to profile our platform components. All time measurements were executed six times, whereupon an average was calculated over the last five runs to avoid startup effects.

Structural Metadata Generation As discussed in Section 5.3.2.1, the SMG enables the creation of RDF triples compliant with the structural part of the model for media bitstreams. Enhanced Flavor-based [44] parsers are used to implement the SMG. For each of the seven news sequences, the SMG is used to generate their structural metadata. The memory usage of the SMG is low and constant (approximatly 3 MB). Its execution times are provided in Table 5.4. For all media bitstreams (audio and video streams), the SMG is able to generate the structural metadata in real time (i.e., the execution speed is higher than the bit rate of the media bitstream). The execution time of the SMG is dependent on the following parameters.

- *# parse units per second*: a media bitstream characterized by a higher number of parse units per second implies a higher execution time for the SMG. Note that the number of parse units per second is dependent on the coding format (and its encoding parameters). For instance, a parse unit in H.264/AVC corresponds to a Network Abstraction Layer Unit (NALU). We have encoded the seven video news sequences in such a way that each NALU corresponds to one frame. Hence, the video streams are characterized by 25 parse units per second. Further, parse

Table 5.4: Execution times for the generation of structural metadata.

	Video			Audio		
Name	Time (s)	Speed (MBit/s)	# Data blocks	Time (s)	Speed (KBit/s)	# Data blocks
news1	695.8	2.5	32561	398.3	405.5	61051
news2	753.0	2.1	32538	381.8	417.0	61008
news3	634.2	2.3	31875	395.7	412.0	59765
news4	625.1	2.5	32111	396.2	413.4	60207
news5	779.8	2.0	41008	474.4	385.3	76889
news6	736.6	2.0	31729	403.0	391.3	59491
news7	659.7	2.1	32638	392.5	425.6	61196

units for the audio news sequences correspond to AAC frames (containing 1024 samples), implying that the audio news sequences are characterized by 46.9 parse units per second.

- *# skipped bytes per parse unit*: a higher number of skipped bytes per parse unit implies a lower execution time for the SMG. These skipped bytes correspond to the coded (audio or video) data (e.g., motion vectors and transform coefficients), because the SMG only parses high-level syntax structures of a media bitstream. In our example, the video news sequences have a higher bit rate than the audio news sequences (see Table 5.3), implying that the video streams contain more coded data and hence that more bytes can be skipped by the SMG.

Table 5.4 also shows the number of data blocks that is generated for each media bitstream. Since each data block is represented as an RDF graph, we can calculate the number of RDF triples that is necessary to represent the structural metadata. One RDF data block graph consists of 5 RDF triples[7]. Further, one RAU (consisting of 3 RDF triples) points to 16 data blocks (for video) or 30 data blocks (for audio). For example, the total number of RDF triples to represent the structural metadata for the *news1* video sequence is equal to $(32561 * 5) + ((32561/16) * 3) = 168910$.

Delivery of News Fragments Three scenarios are considered to evaluate the delivery of (partial) media bitstreams. In the first scenario, a fragment

[7]The number of RDF triples to represent a class instance is one (to indicate the class) plus its number of properties.

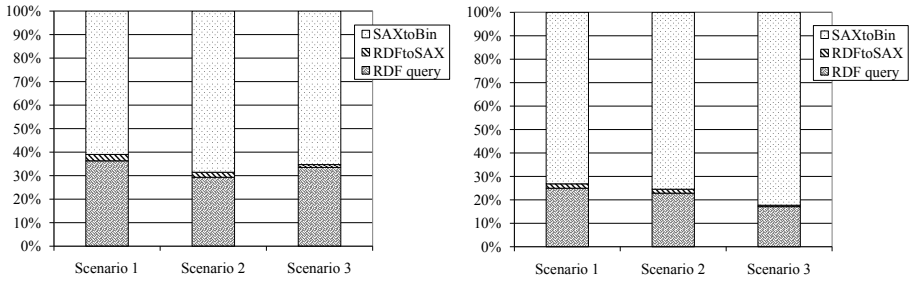
Table 5.5: Characteristics of the three delivery scenarios applied to the news2 sequence.

	Original start offset (s)	Fragment length (s)	Fragment size (MB)		Peak memory usage (MB)		Latency (s)	
			RTP	MP4	RTP	MP4	RTP	MP4
Scenario 1	18	128	12.0	11.8	10	24	0.2	2.4
Scenario 2	810	139	18.0	18.5	10	25	0.2	2.5
Scenario 3	18	267	30.8	30.3	10	37	0.2	8.2

of the media bitstream is selected occuring in the beginning of the media bitstream. The second scenario takes a fragment at the end of the media bitstream. The third scenario takes both fragments of scenarios 1 and 2. We distinguish these three scenarios to be able to investigate the impact of the place of the shots of interest in the original bitstream and the length of the selected fragments. More information regarding the resulting multimedia fragments of the *news2* audio and video sequences is provided in Table 5.5.

We have evaluated the media adaptation and delivery processes within NinSuna in terms of peak memory consumption and execution times. Regarding the peak memory consumption, we do not consider the memory usage of the Java Virtual Machine and the Java Application Server, i.e., only the memory usage is measured for the APE (i.e., data block selection and adaptation, RDF-to-XML transformation, and BSD transformation and BSDtoBin). As shown in Table 5.5, RTP delivery is characterized by a low and constant memory usage (i.e., 10 MB). On the contrary, delivery using MP4 introduces memory usage that is dependent on the length (in terms of data blocks) of the multimedia fragments. This is due to the XSLT transformation that needs to store the full XML document, resulting from the RDF-to-SAX transformation, in memory. Note that this is necessary due to the presence of headers occurring in front of the MP4 file and covering information regarding the full bitstream (e.g., a list of random access points). Also in Table 5.5, the time between the client's request and the first delivered byte (measured at server-side to avoid network delay) is provided (i.e., the latency). For RTP, the latency is low and independent of the length and position of the media fragment (i.e., 0.2 s). For MP4, the latency is dependent on the length of the requested media fragment. For instance, for scenario 3, the resulting MP4 file is available for (progressive) download after 8.2 s. Note that the latency for MP4 increases non-linearly because the performance of the XPath evaluation process during the XSLT transformation decreases, due to an increasing DOM tree.

In Figure 5.11, the proportion between components of the APE are shown in terms of execution time percentages. RDF query, RDFtoSAX, and SAXto-



(a) News fragments delivered using RTP (b) News fragments delivered using MP4

Figure 5.11: Proportion in terms of execution time percentages between components of the NinSuna platform.

Bin correspond to data block selection, data block transformation and RDF-to-XML transformation, and BSD transformation and BSDtoBin respectively. For RTP delivery, SAXtoBin takes most of the time with 65 % on average. RDFtoSAX only requires a small proportion in terms of execution time (2 %), while RDF query takes 33 %. These proportions are independent of the length of the resulting media fragment, i.e., they are only influenced by the resulting bit rate of the media fragment. For MP4 delivery, the proportions between the APE components are not independent of the length of the media fragment (the longer the fragment, the more time is spent by SAXtoBin). This is due to the decreasing performance of the BSD transformation when the size of the incoming XML document, which is dependent on the length of the media fragment, increases.

5.3.3 Limitations and Future Work

Although NinSuna solves a number of problems of the MuMiVA platform, there are still a number of issues that need to be solved in the future.

- *Increasing memory usage due to XSLT transformation*: as discussed in Section 5.3.2.3, the use of XSLT to obtain a BSD representing a packaged media resource introduces a memory usage that is dependent on the length of the requested media fragment. Therefore, XSLT should be avoided and replaced by STX or SAX filters. However, the memory usage is also dependent on the delivery format. More specifically, if the headers of a particular delivery format contain information regarding the whole media fragment (e.g., byte offsets), this information is only available when the entire BSD is constructed (for example as a DOM tree).

- *Structural metadata overhead*: the amount of overhead of the structural metadata is dependent on the length of the media bitstream. Hence, for long media bitstreams, the number of RDF triples representing the structural metadata can be a burden for the RDF store. As discussed in Chapter 4, one solution for this problem is to store the structural metadata in an RDF store which is specifically designed for the model for media bitstreams. More specifically, the structural metadata and scalability information can be stored in a highly scalable Relational Database Management System (RDBMS), using a database scheme based on the structural and scalability part of the model for media bitstreams. Another possibility is the definition of a container format based on the model for media bitstreams. This way, the structural metadata is serialized within the headers of this container format.

- *Structural metadata generation*: as discussed in Section 5.3.2.3, enhanced Flavor-based parsers are used to implement the structural metadata generation. Each Flavor-based parser is able to generate structural metadata compliant to the multimedia model for one particular coding format. Note that the enhancements of our parsers need to be implemented manually. As a result, the structural metadata generation module is the only software module within NinSuna containing format-specific software. To avoid this format-specific software, a BSDL-like BintoBSD parser could be created or the BFlavor specification could be extended to support the model for media bitstreams.

5.4 Synchronization

Our two platforms discussed in Section 5.2 and Section 5.3 (i.e., MuMiVA and NinSuna) are able to deliver adapted media bitstreams that were originally synchronized. Both platforms also support high-level semantic adaptations along the temporal axis (e.g., scene selection or video summarization). This kind of adaptations introduces problems for synchronized, compressed media bitstreams (e.g., synchronized audio and video streams). Typically, dependencies exist between different parse units (e.g., frames) in compressed media bitstreams. For instance, intra-coded frames are independent of other frames, while inter-coded frames are dependent on previous and/or future frames. To garantuee that the adapted media bitstream can be decoded in a correct way, cuts in the bitstream should only be performed at random access points, i.e. at frames that are independent of previous frames (see also Section 2.3.2). However, the problem with synchronized media bitstreams is that their random

access points do not necessarily coincide with each other. In this section, we discuss three possible approaches that can be applied within our two platforms.

5.4.1 Synchronization during XML Transformation

The first approach [129] is suited for XML-driven content adaptation and is implemented within the MuMiVA platform. The synchronization between the different media bitstreams is implemented within the XML transformation step. In other words, the transformation of a BSD is not only meant to implement the adaptation of the media bitstreams, but also to synchronize them.

The first step consists of finding a match between byte ranges (units used in a BSD) and timing information in terms of timestamps (units used in content metadata). Therefore, we propose the following algorithm to calculate the mapping between timestamps and byte ranges, taking into account the random access points of the compressed media bitstream [129]. Suppose we have already calculated the relationship between the byte ranges of a media bitstream and its display numbers (note that this relation is coding-format dependent). If D_i corresponds to the display number of frame i, R represents the frame rate of the media bitstream, and T_i is equal to the display time of frame i, then we can state that $T_i = D_i/R$ (we suppose the media bitstream is characterized by a fixed frame rate). From the content metadata, we know T_x, i.e., the start time of shot x. Hence, a display number D_{result} corresponding to a frame that represents the beginning of shot x on a random access point can be calculated as follows:

$$D_{result} = RAP(D_x) \text{ with } D_x = T_x * R.$$

The function $RAP()$ takes as input a display number D_x and returns the closest display number D_y, with y corresponding to a random access point and where $D_x \geq D_y$.

As an example, we want to select a fragment starting from frame with display number 71 (see Figure 5.12). However, this frame is not a random access point. Hence, $RAP(71)$ is in this example equal to 70. Therefore, the selected fragment will start at the frame with display number 70.

Within a XML-driven content adaptation engine, the above described algorithm for realizing semantic adaptations along the temporal axis is performed during the BSD transformation. First, based on the incoming BSD, display numbers are assigned to particular byte ranges (i.e., representing frames) of the media bitstream. Next, the BSD transformation takes as input the content metadata, selects the relevant scene information (based on the user's interests), and applies the mapping algorithm as discussed above. Finally, a decision is made for each frame whether the frame should be dropped or not.

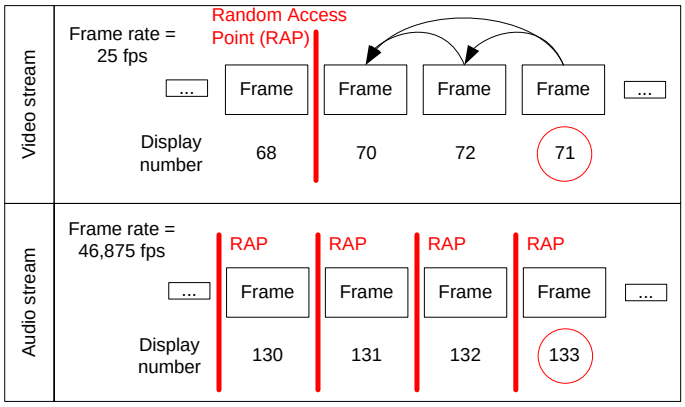


Figure 5.12: Random access points in synchronized audio and video streams.

In order to keep two or more media bitstreams synchronized during semantic adaptation along the temporal axis, we introduce the $SRAP()$ function, which provides the display number of the closest previous random access point that all the synchronized media bitstreams have in common. n media bitstreams have a common random access point c if $T_c = T_{x_1} = T_{x_2} = ... = T_{x_n}$, with T_x equal to the display time of frame x and x_i $(0 < i \leq n)$ a frame corresponding to a random access point. Hence, for a number of synchronized media bitstreams, the formula to calculate the correct display number D_{result} corresponding to a frame that represents the beginning of shot x is equal to:

$$D_{result} = SRAP(D_x) \text{ with } D_x = T_x * R_b,$$

with R_b representing the fixed frame rate of a media bitstream b. When the media bitstream is characterized by a varying frame rate, the display time of a particular frame needs to be determined using external information (e.g., presentation times available in an MP4 container).

An illustration of the above formula can be found in Fig. 5.12. Two synchronized audio and video streams need to be cut at 2.84 s. In this example, the audio stream is straightforward to adapt, since each audio frame corresponds to a random access point. When no synchronization was needed between the audio and video stream, we could have cut the audio stream in this example at audio frame 133. However, since we need to obtain synchronization with the video stream, the $SRAP()$ function needs to be applied. Since the closest previous random access point of the video stream is video frame 70 (or timepoint 2.80 s), the random access point for the audio stream that needs to be selected is audio frame 131.

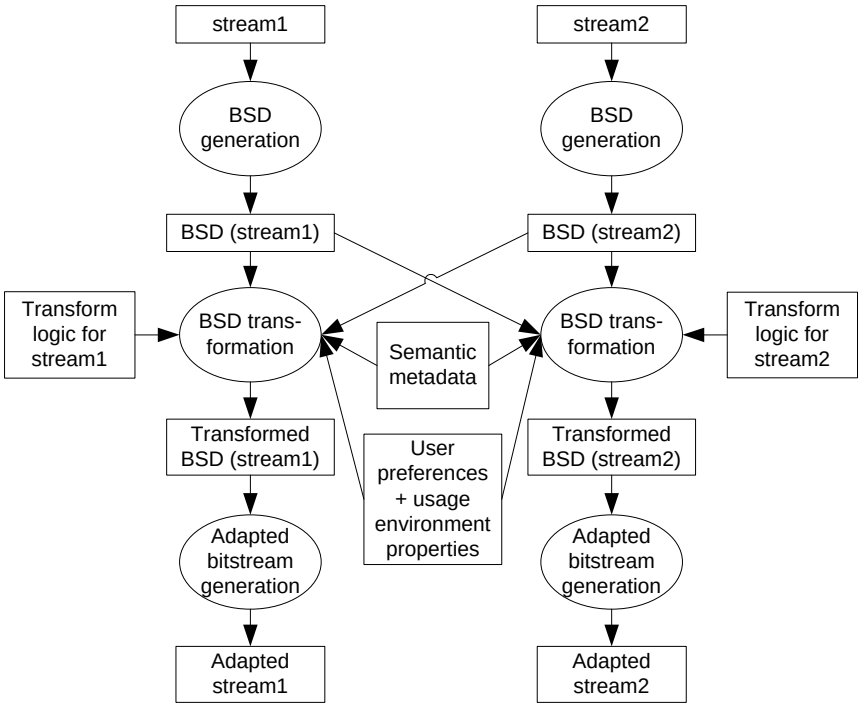


Figure 5.13: Adapting synchronized media bitstreams using XML-driven content adaptation.

The general workflow for the adaptation of two synchronized media bitstreams using format-independent adaptation engines is depicted in Fig. 5.13. The two adaptation chains for both streams can be executed in parallel. The content metadata and user preferences (i.e., the desired scenes) are provided as an input for the BSD transformation processes. To be able to evaluate $SRAP()$, the BSD transformation needs input from the other (synchronized) media bitstreams by means of their BSD. This is necessary to determine the display numbers of the random access points occurring in the other (synchronized) media bitstreams. Note that, except for live scenarios, the BSD generation is only performed once for each bitstream. Hence, the information regarding the occurrence of random access points can be extracted from the obtained BSDs. In this case, the BSD transformation takes as input, next to the BSD and the transformation logic, a list of display times (corresponding to random access points) for each media bitstream.

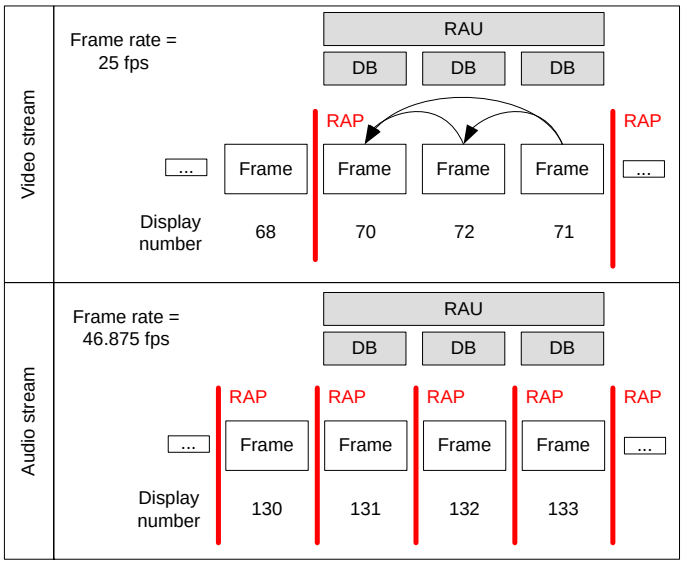


Figure 5.14: Random Access Unit alignment with the model for media bitstreams. RAP, RAU, and DB stand for Random Access Point, Random Access Unit, and Data Block respectively.

5.4.2 Synchronization during Structural Metadata Generation

A second option is to implement the synchronization between different media bitstreams in the structural metadata generation step. We used this approach within the NinSuna platform. The idea is to align the Random Access Units (RAUs) in the structural metadata, as illustrated in Figure 5.14. As discussed in Chapter 4, RAUs start with a Random Access Point (RAP) and end just before the next RAP. Hence, generation of structural metadata (compliant with the model for media bitstreams) of the audio stream depicted in Figure 5.14 should result in one RAU per audio frame (because each audio frame is a RAP). However, because this audio stream is synchronized with a video stream, the RAUs of both bitstreams are aligned during the structural metadata generation.

An advantage of this approach is that the adaptation process (i.e., selection and adaptation of data blocks) is totally unaware of synchronization issues between different bitstreams. Semantic adaptations will not disturb the synchronization between two or more streams, because these adaptations are based on the selection of RAUs (as discussed in Chapter 4).

During the structural metadata generation step, the media bitstream which is least flexible in terms of random access (i.e., the video stream in Figure 5.14)

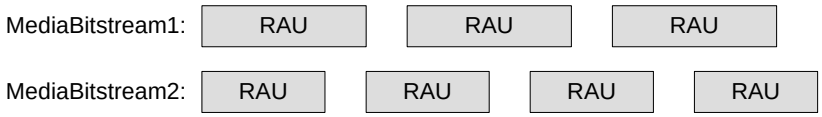


Figure 5.15: Synchronized media bitstreams with non-coinciding random access units.

is handled first. Next, structural metadata are generated for the other synchronized bitstreams (i.e., the audio stream in Figure 5.14), taking into account the structural metadata of the first bitstream (i.e., its RAUs). This way, the RAUs of the synchronized media bitstreams can be aligned as depicted in Figure 5.14.

However, there are a number of limitations when the synchronization issue is implemented in the structural metadata generation step:

- When only one of the synchronized media bitstreams is requested, we still have to deal with the aligned random access units (due to the synchronization). For instance, consider the audio stream in Figure 5.14. If this audio stream needs to be delivered from timepoint 2.84 s (i.e., audio frame 133) without its corresponding video stream, the audio stream needs to be cut at audio frame 131 (i.e., the start of a random access unit). Hence, although only the audio stream needs to be delivered and each audio frame corresponds to a random access point, the aligned random access units need to be taken into account.

- If two or more synchronized media bitstreams have less random access in common, synchronization in the structural metadata generation step becomes infeasible. Such a scenario is depicted in Figure 5.15. The resulting aligned random access units will be too large in such cases, which will cause an insufficient amount of random access for the media bitstreams.

5.4.3 Synchronization during Packaging

In order to avoid the limitations of implementing the synchronization in the structural metadata generation step, we propose a third possibility to solve the synchronization issues caused by semantic adaptations. Therefore, we shift the synchronization from the structural metadata generation to the packaging process (discussed in Section 5.3.1). Since semantic adaptations can cause gaps in the initial timestamps of the data blocks, the timestamps of the data blocks need to be recalculated (i.e., the gaps need to be detected and corrected)

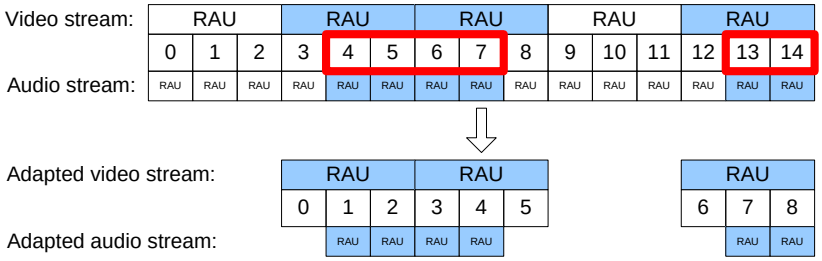


Figure 5.16: Recalculation of timestamps after semantic adaptation.

during the packaging process (and more specifically during the RDF-to-XML transformation), as explained in Section 5.3.1.2.

When synchronization is implemented during the structural metadata generation step, the random access units of the synchronized media bitstreams were aligned. Hence, semantic adaptations cause gaps at the same positions for all synchronized media bitstreams. However, when we drop the condition that random access units of synchronized media bitstreams have to be aligned, the recalculation algorithm for the timestamps needs to be adjusted. In Figure 5.16, a scenario is shown with two synchronized media bitstreams having non-aligned random access units[8]. Suppose we want to select data blocks from the synchronized media bitstreams starting from timestamp 4 to 7 and from 13 to 14. The algorithm to obtain the corrected timestamps can be formulated as follows. Initialize the variable $timestampFix$ to 0. Handle all synchronized media bitstreams in parallel and search for the next gap in the sequence of timestamps. Meanwhile, correct the timestamps of the processed data blocks with $timestampFix$. When a gap is detected in all the synchronized media bitstreams, add the number corresponding to the smallest detected gap to $timestampFix$. Repeat these steps until the end of the synchronized bitstreams is reached.

We illustrate this algorithm by applying it to the example given in Figure 5.16. For the video stream, the first gap that is detected is situated at the first selected data block, which has a timestamp equal to 3. The corresponding gap detected in the audio stream is equal to 4. Hence, $timestampFix$ is increased by $min(3, 4) = 3$. Hence, timestamps 4, 5, 6, and 7 are corrected by 3 resulting in the timestamps 1, 2, 3, and 4 for the audio stream. The next gap is located at the data block with timestamp equal to 12 for the video stream (gap is equal to 3). The corresponding gap for the audio stream is detected at the

[8]Note that, as a matter of convenience, both media bitstreams have the same number of timestamps per second.

data block with timestamp equal to 13, resulting in a gap equal to 5. Hence, $timestampFix$ is increased by $min(3, 5) = 3$ and is now equal to 6. For example, the data blocks of the video stream with timestamp equal to 12, 13, and 14 are thus corrected to 6, 7, and 8.

As discussed in Section 5.3.2.1, there are two possible scenarios for the packaging process, based on the requested delivery service (i.e., streaming or download). It is important to remark that the revised timestamp recalculation algorithm can only be implemented in the RDF-to-XML transformation in case of the download scenario (see Figure 5.10(b)). In case of the streaming scenario (see Figure 5.10(a)), separate RDF-to-XML transformation processes are present for the different synchronized media bitstreams. Since no communication between these processes is available, it is impossible to detect the smallest gap of all synchronized media bitstreams (as required by the above described algorithm). Therefore, in the streaming scenario, communication between the APEs is necessary to implement the timestamp recalculation in the RDF-to-XML transformation process. Another option is to implement the timestamp recalculation in the streaming service.

5.5 Related Work

European projects such as ISIS (Intelligent Scalability for Interoperable Services, [47]) and DANAE (Dynamic and distributed Adaptation of scalable multimedia coNtent in a context-Aware Environment, [101]) aimed at the design, implemention, and validation of a multimedia framework that allows to adapt audio-visual content to a wide range of service scenarios. Both projects use XML-driven content adaptation. However, the scope of both projects is much broader than only media resource adaptation, since they also investigate context collection, advanced adaptation decision taking, etc. Key points where MuMiVA differentiates from these projects are:

- support for both MPEG-21 gBS Schema and MPEG-B BSDL, while ISIS and DANAE were focussed on MPEG-21 gBS Schema only;
- support for STX and SAX filters to implement streaming BSD transformations, while ISIS and DANAE only provide support for XSLT;
- a distributed architecture to achieve a scalable multimedia adaptation and delivery platform, while ISIS and DANAE were not focussed on scalability of their media resource adaptation platforms.

The Continuous Media Markup Language (CMML, [96]) allows to annotate and index continuous media files. The presented architecture is able to

extract temporal segments of media resources using a temporal URI scheme. A new file format is presented (i.e., Annodex), which enables encapsulation of any type of streamable media resource (i.e., coding-format independent). Annodex is based on the Ogg encapsulation format and is basically a bitstream consisting of media bitstreams combined with a CMML file. Comparing Annodex to NinSuna, we can state that both solutions allow to extract temporal fragments from media resources in a format-independent manner. However, Annodex requires the use of a single delivery format (i.e., Ogg) while NinSuna is able to deliver media content using any delivery format, in a format-independent way. Further, in contrast to our platforms, Annodex does not support structural adaptations (i.e., exploitation of scalability layers).

Digital Item Streaming (DIS, [64]) is part 18 of MPEG-21 and enables the incremental delivery of a Digital Item (covering both metadata and media resources) in a piece-wise fashion. DIS relies on the Bitstream Binding Language (BBL, [117]) for this purpose. BBL defines syntax and semantics to describe instructions on how a Digital Item can be fragmented and mapped into one or more delivery channels. It uses the same principles for serializing the packed media bitstream as NinSuna, i.e., MPEG-B BSDL is used to abstract the media bitstream and to enable the use of format-agnostic software modules. However, the BBL approach requires a new language to be used to specify the fragmentation and packetization process. Our proposed method to perform format-independent packaging only requires knowledge of commonly used XML transformation languages such as XSLT or STX. Furthermore, our model for media bitstreams provides support for the multimedia packaging process (i.e., timestamp support and coding-format specific parameters). Hence, this information can already be calculated during the metadata generation step, which is in contrast to the BBL approach where this information needs to be calculated during the packaging process.

Ransburg *et al.* propose to use *Media Streaming Instructions* within gBSDs to implement a generic streaming server [102]. More specifically, access units (i.e., the smallest unit of data to which timing may be attached) are identified and timestamps are assigned to them. Note that these Media Streaming Instructions have been adopted in the second amendment of the MPEG-21 DIA specification [65]. Using Media Streaming Instructions, the fragmentation process and timestamp calculation is performed during the gBSD generation step (i.e., during structural metadata generation). However, the fragmentation process is dependent on the delivery format (e.g., fragmentation of H.264/AVC streams is different for RTP and MP4 packetization). Also, gBSDs including Media Streaming Instructions are processed by delivery-format specific software modules (e.g., an RTP packetizer).

Smooth Streaming [86], which is an Internet Information Server (ISS) Media Services extension, enables adaptive streaming of multimedia content to Silverlight [85] clients over HTTP. Providing multiple encoded bitrates of the same media source allows Silverlight clients to seamlessly and dynamically switch between bitrates depending on network conditions and CPU power. Further, by using MP4 fragments combined with XML-based descriptions of the media content (called manifest files[9]), temporal media fragment selection can be applied. However, Smooth Streaming does not use a real streaming protocol, but a rather smart form of progressive download. Further, it only supports one delivery format (i.e., MP4) and four coding formats (i.e., H.264/AVC, VC-1, AAC, and WMA).

Finally, the Darwin Streaming Server[10] (DSS) is an open source, cross-platform RTP/RTSP streaming server. It provides a coding-format agnostic design, i.e., no codecs are present in the server. The streaming of media resources is guided by hint tracks, which contain all the information necessary to packetize and stream the media resource. Note that the creation of these hint tracks is coding-format specific (e.g., MP4Box[11] is a commonly used tool for the creation of hint tracks). Hint tracks can be compared to a part of our structural metadata (i.e., the mapping of timestamps to byte ranges of the media resource). However, support for adaptation operations is not available in DSS. Also, packing multimedia content with other delivery formats (other than RTP) is not possible.

5.6 Conclusions and Original Contributions

In this chapter, we have introduced two multimedia delivery platforms relying on format-agnostic software modules. First, we have discussed MuMiVA, which is a multimedia delivery platform relying on XML-driven content adaptation engines. MuMiVA tackles the diversity in the current multimedia landscape by streaming multimedia content that is adapted according to the constraints of a certain usage environment. The multimedia content is customized using format-agnostic adaptation engines which, in their turn, use MPEG-B BSDL and MPEG-21 gBS Schema as underlying technologies. An in-depth discussion of the architecture and functioning of our multimedia delivery platform has been provided. Furthermore, we have shown that MuMiVA is a fully integrated, extensible platform that is deployable in streaming environments.

[9] More technical information can be found on `http://alexzambelli.com/blog/2009/02/10/smooth-streaming-architecture/`

[10] `http://developer.apple.com/opensource/server/streaming/`

[11] `http://gpac.sourceforge.net/packager.php`

In order to demonstrate the flexibility of MuMiVA in terms of applications and coding formats, two different applications (i.e., exploitation of temporal scalability and shot selection) were applied to two different coding formats (i.e., MPEG-4 Visual and H.264/AVC). We identified the following factors that influence the performance in terms of CPU usage of the XML-driven content adaptation engine used in MuMiVA:

- *application*: one application can be more complex than another application and combining two applications can decrease the performance;
- *coding format*: the parsing process of coding formats differs in complexity;
- *level of detail of the gBSD*: the performance decreases when the level of detail of the gBSD increases.

Second, we presented NinSuna, which is a format-independent multimedia content adaptation and delivery platform that provides solutions for the shortcomings of our MuMiVA platform, i.e., interoperability problems of XML-driven content adaptation and the lack of a generic multimedia delivery solution. NinSuna is based on our model for media bitstreams, which covers the structural, content, and scalability properties of media bitstreams. Further, it provides support for a seamless integration of adaptation operations and content metadata, and supports format-independent packaging of multimedia content. Multimedia adaptation is performed by selecting and adapting portions of the structural metadata using SPARQL. Multimedia packaging is obtained by encapsulating the selected and adapted structural metadata within a specific delivery format. This packaging process is implemented using XML transformation filters and MPEG-B BSDL. We evaluated the NinSuna platform by enabling the user to select news fragments matching his/her specific interests and usage environment characteristics. The multimedia content, encoded with H.264/AVC and AAC, was delivered through RTP or MP4 according to the choice of the user. Both the generation of the structural metadata and the adaptation and delivery of news fragments can occur in real time. The performance of the media adaptation and delivery in terms of memory usage and latency depends on the delivery format.

Finally, we provided more insight into the synchronization issues caused by semantic adaptations. Three possible solutions were presented, i.e., one for MuMiVA and two for NinSuna. Within the MuMiVA platform, synchronization between adapted media bitstreams is implemented in the BSD transformation step. Within the NinSuna platform, the synchronization problem is solved during the structural metadata generation step by aligning the random access

units of the synchronized media bitstreams. Another possibility for the NinSuna platform would be to introduce a synchronization manager which is able to synchronize the adapted media bitstreams during the packaging process.

The research that has led to this chapter is also described in the following publications.

1. D. Van Deursen, S. De Bruyne, W. Van Lancker, W. De Neve, D. De Schrijver, H. Hellwagner, and R. Van de Walle. MuMiVA: a Multimedia Delivery Platform using Format-agnostic, XML-driven Content Adaptation. In *Proceedings of the 9th International Symposium on Multimedia*, pages 131–138, December 2007, Taichung, Taiwan

2. D. Van Deursen, W. Van Lancker, T. Paridaens, W. De Neve, E. Mannens, and R. Van de Walle. NinSuna: a Format-independent Multimedia Content Adaptation Platform based on Semantic Web Technologies. In *Proceedings of the 10th International Symposium on Multimedia*, pages 491–492, December 2008, Berkeley, United States

3. E. Mannens, R. Troncy, K. Braeckman, D. Van Deursen, W. Van Lancker, R. De Sutter, and R. Van de Walle. Automatic Information Enrichment in News Production. In *Proceedings of the 10th International Workshop on Image Analysis for Multimedia Interactive Services*, pages 61–64, May 2009, London, United Kingdom

4. D. Van Deursen, W. Van Lancker, W. De Neve, T. Paridaens, E. Mannens, and R. Van de Walle. Semantic Adaptation of Synchronized Multimedia Streams in a Format-independent Way. In *Proceedings of the 27th Picture Coding Symposium*, 4 pages on CD-ROM, May 2009, Chicago, United States

5. D. Van Deursen, W. Van Lancker, T. Paridaens, W. De Neve, E. Mannens, and R. Van de Walle. NinSuna: a Fully Integrated Platform for Format-independent Multimedia Content Adaptation and Delivery based on Semantic Web Technologies. Submitted to *Multimedia Tools and applications – Special Issue on Data Semantics for Multimedia Systems*

6. S. Coppens, E. Mannens, D. Van Deursen, and R. Van de Walle. Semantic Bricks for Performing Arts Archiving and Dissemination. Accepted for publication in *the Insurance Accounting & Systems Association (IASA) 2009 Annual Educational Conference and Business Show*

Chapter 6

Conclusions

I never think of the future - it comes soon enough.

Albert Einstein (1879 - 1955)

The multimedia landscape is characterized by a growing amount of multimedia content. This is due to ongoing digitization processes by public and private broadcasters, as well as to the growing popularity of user-generated content. There is also a large diversity in end-user devices that are able to consume multimedia. These devices differ in terms of device characteristics such as screen size, processing power, and battery life, and in terms of network characteristics, such as bandwidth, jitter, and error robustness. Furthermore, end-users with specific preferences often want to consume personalized versions of multimedia content (e.g., an end-user only requesting scenes satisfying his/her interests). Finally, the number of multimedia coding standards (e.g., MPEG-1 Audio, H.264/AVC, and JPEG2000) and multimedia delivery formats (e.g., MP4, Ogg, and RTP) has grown significantly over the last few years. Hence, the efficient delivery of multimedia content in today's world of ubiquitous multimedia consumption is an important technological challenge.

In order to obtain Universal Multimedia Access and thus providing (personalized) multimedia content anywhere, at anytime, and on any device, a transparent multimedia content adaptation and delivery approach is needed. In this context, metadata, which are generally defined as 'data about data', play a crucial role. Multimedia metadata enable the effective organization, access, and interpretation of multimedia content. Therefore, metadata have an increasingly important role in bringing order to the growing amount of available multimedia content. In this dissertation, we have tackled the aforementioned problems by using format-independent content adaptation and delivery

techniques. Furthermore, these techniques provide a seamless integration with today's manifold available multimedia metadata schemes.

In Chapter 2, we gave an overview of existing format-independent content adaptation techniques. These techniques rely on XML-based Bitstream Syntax Descriptions (BSDs), which contain information about the high-level structure of a media bitstream. They typically apply a three-step-based adaptation chain to obtain format-independent adaptation, i.e., BSD generation, BSD transformation, and adapted bitstream generation. Rather than directly operating on the binary bitstream, the actual adaptation is performed on the BSD level, enabling the use of already existing XML transformation technologies such as XSLT or STX. Note that a BSD is not meant to replace the original binary data; it rather acts as an additional layer, similar to metadata. Further in this chapter, we discussed how a BSD-based content adaptation chain can be implemented by means of generic, coding-format independent software modules. Examples of existing format-independent adaptation techniques following the three-step-based approach are MPEG-B BSDL, MPEG-21 gBS Schema, XFlavor, and BFlavor.

One important restriction of BSD-driven content adaptation techniques is that they can only perform high-level adaptation operations. Removing particular data blocks or modifying the value of certain syntax elements are considered as high-level adaptation operations. We identified two main target applications for BSD-driven content adaptation techniques, i.e., structural and semantic adaptations. Structural adaptations are typically performed to adapt media bitstreams by exploiting their scalability properties in order to meet the terminal and network characteristics of the end-user. Hence, structural adaptations are possible on condition that the media bitstreams consist of a number of scalability layers. Semantic adaptations are high-level adaptations based on semantic information about the multimedia content (e.g., selection of specific temporal segments that are of interest to the user). In this dissertation, we focussed on semantic adaptations along the temporal axis. Hence, semantic adaptations are possible on condition that media bitstreams are provided with random access points that are occurring in a regular way.

Also, in Chapter 2, we described a number of remaining challenges for format-independent content adaptation techniques, which were tackled in the following chapters of this dissertation:

- efficient and format-independent generation of generic Bitstream Syntax Descriptions (gBSDs);
- format-independent definition and implementation of high-level adaptation operations;

- integration with metadata standards;
- design and implementation of a fully integrated adaptation platform, based on format-independent content adaptation;
- format-independent packaging of adapted multimedia content;
- reduction of the structural metadata overhead.

In Chapter 3, we tackled a challenge that is specific for MPEG-21 gBS Schema, i.e., efficient and format-independent generation of generic Bitstream Syntax Descriptions (gBSDs). Within the MPEG-21 DIA specification, only the gBS Schema is described, together with the behaviour of a gBSDtoBin parser. This implies that a gBSD may be generated in any proprietary way. Therefore, we have proposed gBFlavor, which is an efficient tool for the generation of gBSDs in a format-independent manner. An existing format-independent solution is a two-step approach as currently proposed in the scientific literature, i.e., format-specific BSD generation, followed by a transformation in a gBSD. However, the latter solution requires knowledge of two different technologies (i.e., BSD generation and BSD-to-gBSD transformation). Furthermore, often more detail is needed in the BSD than in the resulting gBSD, implying a decrease in execution speed of the format-specific BSD generation process. These detailed BSDs are necessary for the BSD-to-gBSD transformation to produce an application-specific gBSD.

gBFlavor makes it possible to automatically generate a format-specific parser that is able to produce an application-specific gBSD for a given bitstream. Such a parser is generated by the gbflavorc translator, which takes as input a gBFlavor code. We have proposed the specification for this code, which describes the high-level structure of a particular coding format. The gBFlavor specification, which is an extension of the BFlavor specification, provides support for the addition of semantically meaningful information to gBSDs (by means of markers). Hence, gBSDs targeting specific applications can be obtained.

Further in Chapter 3, we have compared gBFlavor with the existing two-step approach. This comparison allowed getting an estimate of the expressive power and performance of a gBFlavor-enabled adaptation framework. The creation of gBSDs using gBFlavor avoids the two-step approach, since only one technology is needed to obtain gBSDs targeting a specific application. The exploitation of the scalable properties of two coding formats, i.e., SVC and JPEG2000, was used as the target application in our gBFlavor-enabled adaptation framework. Performance results have shown that gBFlavor outperforms the two-step approach in terms of execution times.

In order to solve a number of problems with XML-driven content adaptation (i.e., lack of support for defining high-level adaptation operations in a format-independent way and an ad-hoc integration with content metadata), we have presented a new format-independent adaptation technique in Chapter 4. It relies on a model for media bitstreams that takes into account the structural metadata, content metadata, and scalability information. The model is implemented using the Web Ontology Language (OWL). Existing coding formats are mapped to the structural part of the model, while existing metadata standards can be linked to the metadata content part of the model. Our new adaptation technique, which is called model-driven content adaptation, is based on executing SPARQL Protocol And RDF Query Language (SPARQL) queries over instances of the model for media bitstreams.

A comparison was made between model-driven content adaptation and existing format-independent and format-specific content adaptation techniques using different criteria. Compared to XML-driven content adaptation, advantages of model-driven content adaptation are the low amount of knowledge needed to define adaptation operations and the seamless integration with content metadata. Furthermore, both techniques have a comparable performance in terms of execution speed.

Two challenges were solved in Chapter 5: format-independent packaging of adapted multimedia content and design and implementation of fully integrated adaptation and delivery platforms, based on format-independent adaptation and packaging techniques. Therefore, we have introduced two multimedia delivery platforms relying on format-agnostic software modules: MuMiVA and NinSuna.

MuMiVA tackles the diversity in the current multimedia landscape by streaming multimedia content that is adapted according to the constraints of a certain usage environment. The multimedia content is customized using format-agnostic adaptation engines which, in their turn, use MPEG-B BSDL and MPEG-21 gBS Schema. We have discussed MuMiVA's architecture and functioning, and elaborated on its extensibility features. The platform was evaluated using two different adaptation operations (i.e., exploitation of temporal scalability and shot selection) applied to two different coding formats (i.e., MPEG-4 Visual and H.264/AVC). We concluded that the performance of MuMiVA in terms of CPU usage is not only dependent on the adaptation operation and coding format, but also on the level of detail of the (g)BSDs used.

NinSuna is a format-independent multimedia content adaptation and delivery platform that provides solutions for the shortcomings of MuMiVA, i.e., interoperability problems of XML-driven content adaptation and the lack of

a generic multimedia delivery solution. NinSuna is based on our model for media bitstreams, which was introduced in Chapter 4. It provides support for a seamless integration of adaptation operations and content metadata, and supports format-independent packaging of multimedia content. Multimedia adaptation is performed by selecting and adapting portions of the structural metadata using SPARQL. Multimedia packaging is obtained by encapsulating the selected and adapted structural metadata within a specific delivery format. This packaging process is implemented using XML transformation filters and MPEG-B BSDL. We evaluated the NinSuna platform by enabling the user to select news fragments matching his/her specific interests and usage environment characteristics. The multimedia content, encoded with H.264/AVC and AAC, can be delivered through RTP or MP4 according to the choice of the user. Both the generation of the structural metadata and the adaptation and delivery of news fragments can occur in real time. The performance of the media adaptation and delivery steps in terms of memory usage and latency depends on the delivery format.

In this dissertation, we have shown that format-independent media resource adaptation and delivery is feasible in practice. Multimedia content can be adapted and delivered in real-time using a feasible amount of memory. The generation of the structural metadata, which is less time-critical than the actual adaptation and delivery, is also characterized by reasonable execution times and memory consumption. Furthermore, in order to prove the practicability of format-independent content adaptation and delivery techniques in real-life scenarios, two fully integrated format-independent multimedia content adaptation and delivery platforms were developed in Chapter 5. However, the question remains whether such platforms will ever be used in real-world multimedia applications. In order to be able to answer this question in a positive way, a number of additional challenges need to be addressed. Some of them are already described in the scientific literature, while others are considered as future work.

Adaptation Possibilities

As discussed in this dissertation, format-independent content adaptation techniques are characterized by limited adaptation possibilities. More specifically, only high-level bitstream structures can be removed and only high-level syntax elements can be modified. Hence, compressed media bitstreams have to support adaptation operations that are possible without the need of a complete or partial recode process. Two options exist to handle this issue.

- *Adoption of scalable coding formats*: as discussed in Chapter 2, scalable coding is an interesting coding technique in the context of format-independent adaptation since it enables the extraction of multiple (lower quality) versions of the same media resource without the need of a complete recoding process. A significant number of scalable coding formats exists. However, thus far, scalable coding formats are rarely used in practice mainly due to their complexity and coding efficiency problems.

- *Hybrid adaptation systems*: in contrast to coding-format independent adaptation techniques, coding-format specific adaptation techniques are more flexible in terms of adaptation possibilities. They typically perform low-level (coding-format specific) adaptation operations such as adapting transform coefficients or converting from one coding format to another one (e.g., MPEG-2 Video to H.264/AVC [147]). Hence, coding-format specific and agnostic adaptation techniques are complementary. Therefore, it would be interesting to investigate in which way these two approaches can be integrated into one hybrid adaptation platform.

Structural Metadata Overhead

Next to limited adaptation possibilities, description-driven content adaptation introduces structural metadata, which results in file size overhead. In this dissertation, structural metadata was expressed by means of XML in case of XML-driven content adaptation and by means of RDF triples in case of model-driven content adaptation. Structural metadata in its textual form (i.e., uncompressed) is too verbose to be used in real-world multimedia applications. The following actions can be undertaken to tackle this problem.

- *Compressed structural metadata*: as already discussed in Chapter 2, XML-based structural metadata can be compressed by using algorithms such as WinZip, BiM, or XWRT. However, such an approach is only viable if XML transformations can be applied to the compressed form of the structural metadata. This way, we can avoid the uncompression (and possibly recompression) of the structural metadata during the adaptation process. An initial approach was introduced by Timmerer *et al.* in [120], where an approach is presented to transform XML documents, encoded with BiM [63], in the binary domain (as discussed in Chapter 2). In case of model-driven content adaptation, generic RDF storage solutions become insufficient in terms of scalability due to a high number of RDF triples representing the structural metadata. As discussed in Chapter 4, other solutions should be considered to store these RDF triples such as

an RDF store specifically designed for the model for media bitstreams. Hence, an RDBMS using a database scheme based on the structural and scalability part of the model for media bitstreams could serve as an efficient RDF store for model-driven content adaptation. RDBMSs should be capable of dealing with a large amount of structural metadata since they are mature, stable, and scalable, while also providing a high performance in terms of query execution speed.

- *Compact structural metadata*: besides compression, the structural metadata can be made more compact by removing information that is not necessary for the transformation or adapted bitstream generation process. In this dissertation, we applied this approach in two different ways. First, in Chapter 3, gBFlavor supports the insertion of markers within a gBSD, which results in a change in the hierarchical structure of the gBSD. More specifically, gBSD elements that do not include a marker, will not occur in the resulting gBSD since these are considered useless during the transformation and binarization steps. Second, in Chapter 4, the structural metadata are compliant to our model for media bitstreams. Hence, coding-format specific information within the structural metadata is avoided since it is now fully independent of the underlying coding format in terms of syntax elements and hierarchical structure. Note that in some specific cases, one can even go further to compact the structural metadata. As elaborated on in [32], coding formats such as JPEG XR and JPEG2000 are able to store an index table within the headers. Such an index table can be roughly compared to a binary representation of the structural metadata. Hence, parsing the actual data packets (i.e., tiles) is no longer necessary, because this information can be constructed from the main headers and the index table.

Adaptation Decisions

In this dissertation, we presented solutions to adapt and deliver media resources independent of the coding format. An important research domain is the logic that is needed to steer the adaptation engines. An interesting challenge is the integration of format-independent adaptation techniques and adaptation-decision taking logic. Two possibilities exist to take structural and semantic adaptation decisions.

- *Manual*: the end-user decides which adaptation needs to be applied to the media resource. For instance, he/she can select the bit rate of the media resource (structural adaptation) or he/she can select which scenes the resulting media resource should contain (semantic adaptation).

- *Automatic*: adaptations are taken by a so-called Adaptation-Decision Taking Engine (ADTE). In order to be able to take the proper decisions, the ADTE needs information about the end-user's preferences and its usage environment. The latter can for example be represented in MPEG-21 DIA [61]. For structural adaptation decisions, the ADTE typically compares the media resource characteristics (e.g., resolution) with the usage environment (e.g., screen size). A number of adaptation-decision algorithms have already been presented in the context of the MPEG-21 DIA framework [87, 88, 97, 98]. For semantic adaptation decisions, techniques such as content ranking and content recommendation can be applied to make proper decisions [5]. Note that with our introduction of Semantic Web technologies during the adaptation process (i.e., model-driven content adaptation), rule-based ADTEs could be integrated. More specifically, usage environment descriptions and bitstream characteristics can be expressed in RDF. This way, the adaptation-decision taking process can be steered by rules defined on top of these descriptions.

In-network Adaptations

The presented media resource adaptation and delivery platforms in Chapter 5 perform the adaptation operations at server-side. However, it would be interesting to investigate how these adaptation operations could be executed within network adaptation nodes. In [73], Kuschnig *et al.* presented an RTP/RTSP-based network adaptation node for SVC, based on MPEG-21 gBS Schema. In-network adaptation aims at minimizing adaptation delay by placing adaptation nodes close to the location where dynamically changing usage environments are expected (e.g., in the wireless access networks of the end-users). Another feature of in-network adaptation is to save bandwidth when multiple clients want to consume the same content. Hence, the content is delivered to the access network of the clients, where it is replicated for and adapted to the usage environment of each client. This way, bandwidth can be saved in the core network.

Next to Client-(Proxy-)Server architectures, Peer-to-Peer (P2P) networks are gaining importance to perform multimedia content streaming [92]. P2P networks avoid the bottleneck around a centralized server due to its distributed design and architecture [84]. For instance, streaming and adapting SVC encoded bitstreams over a P2P network is described in [8]. However, it would be interesting to investigate how P2P networks and format-independent content adaptation techniques could be integrated. For instance, in [54], the authors propose a P2P streaming solution based on MPEG-21 gBS Schema. Fur-

ther, the SEAmless Content Delivery (SEA, [109]) project aims to implement a context-aware networking delivery platform based on SVC, on-the-fly content adaptation, and P2P overlays.

Application Scenarios

Compared to format-specific adaptation techniques, format-independent adaptation techniques come with additional complexity. Whether format-independent content adaptation techniques will be adopted in real-world multimedia applications depends on, among other things, the presence of application scenarios that justify the additional complexity. In the early days of XML-driven content adaptation, these scenarios were hard to find, mainly because of two reasons.

- *Lack of adoption of scalable coding formats*: as elaborated on above, format-independent adaptation techniques are dependent on the adaptation possibilities of the underlying coding format. As scalable coding formats provide several adaptation possibilities along their different scalability axes, they are a rewarding coding technique to be used in combination with format-independent content adaptation.

- *Focus on structural adaptations*: the primary motivation behind the development of XML-driven content adaptation was the exploitation of scalability with metadata tools, i.e., structural adaptations. However, these are only possible if the (scalable) coding format provides support for these adaptation operations. Furthermore, scalable coding formats often come with low-complexity bitstream extractors which are able to perform exactly the same structural adaptations as XML-driven content adaptation.

However, a number of observations in the current multimedia landscape indicate an increasing importance of format-independent content adaptation in the near future.

- *Development of new scalable coding formats*: there is a growing interest in scalable coding techniques. Within the video coding community, a fully scalable coding format (i.e., SVC) was developed and standardized in July 2007. Further, Microsoft has developed a scalable image coding format (i.e., HD Photo) which will be standardized in the course of 2009 under the name JPEG XR. With the development of new scalable coding formats such as SVC and JPEG XR, new chances arise for scalable coding formats to be adopted.

- *Efficiency of adaptation engines*: compared to format-specific bitstream extractors, format-independent content adaptation has an advantage in terms of processing efficiency. Indeed, ordinary bitstream extractors have to parse the media resource each time it needs to be adapted. On the contrary, format-independent content adaptation only has to parse media resources once (i.e., during the structural metadata generation). Since the parsing process of newly developed coding formats becomes more and more complex (e.g., parsing an MPEG-1 Video stream is less complex than parsing an H.264/AVC stream), this form of caching applied by format-independent content adaptation will gain importance.

- *Semantic adaptations*: with an increasing interest in personalization of multimedia content, semantic adaptations are gaining importance. There is also a growing amount of content metadata that can assist in requesting personalized versions of media resources. Also, high-level semantic adaptation is one of the use cases of the W3C Media Fragments Working Group[1] (see also Annex D). Their mission is to address temporal and spatial media fragments in the Web using Uniform Resource Identifiers (URIs). It is important to remark that nearly every video coding format supports semantic adaptations along the temporal axis; the only condition is the occurrence of random access points in a regular way. This is in contrast to structural adaptations, which are heavily dependent on the abilities of the coding formats. Hence, thanks to their format-independent nature, format-independent content adaptation techniques are perfectly suited to perform semantic adaptations (combined with structural adaptations) for a wide range of coding formats. Furthermore, the adaptation technique presented in Chapter 4 (i.e., model-driven content adaptation) provides a seamless integration between content metadata and semantic adaptation operations. With the standardization of media fragment identifiers, semantic adaptation could result in a breakthrough for format-independent content adaptation.

To conclude, we hope that this dissertation convinced the reader that within the wide range of multimedia adaptation techniques, there is room for format-independent content adaptation. Furthermore, this dissertation can serve as a basis for future work towards the deployment of format-independent adaptation techniques in real-world multimedia applications.

[1] `http://www.w3.org/2008/WebVideo/Fragments/`

Appendix A

Syntax and BSD fragments

A.1 Introduction

In this annex, a number of syntax and BSD fragments for BSDL, gBS Schema, and XFlavor are listed. Furthermore, an XSLT and STX stylesheet are shown for the H.264/AVC coding format illustrating the removal of NAL units containing non-reference B slices (i.e., having nal_ref_idc equal to 0 and slice_type equal to 1 or 6). Note that these stylesheets need to be applied to BSDs generated using BSDL's BintoBSD parser with a BS Schema containing details up to and including the slice header.

A.2 MPEG-B BSDL

Listing A.1: Excerpt from a BS Schema for H.264/AVC.

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns:bs2="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS"
  xmlns:jvt="h264_avc" targetNamespace="h264_avc"
  elementFormDefault="qualified"
  bs2:rootElement="jvt:bitstream">

  <xsd:element name="bitstream">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="jvt:byte_stream_nal_unit"
            maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute ref="bs1:bitstreamURI"/>
```

```
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="byte_stream_nal_unit">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="zero_byte" type="
              xsd:unsignedByte" fixed="0" minOccurs="0"
              maxOccurs="unbounded" bs2:ifNext="000000"/>
          <xsd:element name="startcode" type="
              jvt:StartCodeType" fixed="000001" bs2:ifNext="
              000001"/>
          <xsd:element ref="jvt:nal_unit" minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="nal_unit">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="forbidden_zero_bit" type="jvt:b1
              " fixed="0"/>
          <xsd:element name="nal_ref_idc" type="jvt:b2"/>
          <xsd:element name="nal_unit_type" type="jvt:b5"/>
          <xsd:element name="raw_byte_sequence_payload"
              minOccurs="0" type="jvt:ByteRangeType"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

  </xsd:schema>
```

Listing A.2: Excerpt from a BSD for H.264/AVC, compliant with the BS Schema shown in Listing A.1.

```
<bitstream
  xmlns="h264_avc"
  bs1:bitstreamURI="city_300_4cif.264"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="h264_avc H264_AVC.xsd"
  xmlns:jvt="h264_avc">

  <byte_stream_nal_unit>
    <zero_byte>00</zero_byte>
    <startcode>000001</startcode>
    <nal_unit>
      <forbidden_zero_bit>0</forbidden_zero_bit>
```

```
          <nal_ref_idc>3</nal_ref_idc>
          <nal_unit_type>7</nal_unit_type>
          <raw_byte_sequence_payload>5 86</
              raw_byte_sequence_payload>
        </nal_unit>
      </byte_stream_nal_unit>

      <!-- ... -->

    </bitstream>
```

A.3 MPEG-21 gBS Schema

Listing A.3: Excerpt from a gBSD for H.264/AVC, compliant with gBS Schema.

```
<dia:DIA xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS" xmlns="
    urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS">
  <dia:DescriptionMetadata>
    <dia:ClassificationSchemeAlias alias="jvt" href="
        h264_avc"/>
  </dia:DescriptionMetadata>
  <dia:Description
    xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    bs1:bitstreamURI="city_300_4cif.264"
    xsi:type="gBSDType"
    addressUnit="bit"
    addressMode="absolute">
    <gBSDUnit syntacticalLabel=":jvt:bitstream" start="0">
      <gBSDUnit syntacticalLabel=":jvt:byte_stream_nal_unit
          " start="0">
        <gBSDUnit syntacticalLabel=":jvt:zero_byte" start="
            0" length="8"/>
        <gBSDUnit syntacticalLabel=":jvt:startcode" start="
            8" length="24"/>
        <gBSDUnit syntacticalLabel=":jvt:nal_unit" start="
            32">
          <Parameter name=":jvt:forbidden_zero_bit" start="
              32" length="1">
            <Value xsi:type="b1">0</Value>
          </Parameter>
          <Parameter name=":jvt:nal_ref_idc" start="33"
              length="2">
            <Value xsi:type="b2">3</Value>
          </Parameter>
```

```
            <Parameter name=":jvt:nal_unit_type" start="35"
                length="5">
              <Value xsi:type="b5">7</Value>
            </Parameter>
            <gBSDUnit syntacticalLabel="
                :jvt:raw_byte_sequence_payload" start="40"
                length="688"/>
          </gBSDUnit>
        </gBSDUnit>
      </gBSDUnit>
      <!-- ... -->
    </dia:Description>
  </dia:DIA>
```

A.4 XFlavor

Listing A.4: Excerpt from an XFlavor code for H.264/AVC.

```
class bitstream {
  while( nextbits(32)==0 ){
    Byte_stream_nal_unit byte_stream_nal_unit;
  }
}

class Byte_stream_nal_unit {
  while ( nextbits(24) != 0x000001 )
    bit(8) zero_byte = 0;
  bit(24) startcode = 0x000001;
  Nal_unit nal_unit;
}

class Nal_unit {
  bit(1) forbidden_zero_bit = 0;
  bit(2) nal_ref_idc;
  bit(5) nal_unit_type;
  Raw_byte_sequence_payload raw_byte_sequence_payload;
}

class Raw_byte_sequence_payload {
  // ...
}
```

Listing A.5: Excerpt from a BSD for H.264/AVC, compliant with the XFlavor code shown in Listing A.4.

```
<bitstream
  xmlns="http://www.ee.columbia.edu/flavor"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <byte_stream_nal_unit>
    <zero_byte bitLen="8">0</zero_byte>
    <startcode bitLen="24">1</startcode>
    <nal_unit>
      <forbidden_zero_bit bitLen="1">0</forbidden_zero_bit>
      <nal_ref_idc bitLen="2">3</nal_ref_idc>
      <nal_unit_type bitLen="5">7</nal_unit_type>
      <raw_byte_sequence_payload>
        <!-- ... -->
      </raw_byte_sequence_payload>
    </nal_unit>
  </byte_stream_nal_unit>
  <!-- ... -->
</bitstream>
```

A.5 Stylesheets

Listing A.6: Example of an XSLT stylesheet for H.264/AVC removing non-reference B slices.

```
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="h264_avc"
  xmlns:jvt="h264_avc">

  <xsl:output method="xml" indent="yes"/>

  <!-- drop non-referenced B slices -->
  <xsl:template match="jvt:byte_stream_nal_unit[
      jvt:nal_unit/jvt:nal_ref_idc = 0]">
    <xsl:variable name="slice_type" select="jvt:nal_unit/
        jvt:raw_byte_sequence_payload/jvt:slice/
        jvt:slice_header/jvt:slice_type"/>
    <xsl:if test="$slice_type != 1 and $slice_type != 6">
      <xsl:copy>
        <xsl:apply-templates/>
      </xsl:copy>
    </xsl:if>
    <!-- else: drop the NALU -->
```

```
    </xsl:template>

    <!-- default: pass-through -->
    <xsl:template match="@*|node()">
      <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
      </xsl:copy>
    </xsl:template>

  </xsl:stylesheet>
```

Listing A.7: Example of a STX stylesheet for H.264/AVC removing non-reference B slices.

```
<stx:transform
  xmlns:stx="http://stx.sourceforge.net/2002/ns"
  version="1.0"
  pass-through="all"
  strip-space="no"
  stxpath-default-namespace="h264_avc"
  output-method="xml">

  <stx:variable name="slice_type"/>
  <stx:variable name="isReference"/>

  <!-- drop non-reference B slices -->
  <stx:template match="byte_stream_nal_unit">
    <stx:buffer name="nalu_buffer">
      <stx:process-self group="nalu_group"/>
    </stx:buffer>
    <stx:if test="$isReference != 0 or ($slice_type != 1
        and $slice_type != 6)">
      <stx:process-buffer name="nalu_buffer" group="empty"/
          >
    </stx:if>
    <!-- else: drop the NALU -->
  </stx:template>

  <stx:group name="nalu_group">
    <stx:template match="nal_ref_idc">
      <stx:assign name="isReference" select="."/>
      <stx:process-self group="nalu_group"/>
    </stx:template>
    <stx:template match="slice_type">
      <stx:assign name="slice_type" select="."/>
      <stx:process-self group="nalu_group"/>
    </stx:template>
  </stx:group>
```

```
    <stx:group name="empty"/>
</stx:transform>
```

Appendix B

Automatic generation of gBSDs using BSDL and gBFlavor

B.1 BSD-to-gBSD Conversion for SVC Bitstreams

Listing B.1: Excerpt of a BS Schema for SVC.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Bit_stream">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="byte_stream_nal_unit" maxOccurs=
            "unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="zero_byte" type="
                  xsd:unsignedByte" fixed="0" minOccurs="0"
                  bs2:ifNext="000000"/>
              <xsd:element name="startcode" fixed="000001">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:hexBinary">
                    <xsd:length value="3"/>
                  </xsd:restriction>
                </xsd:simpleType>
              </xsd:element>
              <xsd:element ref="svc:nal_unit" minOccurs="0"
                  />
            </xsd:sequence>
          </xsd:complexType>
```

```
          </xsd:element>
        </xsd:sequence>
        <xsd:attribute ref="bs1:bitstreamURI"/>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="nal_unit">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="forbidden_zero_bit" type="svc:b1
              " fixed="0"/>
          <xsd:element name="nal_ref_idc" type="svc:b2"/>
          <xsd:element name="nal_unit_type" type="svc:b5"/>
          <xsd:element ref="svc:raw_byte_sequence_payload"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <!-- ... -->
</xsd:schema>
```

Listing B.2: Excerpt of a STX stylesheet for transforming BSDs into gBSDs (with SVC as underlying coding format).

```
<stx:transform pass-through="none" strip-space="no" output-
    method="xml">
  <stx:variable name="prevEnd" select="0"/>
  <stx:variable name="currentSize" select="0"/>
  <stx:template match="Bit_stream">
    <stx:element name="gBSDUnit">
      <stx:attribute name="syntacticalLabel">
          :svc:Bit_stream</stx:attribute>
      <stx:attribute name="start">0</stx:attribute>
      <stx:process-children group="byte_stream_nal_unit"/>
    </stx:element>
  </stx:template>
  <stx:group name="byte_stream_nal_unit">
    <stx:template match="zero_byte">
      <stx:assign name="currentSize" select="$currentSize +
          8"/>
    </stx:template>
    <stx:template match="startcode">
      <stx:assign name="currentSize" select="$currentSize +
          24"/>
    </stx:template>
    <stx:template match="nal_unit_type">
      <stx:assign name="currentSize" select="$currentSize +
          8"/>
    </stx:template>
```

```
      <stx:template match="
          slice_layer_without_partitioning_rbsp">
        <stx:process-children group="slice"/>
        <stx:assign name="prevEnd" select="$prevEnd + $
            currentSize"/>
        <stx:assign name="currentSize" select="0"/>
      </stx:template>
      <!-- ... -->
    </stx:group>
    <stx:group name="slice">
      <stx:template match="slice_payload">
        <stx:variable name="pstart" select="substring-before
            (., ' ')"/>
        <stx:variable name="plength" select="substring-after
            (., ' ') "/>
        <stx:variable name="currentSize" select="$currentSize
             + ($plength*8)"/>
        <stx:element name="gBSDUnit">
          <stx:attribute name="syntacticalLabel">:svc:bsnu</
              stx:attribute>
          <stx:attribute name="start"><stx:value-of select="$
              prevEnd"/></stx:attribute>
          <stx:attribute name="length"><stx:value-of select="
              $currentSize"/></stx:attribute>
        </stx:element>
      </stx:template>
    </stx:group>
    <!-- ... -->
  </stx:transform>
```

B.2 gBFlavor Code for SVC

Listing B.3: Excerpt of a gBFlavor code for SVC [143].

```
/*************************
 * High-level syntax code *
 *************************/

%targetns{jsvm8%targetns}
%ns{jsvm%ns}
%root{BitStream%root}
%emulationBytes{(000003, 0000);%emulationBytes}
%emulateBytes{(000001, 00000301);%emulateBytes}

%context{seq_parameter_set_id%context}
```

```
class SeqParameterSet{
  //...
  ue() seq_parameter_set_id;
  //...
}

class SliceHeaderInScalableExtension{
  ue() first_mb_in_slice;
  ue() slice_type;
  ue() pic_parameter_set_id;
  bit(getcontext("Seq_parameter_set_rbsp",
    getcontext("Pic_parameter_set_rbsp",
        pic_parameter_set_id, $seq_parameter_set_id),
    $log2_max_frame_num_minus4) + 4) frame_num;
  align();
  varByteRange() slice_payload = 0x000001;
}

%context{0%context}
class SubSeqInfo{
  ue() sub_seq_layer_num;
  ue() sub_seq_id;
  //...
}

class RawByteSequencePayload (int nal_unit_type){
  switch(nal_unit_type){
    case 1:
    case 5:
      SliceLayer sliceLayer;
      break;
    case 6:
      Sei sei;
      break;
    case 7:
      SeqParameterSet seqParameterSet;
      break;
    case 8:
      PicParameterSet picParameterSet;
      break;
    case 20 .. 21:
      SliceLayerInScalableExtension
          sliceLayerInScalableExtension;
      break;
    //...
  }
}

class NalUnitHeaderSvcExtension{
```

```
    //...
    bit(3) temporal_level;
    bit(3) dependency_id;
    bit(2) quality_level;
    //...
  }

  class NalUnit{
    bit(1) forbidden_zero_bit = 0;
    bit(2) nal_ref_idc;
    bit(5) nal_unit_type;
    if(nal_unit_type == 20 || nal_unit_type == 21)
      NalUnitHeaderSvcExtension nalUnitHeaderSvcExtension;
    RawByteSequencePayload rawByteSequencePayload(
        nal_unit_type);
  }

  class ByteStreamNalUnit {
    //...
    fixedByteRange(3) startcode = 0x000001;
    NalUnit nalUnit;
    //...
  }

  class BitStream{
    while(1)
      ByteStreamNalUnit byteStreamNalUnit;
  }


  /*****************************
   * Application-specific code *
   *****************************/

  gBSDApp TemporalSpatialQuality_Scalability {

    //assign markers to slice NALUs occurring in the base
        layer
    class NalUnit {
      int temp_level;
      if (nal_unit_type == 1 || nal_unit_type == 5){
        int temp_level = getcontext("SubSeqInfo", 0,
            sub_seq_layer_num);
        setmarker("ByteStreamNalUnit", "", "TL=" + temp_level
             + ":SL=0:QL=0");
      }
    }
```

```
    //assign markers to slice NALUs occurring in the
        enhancement layers
    class NalUnitHeaderSvcExtension {
105   int temp_level, spat_level, qual_level;
      //interpret temporal_level, dependency_id, and
          quality_level to
      //fill in the variables ...
      setmarker("ByteStreamNalUnit", "", "TL=" + temp_level +
           ":SL=" + spat_level + ":QL=" + qual_level);
    }
110 }
```

Listing B.4: Excerpt of a gBSD, generated by gBFlavor and using the high-level syntax code given in Listing B.3 (no application is specified).

```
 1  <gBSDUnit syntacticalLabel=":jsvm:bitStream" start="0">
      <!-- ... -->
      <gBSDUnit syntacticalLabel=":jsvm:byteStreamNalUnit"
          start="53776">
        <!-- ... -->
 5      <gBSDUnit syntacticalLabel=":jsvm:startcode" start="
            53784" length="24"/>
        <gBSDUnit syntacticalLabel=":jsvm:nalUnit" start="53808
            ">
          <Parameter name=":jsvm:forbidden_zero_bit" start="
              53808" length="1">
            <Value xsi:type="b1">0</Value>
          </Parameter>
10        <Parameter name=":jsvm:nal_ref_idc" start="53809"
              length="2">
            <Value xsi:type="b2">3</Value>
          </Parameter>
          <Parameter name=":jsvm:nal_unit_type" start="53811"
              length="5">
            <Value xsi:type="b5">21</Value>
15        </Parameter>
          <gBSDUnit syntacticalLabel=":jsvm:
              nalUnitHeaderSvcExtension" start="53816">
            <!-- ... -->
            <Parameter name=":jsvm:temporal_level" start="53824
                " length="3">
              <Value xsi:type="b3">0</Value>
20          </Parameter>
            <Parameter name=":jsvm:dependency_id" start="53827"
                length="3">
              <Value xsi:type="b3">0</Value>
            </Parameter>
```

```
            <Parameter name=":jsvm:quality_level" start="53830"
                 length="2">
              <Value xsi:type="b2">1</Value>
            </Parameter>
            <!-- ... -->
          </gBSDUnit>
          <gBSDUnit syntacticalLabel=":jsvm:
              rawByteSequencePayload" start="53840">
            <gBSDUnit syntacticalLabel=":jsvm:
                sliceLayerInScalableExtension" start="53840">
              <gBSDUnit syntacticalLabel=":jsvm:
                  sliceHeaderInScalableExtension" start="53840">
                <Parameter name=":jsvm:first_mb_in_slice" start
                    ="53840" length="1">
                  <Value xsi:type="bs1:expGolomb">1</Value>
                </Parameter>
                <Parameter name=":jsvm:slice_type" start="53841
                    ">
                  <Value xsi:type="bs1:expGolomb">4</Value>
                </Parameter>
                <Parameter name=":jsvm:pic_parameter_set_id"
                    start="53846">
                  <Value xsi:type="bs1:expGolomb">4</Value>
                </Parameter>
                <Parameter name=":jsvm:frame_num" start="53851"
                     length="9">
                  <Value xsi:type="b9">0</Value>
                </Parameter>
                <!-- ... -->
                <Parameter name=":jsvm:stuffing" start="53897">
                  <Value xsi:type="bs1:align8">0</Value>
                </Parameter>
                <gBSDUnit syntacticalLabel=":jsvm:slice_payload
                    " start="53904" length="58584"/>
              </gBSDUnit>
            </gBSDUnit>
          </gBSDUnit>
        </gBSDUnit>
      </gBSDUnit>
      <!-- ... -->
</gBSDUnit>
```

Appendix C

Multimedia model and RDF instances

C.1 Model for Media Bitstreams

In Listing C.1, the full model for media bitstreams is provided, implemented using OWL. The result is an OWL DL ontology counting respectively 13, 9, and 15 classes, object properties, and data properties.

Listing C.1: The model for media bitstreams implemented using OWL.

```
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
]>

<rdf:RDF xmlns="http://multimedialab.elis.ugent.be/
    ontologies/multimedia_model.owl#"
  xml:base="http://multimedialab.elis.ugent.be/ontologies/
     multimedia_model.owl"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <owl:Ontology rdf:about=""/>

  <!--////////////////////////////////////////
  // Object Properties
  ////////////////////////////////////////-->
  <owl:ObjectProperty rdf:about="#boundTo">
    <rdfs:domain rdf:resource="#Feature"/>
    <rdfs:range rdf:resource="#ScalabilityAxis"/>
```

```
20    </owl:ObjectProperty>
      <owl:ObjectProperty rdf:about="#constrains">
        <rdfs:range rdf:resource="#ScalabilityAxis"/>
        <rdfs:domain rdf:resource="#ScalabilityAxisInfo"/>
      </owl:ObjectProperty>
25    <owl:ObjectProperty rdf:about="#hasBitstreamData">
        <rdfs:range rdf:resource="#RandomAccessUnit"/>
        <rdfs:domain rdf:resource="#TemporalSegment"/>
      </owl:ObjectProperty>
      <owl:ObjectProperty rdf:about="#hasFeature">
30      <rdfs:range rdf:resource="#Feature"/>
        <rdfs:domain rdf:resource="#MediaBitstream"/>
      </owl:ObjectProperty>
      <owl:ObjectProperty rdf:about="#hasFeatureValue">
        <rdfs:domain rdf:resource="#Feature"/>
35      <rdfs:range rdf:resource="#FeatureValue"/>
      </owl:ObjectProperty>
      <owl:ObjectProperty rdf:about="#hasScalabilityInfo">
        <rdfs:range rdf:resource="#ScalabilityAxisInfo"/>
        <rdfs:domain>
40        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
              <rdf:Description rdf:about="#DataBlock"/>
              <rdf:Description rdf:about="#FeatureValue"/>
            </owl:unionOf>
45        </owl:Class>
        </rdfs:domain>
      </owl:ObjectProperty>
      <owl:ObjectProperty rdf:about="#hasStructure">
        <rdf:type rdf:resource="&owl;TransitiveProperty"/>
50      <rdfs:range>
          <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
              <rdf:Description rdf:about="#DataBlock"/>
              <rdf:Description rdf:about="#RandomAccessUnit"/>
55          </owl:unionOf>
          </owl:Class>
        </rdfs:range>
        <rdfs:domain>
          <owl:Class>
60          <owl:unionOf rdf:parseType="Collection">
              <rdf:Description rdf:about="#DataBlock"/>
              <rdf:Description rdf:about="#MediaBitstream"/>
              <rdf:Description rdf:about="#RandomAccessUnit"/>
            </owl:unionOf>
65        </owl:Class>
        </rdfs:domain>
      </owl:ObjectProperty>
      <owl:ObjectProperty rdf:about="#hasTemporalSegment">
```

```
    <rdfs:range rdf:resource="#TemporalSegment"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <rdf:Description rdf:about="#AnnotatedMultimedia"
              />
          <rdf:Description rdf:about="#TemporalSegment"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#isRepresentedBy">
    <rdfs:domain rdf:resource="#AnnotatedMultimedia"/>
    <rdfs:range rdf:resource="#MediaBitstream"/>
  </owl:ObjectProperty>

  <!--///////////////////////////////////////////
  // Data properties
  ///////////////////////////////////////////-->
  <owl:DatatypeProperty rdf:about="#format">
    <rdfs:domain rdf:resource="#MediaBitstream"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#codingDescription">
    <rdfs:domain rdf:resource="#MediaBitstream"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#description">
    <rdfs:domain rdf:resource="#AnnotatedMultimedia"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#duration">
    <rdfs:range rdf:resource="&xsd;duration"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <rdf:Description rdf:about="#AnnotatedMultimedia"
              />
          <rdf:Description rdf:about="#TemporalSegment"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#keyword">
    <rdfs:range rdf:resource="&xsd;string"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <rdf:Description rdf:about="#AnnotatedMultimedia"
```

```
                />
              <rdf:Description rdf:about="#TemporalSegment"/>
            </owl:unionOf>
          </owl:Class>
        </rdfs:domain>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="#level">
      <rdfs:domain rdf:resource="#ScalabilityAxisInfo"/>
      <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="#nStuffingBytes">
      <rdfs:domain rdf:resource="#StuffingBits"/>
      <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="#nlevels">
      <rdfs:domain rdf:resource="#ScalabilityAxis"/>
      <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="#start">
      <rdfs:range rdf:resource="&xsd;time"/>
      <rdfs:domain>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="#AnnotatedMultimedia"
                />
            <rdf:Description rdf:about="#TemporalSegment"/>
          </owl:unionOf>
        </owl:Class>
      </rdfs:domain>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="#syntaxElementValue">
      <rdfs:domain rdf:resource="#SyntaxElement"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="#timestamp">
      <rdfs:domain rdf:resource="#DataBlock"/>
      <rdfs:range rdf:resource="&xsd;long"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="#timestampRate">
      <rdfs:domain rdf:resource="#MediaBitstream"/>
      <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="#unit">
      <rdfs:domain rdf:resource="#Feature"/>
      <rdfs:range rdf:resource="&xsd;string"/>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty rdf:about="#value">
      <rdfs:domain rdf:resource="#FeatureValue"/>
    </owl:DatatypeProperty>
```

```
    <!--///////////////////////////////////////////
    // Classes
    ///////////////////////////////////////////-->
    <owl:Class rdf:about="#AnnotatedMultimedia">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>
    <owl:Class rdf:about="#DataBlock">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>
    <owl:Class rdf:about="#Feature">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>
    <owl:Class rdf:about="#FeatureValue">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>
    <owl:Class rdf:about="#MediaBitstream">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>
    <owl:Class rdf:about="#RandomAccessUnit">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>
    <owl:Class rdf:about="#ScalabilityAxis">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>
    <owl:Class rdf:about="#ScalabilityAxisInfo">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>
    <owl:Class rdf:about="#StuffingBits">
      <rdfs:subClassOf rdf:resource="#DataBlock"/>
    </owl:Class>
    <owl:Class rdf:about="#SyntaxElement">
      <rdfs:subClassOf rdf:resource="#DataBlock"/>
    </owl:Class>
    <owl:Class rdf:about="#TemporalSegment">
      <rdfs:subClassOf rdf:resource="&owl;Thing"/>
    </owl:Class>
    <owl:Class rdf:about="#TruncatablePayload">
      <rdfs:subClassOf rdf:resource="#DataBlock"/>
    </owl:Class>
  </rdf:RDF>
```

C.2 RDF Instances Compliant to the Multimedia Model

An excerpt of RDF triples describing scalability information, structural metadata, and content metadata is shown in Listing C.2. The underlying media bitstream is encoded with H.264/AVC. First, an instance of the class *MediaBitstream* is created (lines 9-21). It contains one *Feature* (i.e., frame rate) which is linked to the temporal scalability axis (line 95). The media bitstream points directly to two *DataBlock*s corresponding to the sequence and picture parameter set (lines 16-17). It also points to the *RandomAccessUnit*s (lines 18-19). A random access unit consists of a list of data blocks (lines 23-33) containing a *start* and *length* property, together with scalability information. For example, the data block `http://www.foo.be/nieuws_avc.rdf#db_2` is located in the temporal base layer (i.e., level 0 of the temporal scalability axis) (line 49). The content metadata consists of an instance of *AnnotatedMultimedia* containing a list of temporal segments (lines 115-121). Each temporal segment has a *start*, a *duration*, and optionally a *keyword* property. Further, the link to the structural metadata is expressed by the *hasBitstreamData* property (lines 127-128).

Listing C.2: Excerpt of resulting RDF triples compliant with the model for media bitstreams. The underlying H.264/AVC bitstream represents a news sequence.

```
1  <rdf:RDF
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:mmm="http://multimedialab.elis.ugent.be/ontologies/
         multimedia_model.owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
5    xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     >

     <mmm:MultimediaBitstream rdf:about="http://www.foo.be/
         nieuws_avc.rdf#mmb">
10     <mmm:bitstreamSource rdf:resource="http://www.foo.be/
           nieuws.264"/>
       <mmm:hasFeature rdf:resource="http://www.foo.be/
           nieuws_avc.rdf#feat_frame-rate"/>
       <mmm:format>video/H264</mmm:format>
       <rdfs:label>nieuws_avc</rdfs:label>
       <mmm:codecDescription>H.264/AVC</mmm:codecDescription>
15     <mmm:timestampRatePerSecond>25.0</
           mmm:timestampRatePerSecond>
       <mmm:hasStructure rdf:resource="http://www.foo.be/
           nieuws_avc.rdf#db_0"/>
```

```
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#db_1"/>
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#rau_0"/>
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#rau_1/>
    <!-- ... other RAUs ... -->
</mmm:MultimediaBitstream>

<mmm:RandomAccessUnit rdf:about="http://www.foo.be/
    nieuws_avc.rdf#rau_0">
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#db_2"/>
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#db_3"/>
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#db_4"/>
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#db_5"/>
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#db_6"/>
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#db_7"/>
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#db_8"/>
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#db_9"/>
    <mmm:hasStructure rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#db_10"/>
</mmm:RandomAccessUnit>

<mmm:DataBlock rdf:about="http://www.foo.be/nieuws_avc.
    rdf#db_0">
    <mmm:start>3592</mmm:start>
    <mmm:length>224</mmm:length>
    <mmm:timestamp>0</mmm:timestamp>
</mmm:DataBlock>
<mmm:DataBlock rdf:about="http://www.foo.be/nieuws_avc.
    rdf#db_1">
    <mmm:start>3816</mmm:start>
    <mmm:length>64</mmm:length>
    <mmm:timestamp>0</mmm:timestamp>
</mmm:DataBlock>

<mmm:DataBlock rdf:about="http://www.foo.be/nieuws_avc.
    rdf#db_2">
    <mmm:start>3880</mmm:start>
    <mmm:length>4800</mmm:length>
    <mmm:hasScalabilityInfo rdf:resource="http://www.foo.be
```

```
        /nieuws_avc.rdf#si_temp_0"/>
50    <mmm:timestamp>0</mmm:timestamp>
    </mmm:DataBlock>
    <mmm:DataBlock rdf:about="http://www.foo.be/nieuws_avc.
        rdf#db_3">
      <mmm:start>8680</mmm:start>
      <mmm:length>192</mmm:length>
55    <mmm:hasScalabilityInfo rdf:resource="http://www.foo.be
        /nieuws_avc.rdf#si_temp_0"/>
      <mmm:timestamp>4</mmm:timestamp>
    </mmm:DataBlock>
    <mmm:DataBlock rdf:about="http://www.foo.be/nieuws_avc.
        rdf#db_4">
      <mmm:start>8872</mmm:start>
60    <mmm:length>176</mmm:length>
      <mmm:hasScalabilityInfo rdf:resource="http://www.foo.be
        /nieuws_avc.rdf#si_temp_1"/>
      <mmm:timestamp>2</mmm:timestamp>
    </mmm:DataBlock>
    <mmm:DataBlock rdf:about="http://www.foo.be/nieuws_avc.
        rdf#db_5">
65    <mmm:start>9048</mmm:start>
      <mmm:length>184</mmm:length>
      <mmm:hasScalabilityInfo rdf:resource="http://www.foo.be
        /nieuws_avc.rdf#si_temp_2"/>
      <mmm:timestamp>1</mmm:timestamp>
    </mmm:DataBlock>
70  <mmm:DataBlock rdf:about="http://www.foo.be/nieuws_avc.
        rdf#db_6">
      <mmm:start>9232</mmm:start>
      <mmm:length>184</mmm:length>
      <mmm:hasScalabilityInfo rdf:resource="http://www.foo.be
        /nieuws_avc.rdf#si_temp_2"/>
      <mmm:timestamp>3</mmm:timestamp>
75  </mmm:DataBlock>

    <mmm:ScalabilityAxis rdf:about="http://www.foo.be/
        nieuws_avc.rdf#sa_temp">
      <rdfs:label>temporal</rdfs:label>
      <mmm:nlevels>3</mmm:nlevels>
80  </mmm:ScalabilityAxis>
    <mmm:ScalabilityAxisInfo rdf:about="http://www.foo.be/
        nieuws_avc.rdf#si_temp_0">
      <mmm:constrains rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#sa_temp"/>
      <mmm:level>0</mmm:level>
    </mmm:ScalabilityAxisInfo>
85  <mmm:ScalabilityAxisInfo rdf:about="http://www.foo.be/
        nieuws_avc.rdf#si_temp_1">
```

```
    <mmm:constrains rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#sa_temp"/>
    <mmm:level>1</mmm:level>
  </mmm:ScalabilityAxisInfo>
  <mmm:ScalabilityAxisInfo rdf:about="http://www.foo.be/
      nieuws_avc.rdf#si_temp_2">
    <mmm:constrains rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#sa_temp"/>
    <mmm:level>2</mmm:level>
  </mmm:ScalabilityAxisInfo>

  <mmm:Feature rdf:about="http://www.foo.be/nieuws_avc.rdf#
      feat_frame-rate">
    <mmm:boundTo rdf:resource="http://www.foo.be/nieuws_avc
        .rdf#sa_temp"/>
    <mmm:hasFeatureValue rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#featvalue_0"/>
    <mmm:hasFeatureValue rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#featvalue_1"/>
    <mmm:hasFeatureValue rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#featvalue_2"/>
    <rdfs:label>frame rate</rdfs:label>
    <mmm:unit>fps</mmm:unit>
  </mmm:Feature>
  <mmm:FeatureValue rdf:about="http://www.foo.be/nieuws_avc
      .rdf#featvalue_0">
    <mmm:hasScalabilityInfo rdf:resource="http://www.foo.be
        /nieuws_avc.rdf#si_temp_2"/>
    <mmm:value>25.0</mmm:value>
  </mmm:FeatureValue>
  <mmm:FeatureValue rdf:about="http://www.foo.be/nieuws_avc
      .rdf#featvalue_1">
    <mmm:hasScalabilityInfo rdf:resource="http://www.foo.be
        /nieuws_avc.rdf#si_temp_1"/>
    <mmm:value>12.5</mmm:value>
  </mmm:FeatureValue>
  <mmm:FeatureValue rdf:about="http://www.foo.be/nieuws_avc
      .rdf#featvalue_2">
    <mmm:hasScalabilityInfo rdf:resource="http://www.foo.be
        /nieuws_avc.rdf#si_temp_0"/>
    <mmm:value>6.25</mmm:value>
  </mmm:FeatureValue>

  <mmm:AnnotatedMultimedia rdf:about="http://www.foo.be/
      nieuws_avc.rdf#nieuws_annotation">
    <mmm:description>News annotation</mmm:description>
    <mmm:isRepresentedBy rdf:resource="http://www.foo.be/
        nieuws_avc.rdf#mmb"/>
    <mmm:hasTemporalSegment rdf:resource="http://www.foo.be
```

```
          /nieuws_avc.rdf#ts_0"/>
        <mmm:hasTemporalSegment rdf:resource="http://www.foo.be
          /nieuws_avc.rdf#ts_1"/>
120     <!-- ... other temporal segments ... -->
      </mmm:AnnotatedMultimedia>

      <mmm:TemporalSegment rdf:about="http://www.foo.be/
        nieuws_avc.rdf#ts_0">
        <mmm:segmentStart>T00:00:00:00F25</mmm:segmentStart>
125     <mmm:segmentDuration>PT10S0N25F</mmm:segmentDuration>
        <mmm:keyword>intro</mmm:keyword>
        <mmm:hasBitstreamData rdf:resource="http://www.foo.be/
          nieuws.264#rau_0"/>
        <mmm:hasBitstreamData rdf:resource="http://www.foo.be/
          nieuws.264#rau_1"/>
        <!-- ... other RAUs -->
130   </mmm:TemporalSegment>
    </rdf:RDF>
```

Listing C.3: Excerpt of resulting RDF triples after execution of the SPARQL query listed in Listing 4.1), applied to the triples listed in Listing C.2.

```
1   <rdf:RDF
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:mmm="http://multimedialab.elis.ugent.be/ontologies/
        multimedia_model.owl#"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
5     xmlns:owl="http://www.w3.org/2002/07/owl#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      >

      <mmm:DataBlock rdf:about="http://www.foo.be/nieuws_avc.
        rdf#db_0">
10      <mmm:start>3592</mmm:start>
        <mmm:length>224</mmm:length>
        <mmm:timestamp>0</mmm:timestamp>
      </mmm:DataBlock>
      <mmm:DataBlock rdf:about="http://www.foo.be/nieuws_avc.
        rdf#db_1">
15      <mmm:start>3816</mmm:start>
        <mmm:length>64</mmm:length>
        <mmm:timestamp>0</mmm:timestamp>
      </mmm:DataBlock>

20    <mmm:DataBlock rdf:about="http://www.foo.be/nieuws_avc.
        rdf#db_2">
        <mmm:start>3880</mmm:start>
        <mmm:length>4800</mmm:length>
```

```
      <mmm:timestamp>0</mmm:timestamp>
    </mmm:DataBlock>
    <mmm:DataBlock rdf:about="http://www.foo.be/nieuws_avc.
        rdf#db_3">
      <mmm:start>8680</mmm:start>
      <mmm:length>192</mmm:length>
      <mmm:timestamp>4</mmm:timestamp>
    </mmm:DataBlock>

  </rdf:RDF>
```

Appendix D

W3C Media Fragments Working Group

D.1 Introduction

The W3C Media Fragments Working Group[1] (MFWG) is part of W3C's Video in the Web activity[2], which originated from the workshop on Video in the Web[3]. The goal of the Video in the Web activity is to make video a "first class citizen" on the Web. In this context, Video in the Web does not only include video, but also audio and still images. Because of the explosive growth of media on the Web, challenges such as content discovery, searching, indexing, and accessibility are appearing. Enabling users (from individuals to large organizations) to put media on the Web requires the development of a solid architectural foundation that enables people to create, navigate, search, link, consume, and distribute media resources. This way, media resources are effectively part of the Web instead of an extension that does not take full advantage of the Web architecture.

The mission of the MFWG, in which the author is actively participating, is to address temporal and spatial media fragments on the Web using Uniform Resource Identifiers (URIs, [11]). Having global identifiers for arbitrary media fragments would allow substantial benefits, including linking, bookmarking, caching, and indexing. For example, one could be able to point to the 20th second of a news video sequence, bringing you directly to a specific news item within the news sequence. The main tasks of the MFWG can be summarized as follows:

[1] `http://www.w3.org/2008/WebVideo/Fragments/`
[2] `http://www.w3.org/2008/WebVideo/`
[3] `http://www.w3.org/2007/08/video/`

- to develop a mechanism to uniquely identify temporal and/or spatial fragments within a media resource that is independent of underlying coding formats;
- to investigate the delivery of the requested media resource to allow full or partial media retrieval using at least the HTTP protocol;
- to provide a partial mapping between the developed URI syntax and existing ways of defining temporal and/or spatial fragments.

In this Annex, we elaborate on how format-independent adaptation logic could be used within the W3C MFWG. More specifically, we discuss the implementation of the media resource adaptation use case using model-driven content adaptation.

D.2 Media Resource Adaptation Use Case

One of the use cases identified by the MFWG is media resource adaptation [123]. In this use case, a client attempts to retrieve a media fragment using a URI-based syntax. An example of such a URI could be `http://foo.com/media.mp4#t=0,10`, which identifies the first ten seconds of the media resource `http://foo.com/media.mp4`. Of course, the client often has the desire not to retrieve the full resource, but only the identified subpart. In our example, only the first ten seconds of the requested media resource should be delivered to the client.

To save bandwidth and processing power at client-side, the extraction of fragments from media resources preferrably occurs at the server (although this is not mandatory). Note that in Chapter 2, we have defined such media adaptations as *semantic adaptations*. The use case describes three scenarios, each discussing the media fragment delivery along a different axis:

- *temporal*: select a specific sport fragment from a news video;
- *spatial*: select a region corresponding to one particular person from a family picture;
- *track*: select the audio track from a multimedia resource.

Starting from various use cases, a number of requirements were identified that need to be implemented by the MFWG [123]. A subset of these requirements will provide more insight in the kind of adaptation methods that can be used to implement the media resource adaptation use case:

- *Format-independent*: the media fragment URI scheme has to be independent of any coding and delivery format;
- *Unique resource*: media fragments need to be specified as a secondary resource for the complete primary media resource;
- *Valid resource*: media fragments need to be valid media resources by themselves (and can thus be played back by existing media players);
- *Avoid recompression*: media fragments need to be delivered as actual subresources of the media resource, implying that it is preferrable to not decode and recompress media resources in order to create media fragments.

From these four requirements, we can conclude that media fragments can be obtained by using adaptation methods only performing the removal of high-level bitstream structures and the modification of high-level syntax elements. As a consequence, the compressed media bitstreams need to be encoded in such a way that it is possible to perform media fragment extraction without the need of a complete or partial recode process. In Chapter 2, Section 2.3.2, we discussed how semantic adaptations along the temporal axis could be established. To support the extraction of spatial fragments, (interactive) Region-Of-Interest coding should be supported by the coding format. Examples of such coding formats are SVC and JPEG2000, as discussed in Section 2.3.1. Extraction of media tracks is dependent on the container format (e.g., MP4 or Ogg) and is usually not a problem since container formats do not introduce compression algorithms, i.e., they are mainly a wrapper around compressed media bitstreams.

D.3 Using Model-driven Content Adaptation

As discussed in Chapter 2, description-driven content adaptation is able to perform the removal of high-level bitstream structures and the modification of high-level syntax elements. Hence, this technique can be used to implement the media resource adaptation use case. More specifically, in this annex, we use model-driven content adaptation (introduced in Chapter 4).

In Figure D.1, the structural metadata of a multimedia resource is depicted. These structural metadata are compliant to the extended model for media bitstreams (discussed in Chapter 5). The multimedia resource is described by an *AnnotatedMultimedia* instance, which contains two *MediaBitstream* instances (i.e., a video and an audio track). The video and audio tracks are characterized

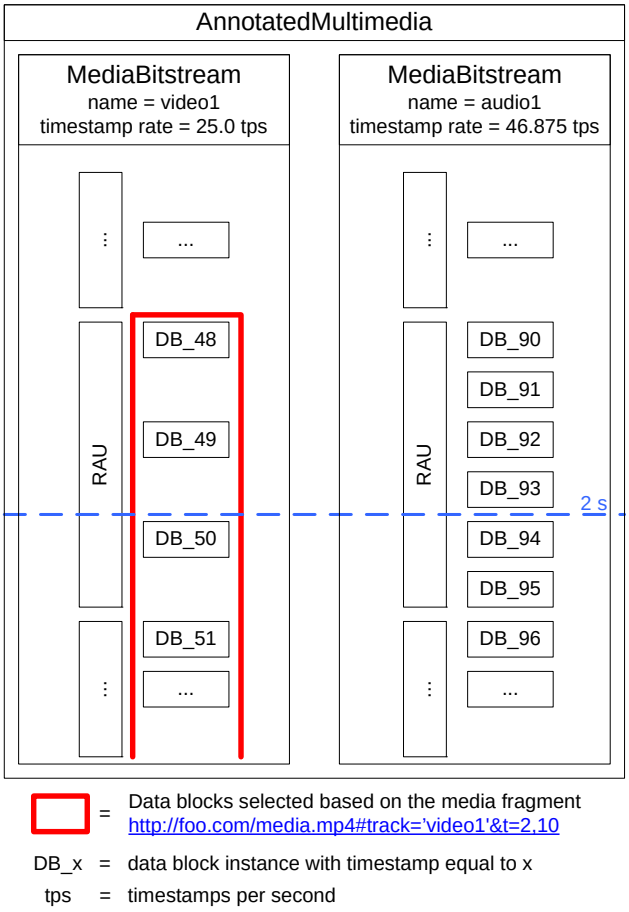


Figure D.1: Extracting temporal and track fragments using model-driven content adaptation.

by a timestamp rate of 25.0 and 46.875 timestamps per second, respectively. Since these two tracks are synchronized, their random access units are aligned, as discussed in Section 5.4.

Suppose we only want to select the video track from the media resource described in Figure D.1, and more specifically only a subpart of the video track starting from 2 s and ending at 10 s. A possible syntax[4] for such a media fragment URI could look as follows: `http://foo.com/media.mp4#track='video1'&t=2,10`. The extraction of this media fragment can be established using model-driven content adaptation, as illustrated in Figure D.1.

[4] At the time of writing this dissertation, the syntax was not specified yet by the MFWG.

First, the track having as name ‘video1’ is selected. In other words, only the *MediaBitstream* instance corresponding to the video track is considered for adaptation and delivery using the model-driven content adaptation tool chain (see Figure 4.14). Next, data blocks are selected based on the temporal parameters (i.e., from 2 s to 10 s). As shown in Figure D.1, DB_50 corresponds to 2 s in the video stream. However, to garantuee that the adapted video bitstream can be decoded in a correct way, cuts in the bitstream should only be performed at random access points (as discussed in Section 2.3.2). Hence, data blocks are selected starting from DB_48. Finally, the selected data blocks are packed in an MP4 container (as discussed in Section 5.3.1.2) and delivered to the client.

References

[1] I. Ahmad, X. Wei, Y. Sun, and Y.-Q. Zhang. Video Transcoding: an Overview of Various Techniques and Research Issues. *IEEE Transactions on Multimedia*, 7(5):793–804, 2005.

[2] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 1985.

[3] M. Amielh and S. Devillers. Multimedia Content Adaptation with XML. In *Proceedings of 8th International Conference on Multimedia Modeling*, pages 127–145, Amsterdam, The Netherlands, November 2001.

[4] M. Amielh and S. Devillers. Bitstream Syntax Description Language: Application of XML-Schema to Multimedia Content Adaptation. In *Proceedings of 11th International World Wide Web Conference*, Honolulu, Hawaii, May 2002. Available on `http://wwwconf.ecs.soton.ac.uk/archive/00000185/01/index.html`.

[5] M. C. Angelides and A. A. Sofokleous. Semantic content ranking through collaborative and content clustering. *Neurocomputing*, 71(13-15):2587–2595, August 2008.

[6] R. Arndt, R. Troncy, S. Staab, L. Hardman, and M. Vacura. COMM: Designing a Well-Founded Multimedia Ontology for the Web. In *6th International Semantic Web Conference (ISWC 2007)*, Busan, Korea, November 2007.

[7] O. Avaro, P. A. Chou, A. Eleftheriadis, C. Herpel, C. Reader, and J. Signès. The MPEG-4 Systems and Description Languages: a Way Ahead in Audio Visual Information Representation. *Signal Processing: Image Communication*, 9(4):385–431, 1997.

[8] P. Baccichet, T. Schierl, T. Wiegand, and B. Girod. Low-delay peer-to-peer streaming using scalable video coding. In *Packet Video 2007*, pages 173–181, Lausanne, Switzerland, November 2007.

[9] J. Baltazar, P. Pinho, and F. Pereira. Visual Attention Driven Image to Video Transmoding. In *Proceedings of 25th Picture Coding Symposium*, pages 6 on CD–ROM, Beijing, China, April 2006.

[10] C. Barton, P. Charles, D. Goyal, M. Raghavachari, M. Fontoura, and V. Josifovski. Streaming XPath processing with forward and backward axes. In *Proceedings of the 19th International Conference on Data Engineering*, pages 455–467, Bangalore, India, 2003.

[11] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 2396: "Uniform Resource Identifier (URI): Generic Syntax," Available on `http://www.ietf.org/rfc/rfc3986.txt`.

[12] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, pages 34–43, May 2001.

[13] P. V. Biron and A. Malhotra, editors. *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation. World Wide Web Consortium, October 2004. Available on `http://www.w3.org/TR/xmlschema-2/`.

[14] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau, editors. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C Recommendation. World Wide Web Consortium, August 2006. Available on `http://www.w3.org/TR/2006/REC-xml-20060816/`.

[15] D. Brickley and R. Guha, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. World Wide Web Consortium, February 2004. Available on `http://www.w3.org/TR/rdf-schema/`.

[16] D. Bulterman, G. Grassel, J. Jansen, A. Koivisto, N. Layaïda, T. Michel, S. Mullender, and D. Zucker, editors. *Synchronized Multimedia Integration Language (SMIL 2.1)*. W3C Recommendation. World Wide Web Consortium, December 2005. Available on `http://www.w3.org/TR/SMIL2/`.

[17] I. Burnett, F. Pereira, R. Van de Walle, and R. Koenen, editors. *The MPEG-21 book*. John Wiley & Sons, March 2006.

[18] S.-F. Chang and A. Vetro. Video Adaptation: Concepts, Technology, and Open Issues. *Proceedings of the IEEE*, 93(1):145–158, January 2005.

[19] C. Christopoulos, A. Skodras, and T. Ebrahimi. The JPEG2000 Still Image Coding System: an Overview. *IEEE Transactions on Consumer Electronics*, 46(4):1103–1127, November 2000.

[20] P. Cimprich. Streaming Transformations for XML, July 2004. Available on `http://stx.sourceforge.net/documents/spec-stx-20040701.html`.

[21] J. Clark, editor. *XSL Transformations (XSLT) Version 1.0*. W3C Recommendation. World Wide Web Consortium, November 1999. Available on `http://www.w3.org/TR/xslt`.

[22] J. Clark and S. DeRose, editors. *XML Path Language (XPath) 1.0*. W3C Recommendation. World Wide Web Consortium, 1999. Available on `http://www.w3.org/TR/xpath.html`.

[23] K. Clark, L. Feigenbaum, and E. Torres, editors. *SPARQL Protocol for RDF*. W3C Recommendation. World Wide Web Consortium, January 2008. Available on `http://www.w3.org/TR/rdf-sparql-protocol/`.

[24] D. Connolly, editor. *Gleaning Resource Descriptions from Dialects of Languages (GRDDL)*. W3C Recommendation. World Wide Web Consortium, September 2007. Available on `http://www.w3.org/TR/grddl/`.

[25] S. De Bruyne, D. De Schrijver, W. De Neve, D. Van Deursen, and R. Van de Walle. Enhanced Shot-Based Video Adaptation using MPEG-21 generic Bitstream Syntax Schema. In *Proceedings of the 1st IEEE Symposium on Computational Intelligence in Image and Signal Processing*, pages 380–385, Honolulu, United States, April 2007.

[26] S. De Bruyne, D. Van Deursen, J. De Cock, W. De Neve, P. Lambert, and R. Van de Walle. A compressed-domain approach for shot boundary detection on H.264/AVC bit streams. *Signal Processing: Image Communication – Special Issue on Semantic Analysis for Interactive Multimedia Services*, 23(7):473–498, August 2008.

[27] J. De Cock, S. Notebaert, P. Lambert, D. De Schrijver, and R. Van de Walle. Requantization Transcoding in Pixel and Frequency Domain for Intra 16x16 in H.264/AVC. *Lecture Notes in Computer Science – Advanced Concepts for Intelligent Vision Systems*, 4179:533–544, September 2006.

[28] W. De Neve, D. De Schrijver, D. Van Deursen, P. Lambert, and R. Van de Walle. Real-Time BSD-Driven Adaptation Along the Temporal Axis of H.264/AVC Bitstreams. In *Lecture Notes in Computer Science – Advances in Multimedia Information Processing - PCM 2006*, volume 4261, pages 131–140, Hangzhou, China, November 2006.

[29] W. De Neve, D. De Schrijver, D. Van Deursen, and R. Van de Walle. XML-Driven Bitstream Extraction Along the Temporal Axis of SMPTE's Video Codec 1. In *Proceedings of the 7th International Workshop on Image Analysis for Multimedia Interactive Services*, pages 233–236, Seoul, South Korea, April 2006.

[30] W. De Neve, D. Van Deursen, D. De Schrijver, K. De Wolf, and R. Van de Walle. Using Bitstream Structure Descriptions for the Exploitation of Multi-layered Temporal Scalability in H.264/AVC's Base Specification. *Lecture Notes in Computer Science: Advances in Mulitmedia Information Processing – PCM 2005*, 3768:641–652, November 2005.

[31] W. De Neve, D. Van Deursen, D. De Schrijver, S. Lerouge, K. De Wolf, and R. Van de Walle. BFlavor: A harmonized approach to media resource adaptation inspired by MPEG-21 BSDL and XFlavor. *Signal Processing: Image Communication*, 21(10):862–889, November 2006.

[32] W. De Neve, D. Van Deursen, W. Van Lancker, Y. M. Ro, and R. Van de Walle. Improved BSDL-based Content Adaptation for JPEG 2000 and HD Photo (JPEG XR), July 2009.

[33] W. De Neve, S. Yang, D. Van Deursen, C. Kim, Y. Ro, and R. Van de Walle. Analysis of BSDL-Based Content Adaptation for JPEG 2000 and HD Photo (JPEG XR). In *Proceedings of the 5th International Conference on Visual Information Engineering: Workshop on Scalable Coded Media Beyond Compression*, pages 717–722, Xi'an, China, July 2008.

[34] D. De Schrijver, W. De Neve, K. De Wolf, R. De Sutter, and R. Van de Walle. An Optimized MPEG-21 BSDL Framework for the Adaptation of Scalable Bitstreams. *Journal of Visual Communication and Image Representation*, 18(3):217–239, June 2007.

[35] D. De Schrijver, W. De Neve, K. De Wolf, P. Lambert, D. Van Deursen, and R. Van de Walle. XML-driven Exploitation of Combined Scalability in Scalable H.264/AVC Bitstreams. In *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems*, pages 1521–1524, New Orleans, United States, May 2007.

[36] D. De Schrijver, W. De Neve, K. De Wolf, and R. Van de Walle. Generating MPEG-21 BSDL Descriptions Using Context-Related Attributes. In *Proceedings of IEEE International Symposium on Multimedia*, pages 79–86, Irvine, USA, December 2005.

[37] D. De Schrijver, W. De Neve, K. De Wolf, D. Van Deursen, and R. Van de Walle. Exploitation of Combined Scalability in Scalable H.264/AVC Bitstreams by Using an MPEG-21 XML-Driven Framework. In *Lecture Notes in Computer Science – Advanced Concepts for Intelligent Vision Systems*, volume 4678, pages 699–710, August 2007.

[38] D. De Schrijver, W. De Neve, D. Van Deursen, S. De Bruyne, and R. Van de Walle. Exploitation of Interactive Region of Interest Scalability in Scalable Video Coding by Using an XML-driven Adaptation Framework. In *Proceedings of the 2nd International Conference on Automated Production of Cross Media Content for Multi-channel Distribution*, pages 223–231, Leeds, United Kingdom, December 2006.

[39] D. De Schrijver, W. De Neve, D. Van Deursen, J. De Cock, and R. Van de Walle. On an evaluation of transformation languages in a fully XML-driven framework for video content adaptation. In *Proceedings of the first international conference on innovative computing, information and control (ICICIC06)*, volume 3, pages 213–216, Beijing, China, September 2006.

[40] D. De Schrijver, W. De Neve, D. Van Deursen, Y. Dhondt, and R. Van de Walle. XML-based Exploitation of Region of Interest Scalability in Scalable Video Coding. In *Proceedings of the 8th International Workshop on Image Analysis for Multimedia Interactive Services*, pages 4 on CD–ROM, Santorini, Greece, June 2007.

[41] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. C. A. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The Semantic Web: The Roles of XML and RDF. *IEEE Internet Computing*, 4(5):63–74, 2000.

[42] S. Devillers, C. Timmerer, J. Heuer, and H. Hellwagner. Bitstream Syntax Description-Based Adaptation in Streaming and Constrained Environments. *IEEE Transactions on Multimedia*, 7(3):463–470, June 2005.

[43] Dublin Core Metadata Initiative. Dublin Core Metadata Element Set, version 1.1: Reference Description. Technical report, Dublin Core Metadata Initiative, 2004. Available on `http://www.dublincore.org/documents/dces/`.

[44] A. Eleftheriadis. Flavor: A Language for Media Representation. In *Proceedings of ACM Multimedia Conference*, pages 1–9, Seattle, WA, November 1997.

[45] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.

[46] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: "Hypertext Transfer Protocol – HTTP/1.1," Available on `http://www.w3.org/Protocols/rfc2616/rfc2616.html`.

[47] P. Gioia, K. Kamyab, I. Wolf, G. Panis, A. Difino, M. Kimiaei, T. DiGiacomo, A. Cotarmanac'h, P. Goulev, A. Graffunder, A. Hutter, B. Negro, C. Concolato, C. Joslin, E. Mamdani, J. Dufourd, and N. Thalmann. ISIS: Intelligent Scalability for Interoperable Services. In *Proceedings of 1st European Conference on Visual Media Production*, pages 295–304, London, United Kingdom, March 2004.

[48] GPAC Project on Advanced Content. Osmo player. Available on `http://gpac.sourceforge.net/`.

[49] M. M. Hannuksela, Y.-K. Wang, and M. Gabbouj. Isolated Regions in Video Coding. *IEEE Transactions on Multimedia*, 6:259–267, April 2004.

[50] M. Hartle, F.-D. Möller, S. Travar, B. Kröger, and M. Mühlhäuser. Using Bitstream Segment Graphs for Complete Data Format Instance Description. In *Proceedings of The Third International Conference on Software and Data Technologies (ICSOFT)*, pages 198–205, Porto, Portugal, August 2008.

[51] D. Hong and A. Eleftheriadis. XFlavor: providing XML features in media representation. *Multimedia Tools and Applications*, 39(1):101–116, 2008.

[52] M. Humphrey and J. Freeman. How xDSL Supports Broadband Services to the Home. *IEEE Network*, 11(1):14–23, January–February 1997.

[53] J. Hunter. Adding Multimedia to the Semantic Web - Building an MPEG-7 Ontology. In *First Semantic Web Working Symposium (SWWS), Proceedings*, pages 261–281, Stanford, USA, 2001.

[54] R. Iqbal, D. Ahmed, and S. Shirmohammadi. Distributed Video Adaptation and Streaming for Heterogeneous Devices. In *Proceedings of the 6th Annual IEEE International Conference on Pervasive Computing and Communications*, pages 492–497, Kowloon, Hong Kong, March 2008.

[55] R. Iqbal, S. Shirmohammadi, and A. El Saddik. A Framework for MPEG-21 DIA Based Adaptation and Perceptual Encryption of H.264 Video. In *Proceedings of SPIE/ACM Multimedia Computing and Networking Conference*, San Jose, USA, January 2007.

[56] ISO/IEC. Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio. ISO/IEC 11172-3:1993, 1993.

[57] ISO/IEC. Information technology – Syntactic metalanguage – Extended BNF. ISO/IEC 14977:1996, December 1996.

[58] ISO/IEC. Information technology – Generic coding of moving pictures and associated audio information: Video. ISO/IEC 13818-2:2000, 2000.

[59] ISO/IEC. Information technology – Coding of Audio, Picture, Multimedia and Hypermedia Information – Part 14: MP4 file format. ISO/IEC 14496-14:2003, December 2003.

[60] ISO/IEC. Information technology – Coding of audio-visual objects – Part 2: Visual. ISO/IEC 14496-2:2004, May 2004.

[61] ISO/IEC. Information technology – Multimedia framework (MPEG-21) – Part 7: Digital Item Adaptation. ISO/IEC 21000-7:2004, October 2004.

[62] ISO/IEC. Information technology – Coding of audio-visual objects – Part 3: Audio. ISO/IEC 14496-3:2005, 2005.

[63] ISO/IEC. Information technology – MPEG systems technologies – Part 1: Binary MPEG Format for XML. ISO/IEC 23001-1:2006, March 2006.

[64] ISO/IEC. Information technology – Multimedia framework (MPEG-21) – Part 18: Digital Item Streaming. ISO/IEC 21000-18:2007, June 2007.

[65] ISO/IEC. Information technology – Multimedia framework (MPEG-21) – Part 7: Digital Item Adaptation, Amendment 2: Dynamic and Distributed Adaptation. ISO/IEC 21000-7:2007/FPDAmd 2, January 2007.

[66] ISO/IEC. Resolutions of the 42nd ISO/IEC JTC1/SC29/WG1 Meeting, 2007-07-02/06. Technical Report ISO/IEC JTC1/SC29/WG1 N4293, Joint Photographic Experts Group (JPEG), Lausanne, Switzerland, July 2007.

[67] ISO/IEC. Information technology – MPEG systems technologies – Part 5: Bitstream Syntax Description Language. ISO/IEC 23001-5:2008, February 2008.

[68] ITU-T and ISO/IEC. Advanced Video Coding for Generic Audiovisual Services. ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, 2003.

[69] S. C. Johnson. Yacc: Yet Another Compiler-Compiler. Available on `http://dinosaur.compilertools.net/yacc/index.html`.

[70] Joint US/EU ad hoc Agent Markup Language Committee. DAML+OIL, March 2001. Available on `http://www.daml.org/2001/03/daml+oil-index.html`.

[71] M. Kay. *XSLT Programmers's Reference, 2nd Edition*. Wrox Press Ltd., Birmingham, UK, 2001.

[72] G. Klyne and J. J. Carroll, editors. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium, February 2004. Available on `http://www.w3.org/TR/rdf-concepts/`.

[73] R. Kuschnig, I. Kofler, M. Ransburg, and H. Hellwagner. Design options and comparison of in-network H.264/SVC adaptation. *Journal of Visual Communication and Image Representation*, 19(8):529–542, 2008.

[74] P. Lambert, D. De Schrijver, D. Van Deursen, W. De Neve, Y. Dhondt, and R. Van de Walle. A Real-Time Content Adaptation Framework for Exploiting ROI Scalability in H.264/AVC. *Lecture Notes in Computer Science – Advanced Concepts for Intelligent Vision Systems*, 4179:442–453, September 2006.

[75] A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne, editors. *Document Object Model (DOM) Level 3 Core Specification*. W3C Recommendation. World Wide Web Consortium, April 2004. Available on `http://www.w3.org/TR/DOM-Level-3-Core/`.

[76] S. Lerouge, P. Lambert, and R. Van de Walle. Multi-criteria Optimization for Scalable Bitstreams. *Lecture Notes in Computer Science, Visual Content Processing and Representation*, 2849:122–130, September 2003.

[77] J. Magalhães and F. Pereira. Using MPEG standards for multimedia customization. *Signal Processing: Image Communication*, 19(5):437–456, May 2004.

[78] B. Manjunath, P. Salembier, and T. Sikora, editors. *Introduction to MPEG-7: Multimedia Content Description Interface*. Wiley, New Jersey, 2003.

[79] E. Mannens, R. Troncy, K. Braeckman, D. Van Deursen, W. Van Lancker, R. De Sutter, and R. Van de Walle. Automatic Information Enrichment in News Production. In *Proceedings of the 10th International Workshop on Image Analysis for Multimedia Interactive Services*, pages 61–64, London, United Kingdom, May 2009.

[80] J. Martinez, R. Koenen, and F. Pereira. MPEG-7: The Generic Multimedia Content Description Standard, Part 1. *IEEE Multimedia*, 9(2):78–87, April-June 2002.

[81] J. Martinez, V. Valdes, J. Bescos, and L. Herranz. Introducing CAIN: a Metadata-driven Content Adaptation Manager Integrating Heterogeneous Content Adaptation Tools. In *Proceedings og International Workshop on Image Analysis for Multimedia Interactive Services*, pages 5 on CD–ROM, Montreux, Switzerland, April 2005.

[82] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, and L. Schneider. The WonderWeb Library of Foundational Ontologies (WFOL). Technical report, WonderWeb Deliverable 17, 2002.

[83] D. McGuinness and F. van Harmelen, editors. *OWL Web Ontology Language: Overview*. W3C Recommendation. World Wide Web Consortium, February 2004. Available on `http://www.w3.org/TR/owl-features/`.

[84] D.-E. Meddour, M. Mushtaq, and T. Ahmed. Open Issues in P2P Multimedia Streaming. In *IEEE ICC 2006 Workshop on Multimedia Communications Workshop (MultiCom)*, June 2006.

[85] Microsoft. Silverlight. Available on `http://silverlight.net/`.

[86] Microsoft. Smooth Streaming. Available on `http://www.iis.net/extensions/SmoothStreaming`.

[87] D. Mukherjee, E. Delfosse, J.-G. Kim, and Y. Wang. Optimal Adaptation Decision-taking for Terminal and Network Quality-of-service. *IEEE Transactions on Multimedia*, 7(3):454–462, June 2005.

[88] D. Mukherjee, G. Kuo, S. Hsiang, S. Liu, and A. Said. Format-independent scalable bit-stream adaptation using MPEG-21 DIA. In *Proceedings of the 2004 International Conference on Image Processing*, volume 4, pages 2793–2796, Singapore, October 2004.

[89] D. Mukherjee and A. Said. Structured Scalable Meta-formats (SSM) for Digital Item Adaptation. In *Internet Imaging IV*, volume 5018 of *Proceedings of SPIE*, January 2003.

[90] OECD. OECD Study on the Participative Web: User Generated Content, October 2007. Available on `http://www.oecd.org/dataoecd/57/14/38393115.pdf`.

[91] J.-R. Ohm. Advances in Scalable Video Coding. *Proceedings of the IEEE*, 93(1):42–56, January 2005.

[92] *P2PMMS'05: Proceedings of the ACM workshop on Advances in peer-to-peer multimedia streaming*, New York, NY, USA, 2005. ACM.

[93] S. B. Palmer. Pondering RDF Path, 2003. Available on `http://infomesh.net/2003/rdfpath`.

[94] G. Panis, A. Hutter, J. Heuer, H. Hellwagner, H. Kosch, C. Timmerer, S. Devillers, and M. Amielh. Bitstream Syntax Description: a Tool for Multimedia Resource Adaptation within MPEG-21. *Signal Processing: Image Communication*, 18(8):721–747, September 2003.

[95] F. Peng and S. S. Chawathe. XPath Queries on Streaming Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Diego, California, June 2003.

[96] S. Pfeiffer, C. Parker, and C. Schremmer. Annodex: A Simple Architecture to Enable Hyperlinking, Search & Retrieval of Time Continuous Data on the Web. In *MIR '03: Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pages 87–93, Berkeley, California, November 2003.

[97] M. Prangl, H. Hellwagner, and T. Szkaliczki. Fast Adaptation Decision Taking for Cross-Modal Multimedia Content Adaptation. In *Proceedings of the 2006 International Conference on Multimedia and Expo*, pages 137–140, Toronto, Canada, July 2006.

[98] M. Prangl, T. Szkaliczki, and H. Hellwagner. A Framework for Utility-based Multimedia Adaptation. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(6):719–728, June 2007.

[99] E. Prud'hommeaux and A. Seaborne, editors. *SPARQL Query Language for RDF*. W3C Recommendation. World Wide Web Consortium, November 2007. Available on `http://www.w3.org/TR/rdf-sparql-query/`.

[100] Quicktime. Available on `http://www.apple.com/quicktime/`.

[101] M. Ransburg, R. Cazoulat, B. Pellan, C. Concolato, S. De Zutter, C. Poppe, A. Hutter, H. Hellwagner, and R. Van de Walle. Dynamic and Distributed Adaptation of Scalable Multimedia Content in a Context-Aware Environment. In *Proceedings of European Symposium on Mobile Media Delivery*, Alghero, Italy, September 2006.

[102] M. Ransburg, S. Devillers, C. Timmerer, and H. Hellwagner. Processing and Delivery of Multimedia Metadata for Multimedia Content Streaming. In *Proceedings of 6th Workshop on Multimedia Semantics - The Role of Metadata*, Aachen, Germany, March 2007.

[103] M. Ransburg and H. Hellwagner. Generic Streaming of Multimedia Content. In *Proceedings of IASTED International Conference on Internet and Multimedia Systems and Applications*, pages 324–330, Grindelwald, Switzerland, February 2005.

[104] RealNetworks. SureStream. Available on `http://service.realnetworks.com/help/library/guides/producerplus85/htmfiles/preparin.htm`.

[105] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC 3550: "RTP: A Transport Protocol for Real-Time Applications," Available on `http://www.ietf.org/rfc/rfc3550.txt`.

[106] H. Schulzrinne, A. Rao, and R. Lanphier. RFC 2326: "Real Time Streaming Protocol," Available on `http://www.ietf.org/rfc/rfc2326.txt`.

[107] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, September 2007.

[108] A. Seaborne and G. Manjunath. SPARQL/Update: a language for updating RDF graphs, January 2008. Available on `http://jena.hpl.hp.com/~afs/SPARQL-Update.html`.

[109] SEAmless Content Delivery. Available on `http://www.ist-sea.eu/`.

[110] J. R. Smith and P. Schirling. Metadata Standards Roundup. *IEEE Multimedia*, 13(2):84–88, 2006.

[111] SMPTE. Material Exchange Format (MXF) – File Format Specification (Standard). SMPTE 377M-2004, 2004.

[112] SMPTE. Standard for Television: VC-1 Compressed Video Bitstream Format and Decoding Process. SMPTE 421M-2006, 2006.

[113] SMPTE. Proposed SMPTE Standard for Television: VC-1 Compressed Video Bitstream Format and Decoding Process. document 421M, SMPTE, New York, USA, August 2005.

[114] A. Souzis. RxPath: a mapping of RDF to the XPath Data Model. In *Extreme Markup Language 2006*, Montreal, Canada, August 2006.

[115] S. Srinivasan, C. Tu, S. L. Regunathan, and G. J. Sullivan. HD Photo: a New Image Coding Technology for Digital Photography. In *Proceedings of the SPIE*, volume 6696, San Diego, US-CA, USA, August 2007.

[116] X. Sun, C.-S. Kim, and C.-C. Jay Kuo. MPEG video markup language and its applications to robust video transmission. *Journal of Visual Communication and Image Representation*, 16(4-5):589–620, August-October 2005.

[117] J. Thomas-Kerr, I. Burnett, and C. Ritz. Format-Independent Multimedia Streaming. In *Proceedings of 2006 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1509–1512, July 2006.

[118] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, editors. *XML Schema Part 1: Structures Second Edition*. W3C Recommendation. World Wide Web Consortium, October 2004. Available on `http://www.w3.org/TR/xmlschema-1/`.

[119] Tim Berners-Lee. Notation 3: An readable language for data on the Web. Available on `http://www.w3.org/DesignIssues/Notation3.html`.

[120] C. Timmerer, T. Frank, and H. Hellwagner. Efficient processing of MPEG-21 metadata in the binary domain. In *Proceedings of SPIE International Symposium ITCom 2005 on Multimedia Systems and Applications VIII*, Boston, Massachusetts, USA, October 2005.

[121] C. Timmerer, G. Panis, H. Kosch, J. Heuer, H. Hellwagner, and A. Hutter. Coding Format Independent Multimedia Content Adaptation using XML. In *Proceedings of SPIE International Symposium ITCom 2003 on Internet Multimedia Management Systems IV*, volume 5242, pages 92–103, Orlando, September 2003.

[122] R. Troncy, W. Bailer, M. Hausenblas, P. Hofmair, and R. Schlatte. Enabling Multimedia Metadata Interoperability by Defining Formal Semantics of MPEG-7 Profiles. In *1st International Conference on Semantics And digital Media Technology (SAMT 2006)*, pages 41–55, Athens, Greece, December 2006.

[123] R. Troncy, J. Jansen, Y. Lafon, E. Mannens, S. Pfeiffer, and D. Van Deursen, editors. *Use cases and requirements for Media Fragments*. W3C Working Draft. World Wide Web Consortium, April 2009. Available on `http://www.w3.org/TR/media-frags-reqs/`.

[124] D. Van Deursen. Ontwikkeling van een referentiemodel voor bitstroomstructuurbeschrijvingstalen, toegepast op MPEG-21 BSDL en (X)FLAVOR. Master's thesis, Ghent University, June 2005.

[125] D. Van Deursen, S. De Bruyne, W. Van Lancker, W. De Neve, D. De Schrijver, H. Hellwagner, and R. Van de Walle. MuMiVA: a Multimedia Delivery Platform using Format-agnostic, XML-driven Content Adaptation. In *Proceedings of the 9th International Symposium on Multimedia*, pages 131–138, Taichung, Taiwan, December 2007.

[126] D. Van Deursen, W. De Neve, D. De Schrijver, and R. Van de Walle. BFlavor: an Optimized XML-based Framework for Multimedia Content Customization. In *Proceedings of the 25th Picture Coding Symposium*, pages 6 on CD–ROM, Beijing, China, April 2006.

[127] D. Van Deursen, W. De Neve, D. De Schrijver, and R. Van de Walle. Automatic generation of generic Bitstream Syntax Descriptions applied to H.264/AVC SVC encoded video streams. In *Proceedings of the 14th International Conference on Image Analysis and Processing*, pages 382–387, Modena, Italy, September 2007.

[128] D. Van Deursen, W. De Neve, D. De Schrijver, and R. Van de Walle. gBFlavor: a New Tool for Fast and Automatic Generation of generic Bitstream Syntax Descriptions. *Multimedia Tools and Applications*, 40(3):453–494, December 2008.

[129] D. Van Deursen, W. De Neve, W. Van Lancker, and R. Van de Walle. Semantic Adaptation of Synchronized Multimedia Streams in a Format-independent Way. In *Proceedings of the 27th Picture Coding Symposium*, pages 4 on CD–ROM, Chicago, United States, May 2009.

[130] D. Van Deursen, D. De Schrijver, S. De Bruyne, and R. Van de Walle. Fully Format Agnostic Media Resource Adaptation Using an Abstract Model for Scalable Bitstreams. In *Proceedings of the 2007 IEEE International Conference on Multimedia and Expo*, pages 240–243, Beijing, China, July 2007.

[131] D. Van Deursen, D. De Schrijver, W. De Neve, and R. Van de Walle. A Real-Time XML-Based Adaptation System for Scalable Video Formats. In *Lecture Notes in Computer Science – Advances in Multimedia Information Processing - PCM 2006*, volume 4261, pages 339–348, Hangzhou, China, November 2006.

[132] D. Van Deursen, C. Poppe, G. Martens, E. Mannens, and R. Van de Walle. XML to RDF Conversion: a Generic Approach. In *Proceedings of the 4th International Conference on Automating Production of Cross Media Content for Multi-channel Distribution*, pages 138–143, Florence, Italy, November 2008.

[133] D. Van Deursen, W. Van Lancker, S. De Bruyne, W. De Neve, E. Mannens, and R. Van de Walle. Format-independent and Metadata-driven Media Resource Adaptation using Semantic Web Technologies. Submitted to *Multimedia Systems*.

[134] D. Van Deursen, W. Van Lancker, W. De Neve, T. Paridaens, E. Mannens, and R. Van de Walle. NinSuna: a Fully Integrated Platform for Format-independent Multimedia Content Adaptation and Delivery based on Semantic Web Technologies. Submitted to *Multimedia Tools and Applications – Special Issue on Data Semantics for Multimedia Systems*.

[135] D. Van Deursen, W. Van Lancker, T. Paridaens, W. De Neve, E. Mannens, and R. Van de Walle. NinSuna: a Format-independent Multimedia Content Adaptation Platform based on Semantic Web Technologies. In *Proceedings of the 10th International Symposium on Multimedia*, pages 491–492, Berkeley, United States, December 2008.

[136] A. Vetro, C. Christopoulos, and T. Ebrahimi. Universal Multimedia Access. *IEEE Signal Processing Magazine*, 20(2):16, March 2003.

[137] A. Vetro, C. Christopoulos, and H. Sun. Video Transcoding Architectures and Techniques: an Overview. *IEEE Signal Processing Magazine*, 20(2):18–29, 2003.

[138] VideoLan. VLC media player. Available on `http://www.videolan.org/`.

[139] W3C Multimedia Semantics Incubator Group. Available on `http://www.w3.org/2005/Incubator/mmsem/`.

[140] B. Walke, P. Seidenberg, and M. Althoff. *UMTS: The Fundamentals*. John Wiley & Sons, April 2003.

[141] N. Walsh. RDF Twig: accessing RDF graphs in XSLT. In *Extreme Markup Language 2003*, Montreal, Canada, August 2003.

[142] R. Whitmer, editor. *Document Object Model (DOM) Level 3 XPath Specification*. W3C Working Group Note. World Wide Web Consortium, February 2004. Available on `http://www.w3.org/TR/DOM-Level-3-XPath/`.

[143] T. Wiegand, G. Sullivan, J. Reichel, H. Schwarz, and M. Wien. Joint Draft 8 of SVC Amendment, October 2006. JVT-document JVT-U201, Hangzhou, China. Available on `http://ftp3.itu.ch/avarch/jvt-site/2006_10_Hangzhou/JVT-U201.zip`.

[144] Windows Media Series. Intelligent Streaming. Available on `http://www.microsoft.com/windows/windowsmedia/howto/articles/intstreaming.aspx`.

[145] WinZip. Available on `http://www.winzip.com/`.

[146] J. Xin, C.-W. Lin, and M.-T. Sun. Digital Video Transcoding. *Proceedings of the IEEE*, 93(1):84–97, 2005.

[147] J. Xin, A. Vetro, H. Sun, and Y. Su. Efficient MPEG-2 to H.264/AVC transcoding of intra-coded video. *EURASIP Journal of Applications and Signal Processing*, 2007(1):217–229, January 2007.

[148] M. Xu, J. Li, Y. Hu, L. Chia, B. Lee, D. Rajan, and J. Cai. An Event-Driven Sports Video Adaptation for the MPEG-21 DIA Framework. In *Proceedings of IEEE International Conference on Multimedia and Expo*, pages 1245–1248, Toronto, Canada, July 2006.

[149] XWRT (XML-WRT). Word Replacing Transform for eXtended Markup Language, Available on `http://xwrt.sourceforge.net/`.