INTEGRATING MPEG-4 MEDIA IN THE CONTEXT OF HTML5 TECHNOLOGIES

by

DIONISIOS KLADIS

B.A., Technological Educational Institute of Crete, 2010

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF APPLIED INFORMATICS
AND MULTIMEDIA

SCHOOL OF APPLIED TECHNOLOGY

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

2015

Approved by:

Major Professor
Athanasios G. Malamos

Abstract

The MPEG-4 standard provides various technologies that deal with delivering different types of multimedia content. This standard defines structures that deal with images, videos, sound, 2D and 3D content, digital rights management data and content descriptors.

The main format that MPEG-4 uses to describe spatial and temporal 3D scene structure and interactions is the Binary Format for Scenes (BIFS). BIFS is a compressed, binary version of the VRML language. But this format doesn't offer much flexibility to authors, so MPEG-4 supports another, textual, format. The Extensible MPEG-4 Textual (XMT) format is used to represent MPEG-4 scenes using a textual XML syntax, and is directly derived from BIFS. Although MPEG-4 is a standard for delivering multimedia content, there is at this point, no specific framework that can offer MPEG-4 content support on web pages, natively, without the need of extra software or plug-ins.

The subject of this thesis is the creation of a framework that will enable authors and developers to display 2D and 3D MPEG-4 content using only HTML5 and JavaScript. This will bring the MPEG-4 standard up to date with modern multimedia trends. Additionally, we explore ways through which to integrate XMT-A interactive scenes with modern video containers, so as to integrate video and 3D scene media transmission and display in the context of HTML5.

To do this, we fuse several different technologies, including X3D, X3DOM, XSLT and the WebM video format. Using this framework, a developer can create an HTML5 web page that can accommodate live interactive X3D scenes that conform to the MPEG-4 standard, can be integrated with WebM video, and are easy to author, read and manipulate. The main contributions of this work are: an exploration of the relationship between the XMT-A standard and the X3D language, an automatic conversion framework between the two, a mechanism for reproducing XMT-A scenes in HTML5 pages using X3DOM, and a transfer and display mechanism of XMT-A scenes through and alongside WebM video files. Ultimate goal is to create an In-Stream advertising platform that delivers interactive 3D advertisements to end-users.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Dr. Athanasios G. Malamos for supporting me and giving me this opportunity, as well as Dr. Markos Zampoglou for his guidelines and valuable help.

Dedication

Dedicated to my family and friends who are always there for me.

Chapter 1 - Introduction

We are living in a time that the World Wide Web is becoming more and more a significant and essential part of people's lives. Almost every aspect of our daily routine involves some form of communication or interactivity through the Web. From sending a simple message to a colleague or friend to controlling airborne delivery drones (another major trend in the last years); the Web is what makes all these technologies possible. It offers lightning fast communication across the globe, powers companies (from small ones to the largest of the world), gives us the opportunity to entertain ourselves by reading books, playing games, hearing our favorite music, watching videos of (TV) shows or movies.

The Web is quite suitable for all kinds of multimedia content. Every web page is full of text, graphics and photos, sounds and music are easily played back and videos can be streamed to a wide range of different devices. There is, though, as major problem; there exists a multitude of different standards for describing and creating multimedia content (whether we are talking about images or video or any other type) and a large number of end-user devices that have different capabilities and features. This leads to the need of standardizing multimedia content in a way that it can be easily authored, delivered, adapted and viewed by as many users as possible. MPEG-4 is one of the many multimedia standards, which deals with the creation of complex interactive multimedia scenes (even though it is mostly recognized for its video codec) that contain sounds, images, videos, 3D objects etc.

In many cases in the past, for a specific standard to function on the Web and operate within a Web browser context, there was the need for special plugins. Those plugins, installed at the user's browser, allowed the display and playback of the multimedia content they were designed for (for example, X3D content need a special plugin). As technology evolves and with the advent of HTML5, the Web gains lots of features that enable developers to use and display various multimedia content "right out of the box", without the need of extra software or plugins. There is a lot of prospect and potential to this, as many different multimedia types can "interact" with each other in order to provide complex and spectacular experience to the end user, all "integrated" in some way under the umbrella of HTML5.

The subject of this thesis is the creation of a framework that will enable authors and developers to display 2D and 3D MPEG-4 content using only HTML5 and JavaScript. This will

bring the MPEG-4 standard up to date with modern multimedia trends. Additionally, we explore ways through which to integrate XMT-A interactive scenes with modern video containers, so as to integrate video and 3D scene media transmission and display in the context of HTML5.

To achieve this, we fuse several different technologies, including X3D, X3DOM, XSLT and the WebM video format. Using this framework, a developer can create an HTML5 web page that can accommodate live interactive 3D scenes that conform to the MPEG-4 standard, can be integrated with WebM video and are easy to author, read and manipulate. The main contributions of this work are: an exploration of the relationship between the XMT-A standard and the X3D language, an automatic conversion framework between the two, a mechanism for reproducing XMT-A scenes in HTML5 pages using X3DOM, and a transfer and display mechanism of XMT-A scenes through and alongside WebM video files.

1.1 "The Champion Application"

As we mentioned before, we merged all these different technologies in order to create a simple to operate yet quite enhanced video player. But, what is the ultimate goal of such technology? We tried to address the needs of a very specific sector of advertising, the so called "In-Stream Advertising". In-Stream ads are the ones that appear before, during or after a video stream, inside the video player (for more details see Chapter 2.1). The video player that we propose can be used in systems or services that incorporate and deliver video playback alongside with ads, such as YouTube or Vimeo. With our framework, the aforementioned services can display fully interactive advertisements that engage the viewer more than the traditional promotional means. The viewer has the ability to interact with the advertisement, examine an object (for example, rotate a model of a car while watching a video about this specific car) from various points of view and more. This kind of ads can be more engaging and potentially produce more profit.

Such an application must meet some specific standards in order to operate seamlessly with the video that the user watches and engage him/her (which results in increased profits). First of all, the term "In-Stream" refers to the ability of the advertisement to be displayed while the video is played back, usually in such a place that renders it visible on the video area. This is done because the user is already engaged with viewing the multimedia content, which means that he/she is much less aware of advertisements that appear in places around the video.

Another aspect in displaying In-Stream advertisements is how it is made visible. One way is that the advertisement takes the whole video area (basically pausing the video playback) in order to promote something and after its end, the user continues watching the requested content. This is the most intrusive way of ads as it prohibits the user from enjoying the video he/she wanted. The other way for displaying an In-Stream advertisement is to "merge" it with the video content. The ad is not a solid image or video, but rather a transparent image or 3D content that is displayed on top of the normal video playback, thus being much less intrusive for the end-user.

In-Stream advertisement content can vary in multimedia type regarding the interaction level. It can feature static, non-interactive content such as images and videos or can be an interactive 3D application that draws the user's attention more efficiently and encourages the interaction between the two, resulting in an engaging and/or immersive experience. Another type of interactivity is whether the advertisement is active or not. When the user clicks the active advertisement (or some part of it), it opens a new web browser window and navigates the user to the webpage of the promoted product or service.

Important role in an effective In-Stream advertisement has the time. It is crucial that the user gets the advertised message in the right moment, without being completely distracted from watching the video. So, an In-Stream advertisement system has to handle every possible scenario regarding the timing; Ads can be displayed in the beginning, just before the video starts, at the end, after the video playback is complete and finally during the playback, at a time slot that is certain that the user is able to receive an advertisement effectively.

A factor that must be taken into account when designing an In-Stream advertisements system is whether the end-user will be able to skip the advertisement or not, something that immediately affects the user's satisfaction. For example, YouTube allows the user to skip the advertisement after a certain amount of time. Another point that affects user satisfaction and engagement implicitly is how the system works; by using an extra piece of plugin (that must be downloaded and installed separately) or by using already established and supported by the web browser technologies that allow seamless and hassle-free operation? Obviously, the latter is the target of a successful In-Stream advertising platform.

With all the previous information in mind it is clear that our proposed system must be compatible with the web standards by using all the necessary features that HTML5 has to offer, as HTML5 is the current state-of-the-art technology that is supported by all the major web

browsers. Specifically, in order to display complex content inside a web page, HTML5 offers the canvas element, that allows drawing various types of content with ease. This is crucial to our application, as the main feature is to display 3D interactive scenes over a streaming video. But, in order to display those interactive 3D scenes, we need a solid framework, which conforms to the web standards, and allows real-time 3D rendering.

This is achieved by using X3DOM, a JavaScript framework that enables 3D rendering on a canvas element with ease, without the need of a specialized plugin or external software. X3DOM reads 3D worlds that are created using the X3D format and supports full interactivity between the various 3D objects and the user's input devices. X3D is a format for describing, creating and distributing 3D content and features a quite easy set of commands for doing all these. Additionally, it supports scripting for adding extended functionality and interactivity.

Because we wanted to support some basic form of timing in our 3D scenes (manipulating various scene aspects based on time) we chose to use the technologies and features that the MPEG-4 standard has to offer. MPEG-4 is one of the leading frameworks (if not the leading) in content description and distribution (although it is commonly known for its video codec). MPEG-4 offers the XMT format; XMT is used to describe, create and distribute 3D content just like the X3D. Moreover, it is based on X3D (has the majority of its features) and supports timing events by using XMT-Ω, a SMIL-like format. The close connection with X3D was the major factor that led us to choose MPEG-4 XMT as the format that the 3D content is created in.

Finally, we needed a carrier for this In-Stream content that we create, so we decided to use a multimedia container format that is focused on web streaming, is supported by the majority of web browsers (without the need of plugins or externally installed codecs) and is easily expandable. WebM is the right video format for the job! It is based on the Matroska multimedia container format, which gives authors great flexibility and the ability to store external files (such as codecs) inside the video file. This particular feature is very useful to our proposed system, as it allows to embed the interactive In-Stream 3D content inside the video file and, when the user request this particular file, transfer and process it with ease.

1.2 Technological overview

In our effort to promote interactivity for the case of In-Stream advertisement we engage several emerging as well as mature technologies. The MPEG-4 standard provides various technologies that deal with delivering different types of multimedia content. This standard defines structures that deal with images, videos, sound, 2D and 3D content, digital rights management data and content descriptors. It can be used in digital TV (either online web TV or broadcasted), mobile multimedia content delivery (such as games, entertainment applications etc.), video conferencing and many more. Although it is widely known mostly for its parts that deal with audio and video compression, the MPEG-4 standard contains an entire framework for media structure, streaming, metadata, rights management and many more. In our proposed application, MPEG-4 is used to describe the audiovisual and 3D content that is displayed as an In-Stream advertisement over the normal video playback. It is our choice as it is the dominant standard for creating, describing and streaming rich and complex audiovisual interactive content over the Web and is supported and used by the majority of devices that can display multimedia content (such as mobile devices, TVs etc.)

MPEG-4 is much more than a simple audiovisual "carrier" with just two multimedia streams in it (one for the audio, one for the video). It is a very complex package, a scene if you prefer, that incorporates a large number of multimedia objects of various types (such as rectangular video, video with shape, audio, 3D objects, speech, text, graphics and many more).

But so many objects need to be somehow placed into some order (where to place each object, if they move, if they take user input, do they make sounds, sound effects etc.) so that the viewer/user gets a visually and aurally catching experience. In other words, there is the need for object composition into the scene. MPEG-4 has a system for describing scenes, called BIFS (BInary Format for Scenes). (Read more about BIFS on chapter 2.3)

BIFS is a compressed, binary version of the VRML language. But this format doesn't offer much flexibility to authors, so MPEG-4 supports another, textual, format. The Extensible MPEG-4 Textual (XMT) format is used to represent MPEG-4 scenes using a textual XML syntax, and is directly derived from BIFS. Our proposed system uses the XMT format in order to create, store and embed in the final video file the interactive 3D content that will be displayed overlaid on the video stream. We chose XMT because it is the proposed format for describing 3D content in the MPEG-4 standard, it is relatively easy to manage and is convertable to the 3D

content standards that are used to display 3D content on the Web. Both XMT and BIFS are described in MPEG-4, Part 11: Scene Description [8][13]. (For more information about the XMT format, read chapter 2.4)

Extensible Binary Meta-Language (EBML for short) is an open standard that was specifically designed to be used in the Matroska project. EBML is designed as a very simple binary version of XML, so its syntax and semantic are separated. Each application, that features an EBML library, can read data based on this standard; the interpretation of the data is done by the application.

WebM is a relatively new multimedia container format that is designed specifically for fulfilling the needs of the web for a totally free and open source multimedia file format. Like all the multimedia container formats (as the Matroska), WebM defines the structure that a WebM file should have (based on the Matroska multimedia container EBML structure) and what specific codecs must be used for encoding video and audio streams [4][5][7]. Because of the close relation of WebM with the Matroska format, one can attach whole files inside of the video file. Our video player is designed to playback special WebM video files that contain an XMT file, which describes the 3D interactive content that will be displayed on top of the normal video playback for In-Stream advertising purposes. (You can find more information about EBML and WebM on chapter 2.5).

X3D is an architecture that provides the entire infrastructure needed in order to create, store, restore and reproduce in real time 3-dimensional scenes and objects using the XML syntax. It is based and improves VRML (the first 3D interchange standard that enabled 3-dimensional content to be displayed over the web) and besides rendering 3D scenes and animations, it offers the ability to use scripts (using ECMAScript or Java), node prototyping, adds more data encodings etc. [11] [24] X3D is officially incorporated within the MPEG-4 multimedia standard as stated in [24].

X3DOM is a framework that gives the ability to web developers to embed X3D elements and scenes into the HTML5 DOM tree. Using this framework, a developer can create an HTML5 web page that can accommodate live interactive X3D scenes that can be manipulated (add, remove or change elements) easily. Technically, X3DOM fills the gap between the X3D content and the HTML5 DOM and connects them. Our proposed video player extracts the aforementioned XMT file from inside of the WebM video and converts it to the X3D format, so

that the interactive 3D content can be handled by the X3DOM framework. Then, X3DOM takes over and displays the interactive In-Stream advertisement. (Read more about the X3D architecture and the X3DOM framework on chapter 2.6)

1.3 Thesis structure and organization

In the following chapters, we present a general theoretic background about the technologies and standards that we put in use for the creation of the framework, as well as analyzing all the steps involved from creating an XMT file to the final step of displaying 3D scenes inside a video playback. In more detail:

- **Chapter 1: Introduction**. In the first chapter we introduce the reader to In-Stream Advertising and give a general description of the fundamental technologies that bring our framework into life, such as the MPEG-4 standard, the XMT scene description, the WebM multimedia container and its structure and the X3D and X3DOM for displaying 3D content inside HTML5 web pages.
- **Chapter 2: Technology**. In this chapter we present in detail all the relevant technologies that we used in order to bring to life our project. Thus, we explain what In-Stream advertising is, give more specific details about the MPEG-4 standard and present in detail the XMT format. Additionally, we present the WebM multimedia format and how it is used in this thesis and introduce in detail the X3D standard, along with the X3DOM framework.
- **Chapter 3: HTML5 video player with In-Stream 3D Interactive Scenes display capabilities**. This is the most important chapter of this thesis. Here, we explain and demonstrate how the proposed framework works in detail. We present the differences between the XMT-A scene description and the X3D standard and proceed to a detailed presentation of each step involved.
- **Chapter 4: Conclusions, applications and future steps**. As the title suggests, in this chapter we summarize all the topics covered in this thesis, present some of the potential uses of the framework and suggest improvements on it.

Chapter 2 - Technology

2.1 In-Stream Advertising

As we mentioned in the introductory chapter, we developed the proposed framework with a specific use case in mind; In-Stream Advertising (ISA in short). In-Stream ads are the ones that appear before, during or after a video stream, inside the video player (just like the "regular" ads displayed on the standard TV program) [39][40].

ISA are divided into three categories:

- **Pre-roll**: These are ads that are displayed before the requested video content is played back. This kind of In-Stream ads is quite intrusive, because the viewer has to wait for a certain amount of time before the playback begins.
- **Mid-roll**: This is the kind of video ads that are displayed during the playback of the requested video content. They are considered to be the most intrusive advertisements, as they either stop the video playback (until their playback is complete) or they are overlaid on top of the normal video content. Yet, they are the most effective from the ISA group.
- **Post-roll**: These are ads that are displayed after the end of the requested video content and are considered to be the least intrusive for the viewer.

In-Stream ads are more effective in promoting a business or a product than the "traditional" banner advertisements, because they engage more easily the end-user (who is accustomed into "removing" banners from his eyes). Additionally, ISA can provide more earnings for the companies that provide the advertising content than the other means of advertising which favor the companies that display them. [40]

In-Stream advertising is one category of Interactive Advertisements. Interactive advertising is used to promote or sell products, services, to communicate various messages to the viewers etc. It uses online or offline multimedia content in such a manner that it gives the ability to the end viewers to interact with the displayed advertisement. As mentioned above, interactive advertisements can be much more effective and engaging as they immerse the user's actions into

them and grabbing the viewer's attention, which in turn could lead to increased revenue for the advertiser.

2.2 MPEG-4

The MPEG-4 standard provides various technologies that deal with delivering different types of multimedia content. This standard defines structures that deal with images, videos, sound, 2D and 3D content, digital rights management data and content descriptors. It can be used in digital TV (either online web TV or broadcasted), mobile multimedia content delivery (such as games, entertainment applications etc.), video conferencing and many more. Although it is widely known mostly for its parts that deal with audio and video compression, the MPEG-4 standard contains an entire framework for media structure, streaming, metadata, rights management and many more [13] [17] [19] [20] [21] [22].

MPEG-4 is divided in several parts, the most relevant (for this thesis) are:

- **Part 11**
  Scene description and application engine (XMT and BIFS are described here).
- **Part 16**
  Animation Framework eXtension (AFX).
- **Part 20**
  Lightweight Application Scene Representation and Simple Aggregation Format.

MPEG-4 provides standardized mechanisms that deal with:

- The representation of units of aural, visual or audiovisual content ("media objects").
- The composition of the media objects in order to create audiovisual scenes.
- Multiplexing and synchronizing all the data that are related with the media objects.
- The interaction with the audiovisual scene produced at the receiver's side.

Content representation

Each scene in MPEG-4 consists of numerous objects (called media objects) such as still images, video sequences, audio tracks etc. MPEG-4 standardizes such media objects that are able to represent natural of synthetic content types and can be 2D or 3D. Media objects, represented in MPEG-4 form, are composed of specific elements that describe how to deal with the media objects inside the scene.

Examples of media objects types follow:

- Still images
- Video sequences
- Audio tracks (natural or synthetic)
- Text
- 3D models

Composition of media objects

As mentioned above, every MPEG-4 scene contains simple media objects like images, video, audio etc. These objects are called primitive object, as they are used to form more complex media objects (compound media objects). For example, a visual object depicting a talking person is bound with the reciprocal audio object (the one with the person's voice) in order to construct a new, compound, media object of a talking person. Another example is a 3 dimensional scene that depicts a rotating cube, which displays a video object on its sides.

MPEG-4 standard defines how to describe and create a scene that, among others, allows to:

- Distribute media objects (primitive or compound) over any given coordinate system.
- Apply transformations in order to reform the geometry or acoustics of a media object.
- Group media objects to create more complex ones.
- Interact with the scene from the user's end in order to change the viewing and listening point.

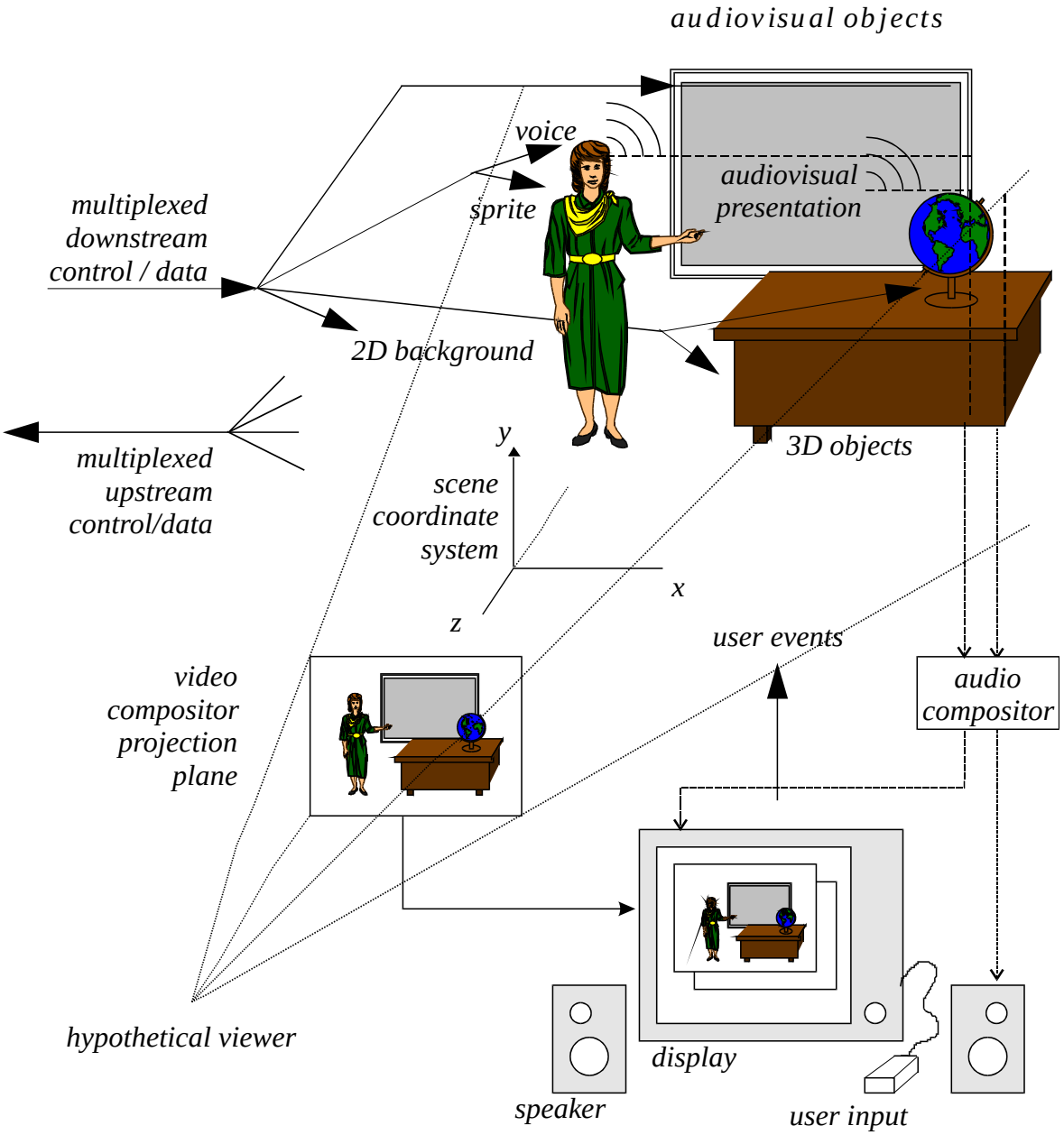

Figure 2-1 Example of an MPEG-4 scene, composed of various objects [13]

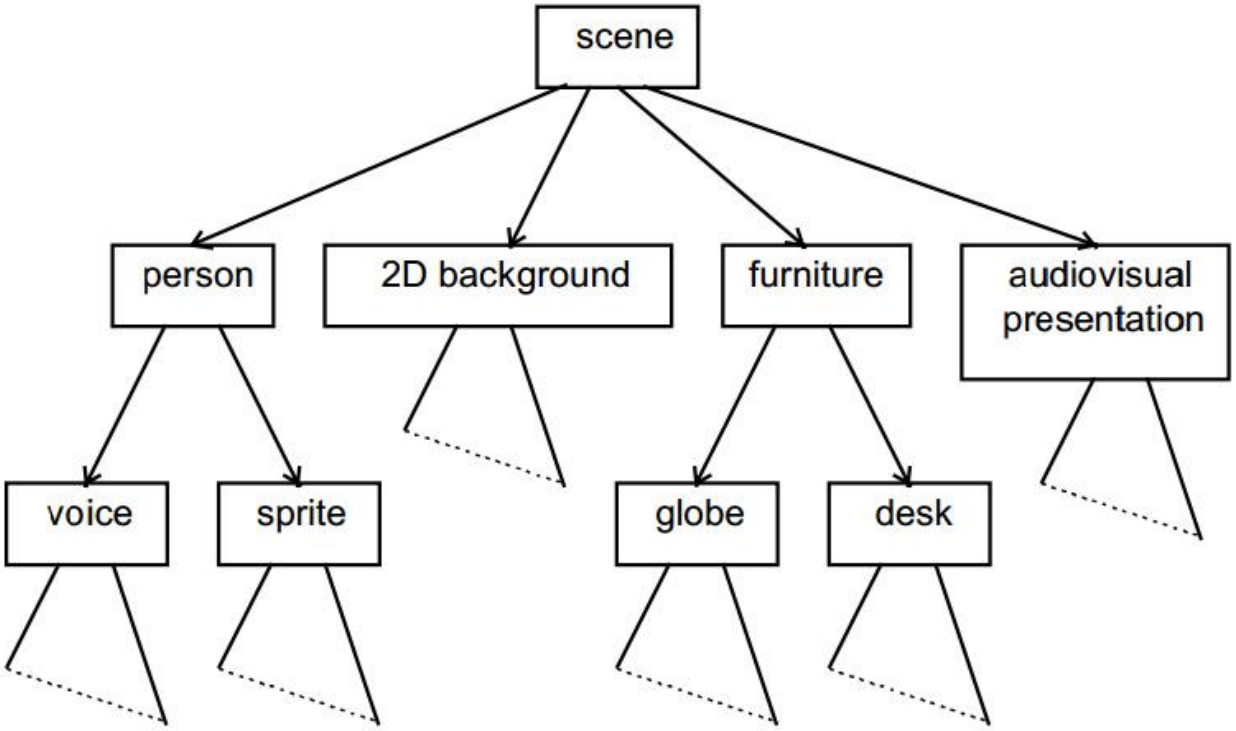

Figure 2-2 Representation of the example scene showing how compound media objects are structured [13]

Describing, multiplexing and synchronizing data

Often, media objects need to stream extra data in order to alter their internal attributes. Those data are transported in one or more elementary streams (a specific form that data must have when they leave a server in order to make them compliant with MPEG-4). Using such streams makes handling of meta-data or hierarchically encoded data easier. This is achieved by assigning a unique object descriptor to streams, which in turn is associated to one media object. [23]

Synchronization of data is achieved through the synchronization layer. Its job is to identify the type of access units contained in a stream (audio, video etc.) and manage their placement in time by reading the timestamps that they carry.

Interaction

A user, while viewing content that is created by "traditional" standards, is able to observe the scene in a way that the author had defined. MPEG-4 allows authors to allow users to interact with the scene. Some actions that users are allowed to perform are:

- Navigation in the scene by changing the viewing and listening point.
- Dragging various objects in the scene in different locations.
- Provoking events by interacting with objects in the scene, using the users' input devices.

2.3 MPEG-4 BIFS

After reading the previous section, it is made clear that MPEG-4 is much more than a simple audiovisual "carrier" with just two multimedia streams in it (one for the audio, one for the video). It is a very complex package, a scene if you prefer, that incorporates a large number of multimedia objects of various types (such as rectangular video, video with shape, audio, 3D objects, speech, text, graphics and many more).

But so many objects need to be somehow placed into some order (where to place each object, if they move, if they take user input, do they make sounds, sound effects etc.) so that the viewer/user gets a visually and aurally catching experience. In other words, there is the need for object composition into the scene. MPEG-4 has a system for describing scenes, called BIFS (BInary Format for Scenes).

As its name suggests, BIFS is a binary format for composing objects in MPEG-4 scenes. Scene data are transmitted into streams to the user's device. Additionally, the media server has the ability to insert or modify content directly into the scene, alter object properties (such as change colour or text etc.). Those modifications are made through specific commands that the BIFS system allows. BIFS content can be played back in different device platforms (for example Computers, Smartphones, Digital TV etc.), without the need of different encodings for each one of them. Furthermore, BIFS is compressed using a specialized algorithm that greatly reduces data volume and allows for quick transmission and delivery.

BIFS scene structure

BIFS uses the concept of trees/graphs. The scene graph contains nodes that represent various media objects (visible or invisible) in the scene. There are various types of nodes in a scene divided into two categories, grouping nodes and leaf nodes. Leaf nodes can contain graphical objects such as 3D meshes, textures, 2D shapes and many more. Grouping nodes, as their name suggests, are used to group together leaf nodes and/or other grouping nodes. This way, more complex object can be created into the scene and is easier to add, modify or delete many nodes together. All nodes contained in a BIFS scene are children of a single top-level parent grouping node.

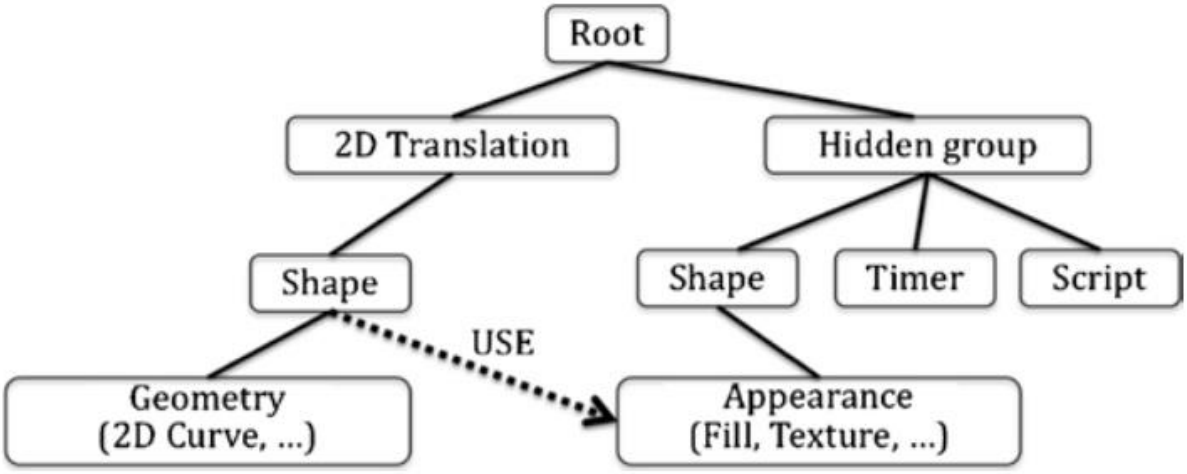

Figure 2-3 Example of a BIFS scene graph [18]

Every node in BIFS is classified into different types with a "system" called Node Data Type (NDT). With NDT, every node has a restriction on which nodes can be its parents. For example, a Cylinder can be child of a Shape node, which consecutively can be child of a 3D grouping node.

Another feature of BIFS is that every node in the scene can be assigned a unique (across the same scene graph) ID that can be an integer number or a string. This gives the ability to the server to find and modify or delete existing nodes.

Every node contains a set of attributes and properties (which in MPEG-4 terms are called fields) that configure various aspects of the node (for example text colour, font size etc.). Some of the data types of fields are:

- Integer
- Time
- Boolean
- Float
- Double
- Color
- Vector (2D, 3D, 4D)
- String
- Image
- Node

Every data type can be single field or multiple field. Single field mean that the field can accept only one value of that data type. Multiple field means that the field can accept an array of values of that specific data type.

Every field has an access method attached to it. Access methods define how the field's value can be used, if it is a read-only field or a write-only field. For read-only fields, the eventOut method applies. For write-only fields, the eventIn method applies. These methods are used when we want to transfer some value from one node to another (due to user interaction, time event or other sources of events).

This value transfer is made through the use of ROUTE nodes. Routes just take a value from a specific field of a node and put it in a specific field of the same data type in a different node. BIFS takes into account that sometimes it is needed to transfer a value from a field into another field of different data type and provides the Valuator node. Valuators take a field value and convert it to some other data type by adding or removing dimensions.

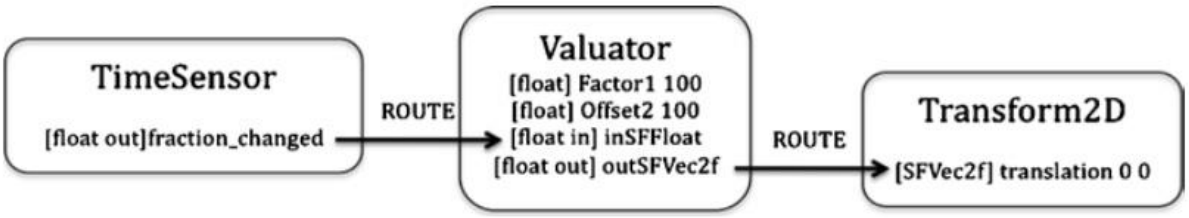

Figure 2-4 Operation of the Valuator node in BIFS [18]

Another feature of BIFS is that it enables developers to reuse nodes that they already have created anywhere in the scene. This leads to better file sizes while creating complex scenes. For example, a Cube that has some specific attributes (like size and texture) can be reused in another model, where the developer will change the size while keeping the same texture.

Finally, although BIFS has a broad set of supported nodes for a wide variety of uses, sometimes there is the need for nodes that don't exist. To address this issue, BIFS defines the PROTOTYPE node. Using PROTOTYPE, developers can create a complex node that is basically a grouping node. This grouping node contains a number of standard nodes defined by BIFS.

Media Objects composition

Media objects, as described above, are a group of BIFS nodes that define a visible (or not) 2D/3D object in the scene. The grouping node that defines a media object is the Shape node and it can display any type of graphics (2D or 3D), movies, images, text and so on. In other words, it defines the geometry of an object (as simple as a sphere, or more complex) as well as the appearance of it (colors, textures etc.).

The geometry node that can be used in Shapes can be any of the 3D primitives (such as cubes, spheres, pyramids etc.) or the 2D curves (such as lines etc.). In the other hand, appearance nodes define how the object is displayed visually. Some of the visual styles provided by BIFS are:

- Solid colours with alpha blending
- Gradients with opacity
- Image textures
- Video textures

- Blurring
- Texture transformations

Because BIFS deals with different types of graphics (2D and 3D), it brings together the two worlds and defines, for the scene, a coordinate system that has its origin at the center of the drawing area. Units in the coordinate system can be either pixels (in favor of the 2D content) or meters (used in 3D content). Grouping nodes define their own coordinate system for their children nodes. This means that most of the geometric nodes position their center right into the center of their parent's coordinate system. The text node is the exception to this; its origin is located on its first character.

Modifications to the default position of a node can be made with various ways. Transform grouping nodes can be used to apply transformations between the local coordinate system and the parent's system. Another way in changing the way nodes are displayed and positioned is through the use of nodes such as the OrderedGroup or the Switch, where they change the order in which the child nodes appear in the scene (thus affecting their positioning). Finally, there are nodes that define different layouts that use a relative positioning system for displaying objects in the scene (for example the Layout, PathLayout nodes and more).

Figure 2-5 Example of a BIFS scene [18]

Animating media objects

BIFS gives the ability to animate the various objects that a scene contains. There are two ways on animating objects: through the use of timers and using scene updates via the commands that BIFS provides.

TimeSensors enable the animation of objects through timers. Every TimeSensor produces events based on time. These events indicate the various states of the timer, including the time that elapsed since its activation, the fractions of time and the activation or deactivation of the sensor itself. Usually, values produced by TimeSensors are passed, through Routes, to interpolator nodes. Interpolator nodes are used to smoothly animate objects in the scene. They hold pairs of values (that are defined by the developer that created the scene) that express the fraction of time and the position of the object in that time. Interpolators accept as input the time fractions produced by TimeSensors, compare the input with their pre-defined pairs and choose the best value for the animation.

The other way of animating objects in the scene is through scene updates. These updates directly modify the contents of the scene (such as adding or removing nodes from the graph), and can be carried in the media stream that holds the scene itself or can be contained into their own stream. The latter, gives the ability to manipulate the stream's playback. Although scene updates give a greater control over the operations performed on the nodes, they don't rely on time for giving smoother animations. For example, scene updates are very good when used to display content similar to video subtitles or graphical commentary.

Interaction with BIFS objects

Like most of the standards for interactive multimedia, BIFS uses the events model to handle interaction between the user and the various objects in the scene. Events can be triggered from numerous sources like pointing devices (clicking, dragging, rotating etc.), keyboards, controllers, scene time events, and many more. BIFS defines a special group of nodes, called sensors, specifically for handling interaction events. Some of them are:

- TouchSensor
- PlaneSensor
- SphereSensor

- CylinderSensor
- Timer
- InputSensor

When an event is triggered, it is passed (using routes) to other nodes for further processing, and may trigger more events. Events processing may be done by:

1. Directly passing event value into nodes.
   This is the simplest way of processing an event. The value generated by the event is passed into the corresponding field of the target node.
2. Executing scene updates.
   BIFS defines the Conditional node. This node is programmed (by the developer) to perform scene updates (add, modify, delete nodes) when an event activates it. This way, developers have the ability to create complex interaction with the scene.
3. Scripting.
   BIFS supports the use of the Script node. Developers can attach code (in VRMLScript, a variant of ECMAScript that derives from JavaScript) into it that can handle very complex interactions.
4. MPEG-J Java framework.
   Finally, BIFS defines the MPEG-J framework that gives the ability to developers to connect the scene with the Java Virtual Machine in order to execute very advanced Java code. Through MPEG-J, the code can directly control various BIFS subsystems like the scene graph, network stack etc.

2.4 MPEG-4 Part 11 - XMT

As mentioned in the previous section, the main format that MPEG-4 uses to describe spatial and temporal 3D scene structure and interactions is the Binary Format for Scenes (BIFS). BIFS is a compressed, binary version of the VRML language. But this format doesn't offer much flexibility to authors, so MPEG-4 supports another, textual, format. The Extensible MPEG-4 Textual (XMT) format is used to represent MPEG-4 scenes using a textual XML syntax, and is directly derived from BIFS. Both XMT and BIFS are described in MPEG-4, Part 11: Scene Description [8][13].

By using the XML formatting, authors are able to provide their multimedia content to other authors or service providers. In addition, MPEG-4, X3D and SMIL interoperability is made possible through the XMT framework [19].

It is composed by two separate levels that define different forms of textual syntax and semantics. XMT-A contains and extends a subset of the X3D standard and uses XML syntax to represent the MPEG-4 content. XMT-Ω, as stated in [19], "is a high-level abstraction of MPEG-4 elements. The goal of XMT-Ω is to facilitate content interchange and provide interoperability with the W3C SMIL language".

Finally, a similar framework to XMT is LASeR (Lightweight Application Scene Representation). LASeR is also a (lightweight and 2D) rich media scene structure description framework that is based on the SVG standard and is targeted towards mobile devices.

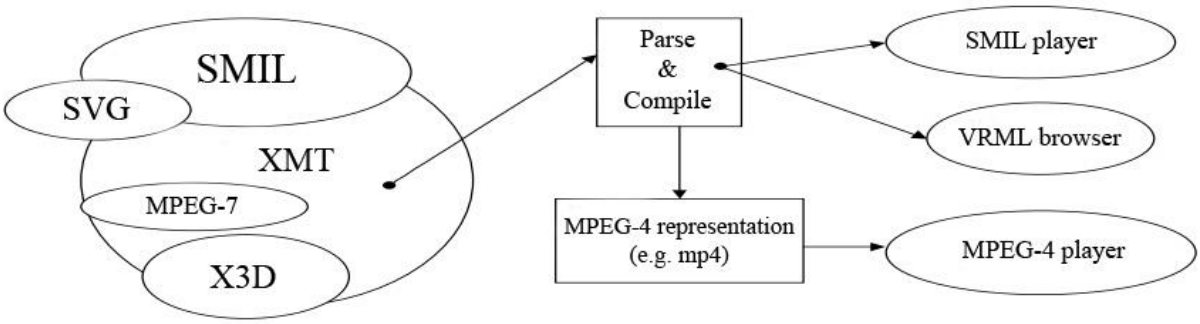

Figure 2-6 Basic representation of the XMT framework [13]

XMT-A

XMT-A is a textual format that uses XML syntax in order to represent MPEG-4 content in a flexible, straightforward and understandable way. It is an one-to-one mapping of the MPEG-4 binary format (BIFS). This means that every BIFS node and functions are completely supported by XMT-A. Additionally, many of the elements that are defined in XMT-A are a subset of the X3D standard, in order to provide some form of compatibility with it.

Basic XMT-A structure

Just like BIFS, XMT-A uses the scene graph concept to describe an MPEG-4 scene. All XMT-A files consist of a root grouping element (<XMT-A>) that is the starting point of the scene. Inside of it there are an optional Header element and a Body node. Header may contain various child nodes that provide different information to the player, configuring the environment in which the scene will be played back (like BIFS configuration nodes etc.).

Body node is, probably, the most important node in the XMT-A file. It contains all the nodes, routes, scripts (etc.) that the scene will display to the end user. As in BIFS, XMT-A supports grouping nodes (that group other nodes, including more grouping nodes, in order to form more complex objects in the scene) and leaf nodes (that can be 3D geometry nodes, 2D curves, textures, images etc.). Re-using already created nodes and prototyping new nodes are also supported in XMT-A.

Similarly to the binary format, unique IDs for each node are supported, along with the various attributes and fields that control the behavior and appearance of the nodes. Field data types are the same as BIFS (integers, strings, colours, Booleans, floats, etc.). Fields can be single valued or accept arrays of values, have access methods (eventIn and eventOut) and can communicate between them (or with scripts) using the Route nodes.

Media objects in XMT-A are composed using grouping nodes for shapes, materials etc. Shapes and object geometry can contain 3D primitives (such as cubes, spheres, pyramids etc.) or the 2D curves (such as lines etc.). Their visual representation is defined and/or modified by material, textures, colors etc.

The coordinate system of the scene is, just like BIFS, in the center of the drawing area, and every grouping node defines its own custom coordinate system for its children. Special nodes that alter the coordinate system of nodes or grouping nodes are supported.

For animation purposes, XMT-A supports the same ways as the binary format. TimeSensors are used to control the position, rotation, scale and so on. Additionally, scene updates can be used to modify/animate nodes directly into the scene.

Finally, interaction between the user and the various nodes of the scene is made by using sensors (TouchSensor, PlaneSensor, InputSensor, Timer, and many more). Processing the events that sensors produce is made with various ways (directly pass value using Routes, scene updates, scripting).

Table 1 Basic XMT-A example

```
<XMT-A>
<Body>
<Replace>
<Scene>
<!--  XMT-A elements go here  -->
</Scene>
<Replace>
</Body>
</XMT-A>
```

XMT-Ω

The XMT-Ω format was created to simplify and help content exchange, to bring ease of use and to be consistent with the Synchronized Multimedia Integration Language (SMIL). It does not describe how the objects in the scene are connected and how the user interacts with them. XMT-Ω operates at a higher level than XMT-A, and defines audiovisual objects, as well as their behavior through time and the relationship that every node has with the others. The relation and compatibility with SMIL provides all the necessary mechanisms in order to control timing of the objects, manage their layout and the transitions between the various objects.

Scientific research on MPEG-4 XMT-A/BIFS

Although MPEG-4 is widely used in multimedia content encoding and delivery, little scientific research has been done on BIFS and XMT (and other related parts of the MPEG-4 standard, such as LaSeR.). In the following paragraphs we present the most remarkable recent works in the field.

In 2005, Qonita Shahab presented in her thesis [34] a framework for transcoding XMT-A encoded scenes into MPEG LASeR scenes in order to be accessible by devices that have limited processing capacity (such as low end smartphones). The proposed framework is very similar to what we present in this thesis, it takes an XMT-A scene and passes it through a transformation XSLT processor that produces the final LASeR scene. The main difference is that it uses the Digital Item Adaptation tools of MPEG-21 in order to obtain useful information about the user's device capabilities. This enables the framework to adapt the various aspects of the initial scene so it can be played back as good as possible.

In 2010, Celakovski and Davcev proposed a system that enables real-time rendering of MPEG-4 3D scene by using the Microsoft XNA framework [38]. XNA is a complete library of tools that enable authors and programmers to create relatively easy multiplatform computer games. The system extracts all the 3D objects from the compressed MPEG-4 BIFS file and then converts them into data structures that are recognizable by the XNA framework.

In 2011 Kim and Ko proposed a method for converting MPEG-4 XMT-A 3D scenes into LASeR 2D scenes for playback on mobile devices [30]. They create 3D scenes using the XMT language, which in turn are passed to an XSLT converter in order to transform the scenes into valid LASeR scenes. The converter takes into account that LASeR can only describe 2D scenes, so it computes and convert the various coordinates and shapes of the 3D space into their 2-dimensional counterparts.

In the same year, Jovanova et. al. proposed an application that gives the ability to users to create in a simple way mobile mixed reality games [31]. Using this tool, authors can define regions of the real world that will be used afterwards in the game. The various locations are stored in a web server using the MPEG-4 format. Then, authors add the functionalities and logic of the end game, as well as the various images, 3D objects, media files etc. Finally, the tool compiles all the choices made by the author in an XMT file and generates the media files needed. Game players use a special application on their mobile device that tracks its position using GPS.

Koong et. al. proposed in 2012 an authoring tool that enables authors to create easily interactive XMT scenes, without any knowledge of XMT grammar, using an easy to use user interface and an intermediate scripting language [32]. First, they create a special document model that encompasses multiple (or just one) scenes inside it, and each scene holds a number of scenarios and cast groups. The Script Language ties all the different parts of the scene together and enables programming.

The same year, Carpentieri and Iannone created and presented an extension for the Impress application (part of the OpenOffice suite of applications, creates electronic presentations, similar to Microsoft Office PowerPoint) that converts electronic presentations into MPEG-4 compatible files, using XMT-Ω [35]. The system is developed in Java, UNO package (a framework that enables the creation of extensions by OpenOffice.org) and IBM Toolkit for Mpeg-4. UNO is used to parse the contents of the electronic presentation (an Impress presentation consists of several XML files and the various multimedia resources), the extension reads the xml files and generates the corresponding XMT-Ω structure and the IBM Toolkit for Mpeg-4 is used to compile the XMT-Ω into an MPEG-4 video file.

In 2013, Kim et. al. introduced the concept of a system that can deliver haptic data along with audiovisual content to the end users [36]. The proposed system uses the MPEG-V, standard that enables the "interaction and interoperability between virtual worlds and real worlds through various sensors and actuators" as stated in [36], for the description of the necessary data that will enable multimedia content that the end user can feel for real. MPEG-V, though, doesn't define any ways for transmitting and delivering the haptic data. For this reason, the authors used the MPEG-4 BIFS format by extending it (i.e. creating new nodes) in order to support the new type of data for haptic interaction.

Uherek et al. proposed, as stated in [33], "an XMT-based approach for the long-term preservation of digital videos in the context of Open Archive Information System infrastructures". Videos can be analyzed and then divided into primitive elements (such as frames, pixels etc.) that can be easily expressed using XML format. This paper proposes a way to extend the XMT file structure in order to be able to accommodate such information. This will result in a format that could easily be played back in the future by devices that don't support directly the source video format.

Finally, Robart, Plumari and Pau introduced a player application that is able to display and playback 3D animated models that conform to the MPEG-4 3DGC format [37]. 3DGC is a framework that offers the ability to authors to describe, compress and stream 3D content using the BIFS file structure. The application developed undertakes all the stages from parsing compressed MPEG-4 files to rendering in real time all the 3D models defined in the source file using OpenGL ES 2.0, while being able to run under various platforms such as Windows, Linux, Android and the Web.

2.5 Media containers for streaming multimedia content on the web

Extensible Binary Meta-Language

Extensible Binary Meta-Language (EBML for short) is an open standard that was specifically designed to be used in the Matroska project. EBML is designed as a very simple binary version of XML, so its syntax and semantic are separated. Each application, that features an EBML library, can read data based on this standard; the interpretation of the data is done by the application.

As mentioned on [1], advantages and disadvantages of EBML follow:

Table 2 EBML advantages and disadvantages

Advantages	**Disadvantages**
Upward compatibility when the format is updated	No equivalent to a DTD or Schema
Unlimited size of binary data	No entity can be defined
Only the space needed for the data is written	No include of external files

EBML Specification

As mentioned in the introduction above, EBML "inherits" the basic attributes of XML, meaning that there are no specific tags defined. Every application defines its own set of tags. EBML, though, defines a set of general data types to be used in the tags:

- **Signed integer**

  This is a big-endian integer number, either positive or negative, that has a size of 1 to 8 octets.

- **Unsigned integer**

  This is a big-endian integer number, without sign, that has a size of 1 to 8 octets.

- **Float**

  This is a big-endian float number that has a size of 4 or 8 octets (32 or 64 bits).

- **String**

  This data type defines a printable ASCII string that when it is necessary is zero padded.

- **UTF-8**

  This data type is similar to the String data type, except that it uses Unicode encoding.

- **Date**

  The Date data type is defined using a signed integer with size of 8 octets that indicates how many nanoseconds have elapsed since the 1st of January2001.

- **Master element**

  This is a container data type that can contain other EBML elements.

- **Binary**

  These are binary data that are not read by the application's parser.

EBML uses the concept of elements in order to form an EBML document. Those elements are defined using an Element ID, the size of the element and the binary data. An element can contain other elements in order to create a nested infrastructure in the document.

EBML defines a very small set of elements in order to construct an EBML document. The following table shows those elements:

Table 3 EBML elements semantic

Element Name	Class-ID	Type	Description
EBML	[1A][45][DF][A3]	Sub-element	Set the EBML characteristics of the data to follow. Each EBML document has to start with this.
EBMLVersion	[42][86]	u-integer	The version of EBML parser used to create the file.
EBMLReadVersion	[42][F7]	u-integer	The minimum EBML version a parser has to support to read this file.
EBMLMaxIDLength	[42][F2]	u-integer	The maximum length of the IDs you'll find in this file (4 or less in Matroska).
EBMLMaxSizeLength	[42][F3]	u-integer	The maximum length of the sizes you'll find in this file (8 or less in Matroska).
DocType	[42][82]	string	A string that describes the type of document that follows this EBML header ('matroska' in our case).
DocTypeVersion	[42][87]	u-integer	The version of DocType interpreter used to create the file.
DocTypeReadVersion	[42][85]	u-integer	The minimum DocType version an interpreter has to support to read this file.

Matroska Media Container

Matroska is a multimedia container format. That means Matroska is not a video or audio compression format, but a “box” that contains various multimedia data (streams) such as audio, video, graphics, text and many more. It is based on EMBL, so it is very flexible and robust [2].

Matroska integrates all the components that are necessary, in order to deliver rich multimedia services to the end users. Some of them are:

- Streaming over networks (either local or the internet) using a broad selection of protocols like HTTP, FTP, CIFS and many more.
- Modularly expandable
- Seeking in the file is very fast
- Multimedia streams can be divided into chapters
- Metadata are fully supported

- High error recovery so that the stream can continue to play even when it is damaged
- Support for menus just like DVD have
- Support for multiple audio, video and subtitle tracks that the user can change during playback

<table>
<tr><td colspan="6">Header</td></tr>
<tr><td colspan="3">EBML Version</td><td colspan="3">DocType</td></tr>
<tr><td colspan="6">Meta Seek Information</td></tr>
<tr><td colspan="6">SeekHead</td></tr>
<tr><td colspan="3">Seek</td><td colspan="3">Seek</td></tr>
<tr><td colspan="2">SeekID</td><td colspan="2">SeekPos</td><td>SeekID</td><td>SeekPos</td></tr>
<tr><td colspan="6">Segment Information</td></tr>
<tr><td colspan="6">Info</td></tr>
<tr><td colspan="3">Title</td><td colspan="3">SegmentUID</td></tr>
<tr><td colspan="3">PrevUID</td><td colspan="3">NextUID</td></tr>
<tr><td colspan="6">Track</td></tr>
<tr><td colspan="6">Tracks</td></tr>
<tr><td colspan="3">TrackEntry</td><td colspan="3">TrackEntry</td></tr>
<tr><td colspan="3">TrackUID</td><td colspan="3">TrackUID</td></tr>
<tr><td>Name</td><td>TrackNumber</td><td>TrackType</td><td>Name</td><td>TrackNumber</td><td>TrackType</td></tr>
<tr><td colspan="6">Chapters</td></tr>
<tr><td colspan="6">Clusters</td></tr>
<tr><td colspan="6">Cluster</td></tr>
<tr><td>BlockGroup</td><td>BlockGroup</td><td>BlockGroup</td><td>BlockGroup</td><td colspan="2">Timecode</td></tr>
<tr><td>Block</td><td>Block</td><td rowspan="2">Block</td><td rowspan="2">Block</td><td colspan="2" rowspan="2"></td></tr>
<tr><td>BlockDuration</td><td>ReferenceBlock</td></tr>
<tr><td colspan="6">Cueing Data</td></tr>
<tr><td colspan="6">Cues</td></tr>
<tr><td colspan="3">CuePoint</td><td colspan="3">CuePoint</td></tr>
<tr><td colspan="2">CueTime</td><td colspan="2">CuePosition</td><td>CueTime</td><td>CuePosition</td></tr>
<tr><td colspan="6">Attachment</td></tr>
<tr><td colspan="6">Attachments</td></tr>
<tr><td colspan="3">AttachedFile</td><td colspan="3">AttachedFile</td></tr>
<tr><td colspan="2">FileName</td><td colspan="2">FileData</td><td>FileName</td><td>FileData</td></tr>
<tr><td colspan="6">Tagging</td></tr>
<tr><td colspan="6">Tags</td></tr>
<tr><td colspan="3">Tag</td><td colspan="3">Tag</td></tr>
<tr><td colspan="2">MultiTitle</td><td colspan="2">Language</td><td>MultiTitle</td><td>Language</td></tr>
</table>

Figure 2-7 Basic EBML layout

Matroska container architecture

Matroska is based on EBML and has a layout that is similar to the previous figure [3].

- **Header**

  This section of the Matroska file contains information regarding the version of EBML that was used to create this file. Additionally, it is specified what type of EBML this file is (i.e. Matroska file). The Header must be present in the beginning of the file, as it is the first section that is read by the application in order to be able to know if it can read the file or not.

  - **EBML Version**

    This is the EBML element that defines the EBML version that the application must be able to read.

  - **DocType**

    This element contains the information regarding the type of the EBML file, a Matroska file.

- **Meta Seek**

  This section of the Matroska file stores the locations of the other primary components of the file and notifies the application about them. The Meta Seek is usually used only when the file is initially opened by the application and is not used when the user seeks for a specific part of the stream (this work is done by the Cues section). Additionally, Meta Seek is not required for the proper operation of the Matroska file, but then the application would have to search the whole file in order to find the data needed each time.

  - **SeekID**

    This field holds the Class-ID of the element that the application will search. For example, if the application wants to get the attached files, that reside in the file, it will search for a SeekID [19][41][A4][69] (this is the Class-ID of the Attachments section).

  - **SeekPosition**

    This element stores the byte position in the file of the component that is specified in SeekID.

- **Segment Information**

This section holds all the information that are necessary to correctly identify the Matroska file.

- **Title:** This is the title of the file
- **SegmentUID**

  This field stores an ID that is unique for a particular file, so that this file can be identified around the world. This is generated in random.
- **PrevUID, NextUID**

  A Matroska file can be part of a sequence of files that are meant to be played one after other. The PrevUID and NextUID elements store the unique ID of the previous and next file in the sequence.

- **Track**

  As mentioned before, Matroska supports the use of multiple tracks (audio, video, subtitles etc.) in one file. The Tracks section contains all the information about those tracks, such as the type, media information (sample rate, resolution etc.) and codec to be used. Additionally, it can store any private data that the codec needs for its operation.

  - **TrackUID**

    Each track in a Matroska file has a unique ID (similar to the SegmentUID). This element is used in Tagging section.
  - **Name**

    Indicates the name of a track.
  - **TrackType**

    Indicates what type of tracks is this (audio, video, subtitles etc.).
- **Clusters**

  This is where all the video (frames) and audio data reside. Although this section doesn't have any restriction on the amount of data or the duration of the data that are stored in it, most developers keep the size of each cluster under 5MB or the duration of it under 5 seconds.

  - **Timecode**

    This elements exists at the beginning of every cluster and specifies in which time the first block of the cluster should be played back.

- **BlockGroup**

  Inside every cluster there is one or more BlockGroups. Those store all the data that are associated with the block.

- **ReferenceBlock**

  This is used to indicate P/B-Frames. This element stores a timecode that points to the block needed. A Block can have multiple ReferenceBlocks, so it is possible to create exceptionally complex references.

- **CueingData**

  This is the section that holds all the cues, and index of "points" that facilitate seeking through the playback of a multimedia track. The use of CueingData is not mandatory, but without it the application will have to search through the entire file in order to find the requested timecode.

  - **CuePoint**

    One of the cue points in the index.

  - **CueTime**

    The time code that the CuePoint points.

  - **CuePosition**

    A list that holds the byte position of each of the tracks in the Matroska file.

- **Attachment**

  This section is used to attach any kind of files (such as pictures, programs, codecs, webpages, 3d content etc.) into a Matroska file. There is no limit in what you can attach or the size of it.

  - **AttachedFile**

    The element that holds all the data for an attached file.

  - **FileName**

    This is the name of the original file that is attached in the Matroska file.

  - **FileData**

    The data that compose the file.

- **Tags**

  This section is used to store data about the file, the tracks and the chapters, that it contains. It is similar in concept to ID3 tags found in MP3 files. Some examples

of the data stored are: script, writer, singer, titles, actors, directors, price and many more. Additionally, most of these can be stored in multiple languages.

WebM

WebM is a relatively new multimedia container format that is designed specifically for fulfilling the needs of the web for a totally free and open source multimedia file format. Like all the multimedia container formats (as the Matroska), WebM defines the structure that a WebM file should have (based on the Matroska multimedia container EBML structure) and what specific codecs must be used for encoding video and audio streams [4][5][7].

Video streams in a WebM file must be compressed using the VP8 video codec. Audio streams must be compressed using the Vorbis audio codec. No other codec can be used in the WebM format (unlike the Matroska). This choice was made with ease of use in mind. The development team wanted WebM to be supported by as many browsers and applications as possible. VP8 codec is supported in the majority of browsers so users don't have to worry about how could they playback WebM files.

WebM has many advantages, compared to other multimedia solutions for the web. First of all it is open, and that means that anyone can customize it and improve it. That is the very essence of the web itself where technologies like HTML, HTTP TCP/IP are open. Another major advantage is that it is developed and optimized especially for the web. It requires low computational power, so that playback is possible in a wide variety of devices such as smartphones, tablets, netbooks etc., its file format is very simple (due to the relationship with EBML/Matroska), it can deliver high quality real-time video and allows for a very easy and fast encoding process.

WebM container architecture

As mentioned previously, WebM container format is based on the Matroska format. In fact, they both use the same architecture with the difference that the WebM uses only a subset of the features/elements that Matroska defines. The following tables show some of the most important supported elements of Matroska by WebM. For a complete list of differences, see WebM Container Guidelines [6].

Table 4 Basic EBML elements support in WebM

Element Name	WebM	Description
EBML	**Supported**	Set the EBML characteristics of the data to follow. Each EBML document has to start with this.
DocType	**Supported**	A string that describes the type of document that follows this EBML header ('webm' in our case).

Table 5 Segment elements support in WebM

Element Name	WebM	Description
Segment	**Supported**	This element contains all other top-level (level 1) elements. Typically a Matroska file is composed of 1 segment.

Table 6 MetaSeek Information elements support in WebM

Element Name	WebM	Description
SeekHead	**Supported**	Contains the position of other level 1 elements.
Seek	**Supported**	Contains a single seek entry to an EBML element.

Table 7 Segment Information elements support in WebM

Element Name	WebM	Description
Duration	**Supported**	Duration of the segment (based on TimecodeScale).
Title	**Supported**	General name of the segment.

Table 8 Cluster elements support in WebM

Element Name	WebM	Description
Cluster	**Supported**	The lower level element containing the (monolithic) Block structure.
PrevSize	**Supported**	Size of the previous Cluster, in octets. Can be useful for backward playing.
Block	**Supported**	Block containing the actual data to be rendered and a timecode relative to the Cluster Timecode.

BlockDuration	**Supported**	The duration of the Block (based on TimecodeScale). This element is mandatory when DefaultDuration is set for the track. When not written and with no DefaultDuration, the value is assumed to be the difference between the timecode of this Block and the timecode of the next Block in "display" order (not coding order). This element can be useful at the end of a Track (as there is not other Block available), or when there is a break in a track like for subtitle tracks.
ReferenceBlock	**Supported**	Timecode of another frame used as a reference (ie: B or P frame). The timecode is relative to the block it's attached to.

Table 9 Track elements support in WebM

Element Name	**WebM**	**Description**
Tracks	**Supported**	A top-level block of information with many tracks described.
TrackType	**Supported**	A set of track types coded on 8 bits (1: video, 2: audio, 3: complex, 0x10: logo, 0x11: subtitle, 0x12: buttons, 0x20: control).
CodecName	**Supported**	A human-readable string specifying the codec.
Video Start		
Video	**Supported**	Video settings.
StereoMode	**Supported**	Stereo-3D video mode. Supported Modes: 0: mono, 1: side by side (left eye is first), 2: top-bottom (right eye is first), 3: top-bottom (left eye is first), 11: side by side (right eye is first) Unsupported Modes: 4: checkboard (right is first), 5: checkboard (left is first), 6: row interleaved (right is first), 7: row interleaved (left is first), 8: column interleaved (right is first), 9: column

		interleaved (left is first), 10: anaglyph (cyan/red)
AspectRatioType	**Supported**	Specify the possible modifications to the aspect ratio (0: free resizing, 1: keep aspect ratio, 2: fixed).
FrameRate	**Supported**	Number of frames per second. Informational only.
Video End		
Audio Start		
Audio	**Supported**	Audio settings.
SamplingFrequency	**Supported**	Sampling frequency in Hz.
Channels	**Supported**	Numbers of channels in the track.
Audio End		

Table 10 Cueing Data elements support in WebM

Element Name	**WebM**	**Description**
Cues	**Supported**	A top-level element to speed seeking access. All entries are local to the segment.
CuePoint	**Supported**	Contains all information relative to a seek point in the segment.
CueTime	**Supported**	Absolute timecode according to the segment time base.

Table 11 Attachment elements support in WebM

No support for Attachments in WebM multimedia format.

2.6 Extensible 3D Graphics (X3D) and X3DOM

X3D is an architecture that provides the entire infrastructure needed in order to create, store, restore and reproduce in real time 3-dimensional scenes and objects using the XML syntax. It is based and improves VRML (the first 3D interchange standard that enabled 3-dimensional content to be displayed over the web) and besides rendering 3D scenes and animations, it offers the ability to use scripts (using ECMAScript or Java), node prototyping, adds more data

encodings etc [11] [24]. X3D is officially incorporated within the MPEG-4 multimedia standard as stated in [24].

The biggest advantage of X3D is that XML is the core that powers it. Its textual format is very lightweight and contains a number of nodes that are used for hyperlinking, complexity adaptation etc. This facilitates the exchange of 3D data with web services and applications, as well as the handling of the data using programming languages like C++ or Java [24]. The main disadvantage, though, of X3D is that it requires some form of add-on or stand-alone browser in order to display the 3D content.

Table 12 Simple X3D example

```
<X3D>
<Scene>
<!--  X3D elements go here  -->
</ Scene >
</X3D>
```

X3D file structure

The general X3D file structure is (as stated in [11]):

- File header
- X3D header statement
- Profile statement
- Component statement
- META statement
- X3D root node
- X3D scene graph

X3D file header

This section of an X3D file holds all the necessary information (optional and required) about the X3D Scene and its capabilities. This is not a visible part of the scene or a specific

element or attribute. Rather, it is an abstract group of nodes and attributes that form the file header.

X3D header statement

This section of an X3D file contains information regarding the X3D version used, the text encoding and the X3D identifier. All X3D files are essentially XML files, so they have to follow the syntactic rules of XML. Thus, in the header statement we find the <?xml ?> declaration, that defines the XML version and the text encoding that the file follows (generally, X3D uses UTF-8 text encoding).

After this declaration, the validation mechanism follows, using the DOCTYPE statement. This section is not required but it is recommended. The DOCTYPE statement holds the Document Type Definition file path as well the X3D identifier and version number.

Table 13 Example XML declaration and DOCTYPE statement found in X3D files

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.1//EN" "http://www.web3d.org/specifications/x3d-3.1.dtd">
```

Profile statement

Following the declaration of the XML version and DOCTYPE, the root node must be present. The root node holds information about the profile of X3D used and the XML Schema reference for validation purposes. The profile statement in the root node is required.

The X3D specification defines numerous profiles. Each of these profiles specifies what subset of the functionality is available to an X3D player. This helps developers and system builders to create players that support specific functionality, reducing the complexity that would result if they tried to enable all the features of X3D.

Table 14 Example of the X3D root node

```
<X3D version="3.1" profile="Immersive"xmlns:xsd="http://www.w3.org/2001/XMLSchema-
instance"xsd:noNamespaceSchemaLocation="http://www.web3d.org/specifications/x3d-3.1.xsd">
```

Each of these profiles contains the full set of the previous profile and defines some new functionality. The profiles defined in X3D are:

- **Core**

  This profile gives the absolute essential definitions that an X3D player requires and is not defined for normal use.

- **Interchange**

  The Interchange profile supports all the basic geometric nodes, such as primitives and polygons, appearance nodes, such as materials and textures, and keyframe animation nodes. It is designed to be used by 3D applications that exchange 3D data between them.

- **Interactive**

  This profile has the same level of support as the Interchange and adds all the necessary nodes that enable the interaction of the users with the 3D scene.

- **MPEG-4Interactive**

  The MPEG-4Interactive profile holds the same functionality as the previous profile, but it was specifically developed to address the need of the MPEG-4 for 3D graphics.

- **CADInterchange**

  This profile was designed to be used in CAD applications. With this in mind, it supports most of the functionality of the Interchange profile and adds extra nodes that are specific to CAD applications.

- **Immersive**

  Immersive profile supports the full set of functionality of the Interactive profile and adds plenty of new nodes, such as 2D geometry, events, effects and many more. It was designed to support most of the functionality of VRML97.

- **Full**

  This profile supports every node that is defined in the X3D specification and adds numerous of new nodes, such as Humanoid Animation, NURBS and many more.

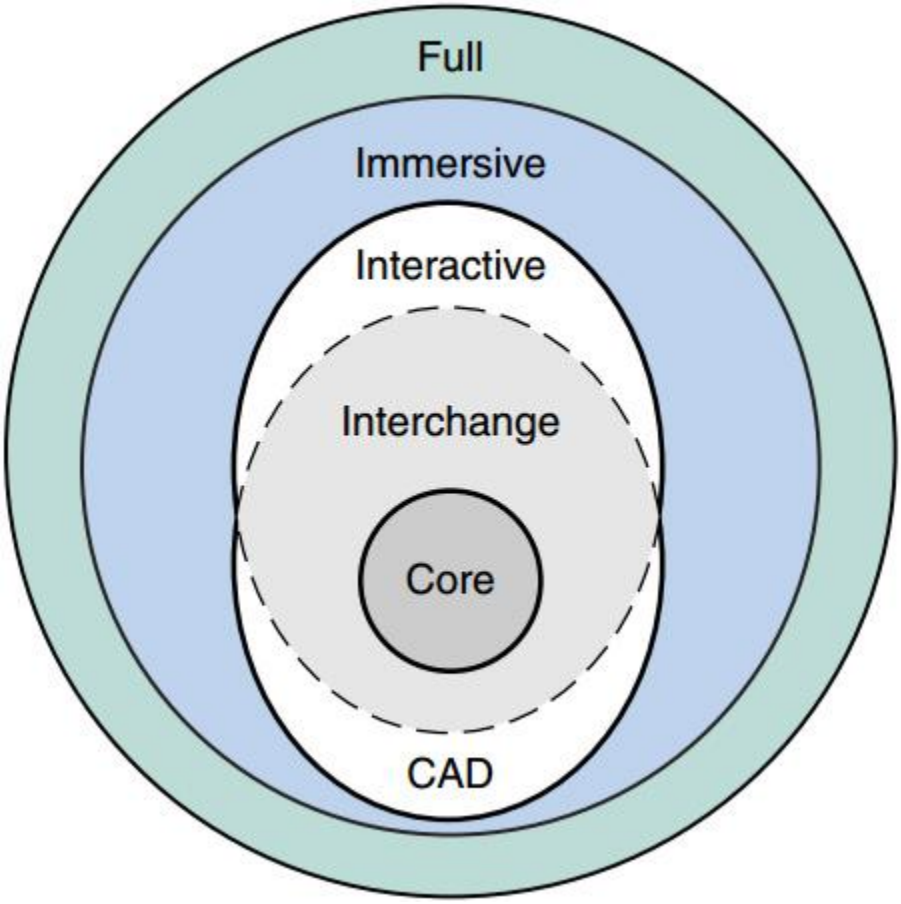

Figure 2-8 Simple diagram showing some of the X3D profiles [14]

Component statement

Sometimes we need to define some information that the player will read and react accordingly. For this reason, right after the X3D root node, we define the head section. Components (metadata as well) are placed inside <head>.

As mentioned previously, X3D is divided into profiles with different functionality support. Each of these profiles is divided as well into components, and each component is divided into levels. Those components (along with their levels) define a specific functionality support. Each of the nodes in X3D belongs to a distinct component.

Components are used to configure the X3D player to support extra functionality than this that the specified profile offers. It is possible to have multiple components in the head. Moreover, they are optional.

Table 15 Example of declaration of components in X3D

```
<head>
        <component name='DIS' level='1'/>
        <component name='Geospatial' level='1'/>
        <component name='H-Anim' level='1'/>
        <component name='NURBS' level='4'/>
</head>
```

Table 16 Components and component levels supported in X3D

Components	Interchange Profile Levels	CAD Interchange Profile Levels	Interactive Profile Levels	Immersive Profile Levels	Full Profile Levels
CADGeometry		2			2
Core	1	1	1	2	2
Cube map environmental texturing					3
Distributed interactive simulation					1
Environmental effects	1		1	2	3
Environmental sensor			1	2	2
Event utilities			1	1	1
Geometry2D				1	2
Geometry3D	2		3	4	4
Geospatial					1
Grouping	1	1	2	2	3
Humanoid animation					1
Interpolation	2		2	2	3
Key device sensor			1	2	2
Lighting	1	1	2	2	3
Navigation	1	2	1	2	2
Networking	1	1	2	3	3
NURBS					4

Pointing device sensor			1	1	1
Programmable shaders					1
Rendering	3	4	2	3	4
Scripting				1	1
Shape	1	2	1	2	3
Sound				1	1
Text				1	1
Texturing	2	2	2	3	3
Texturing3D					2
Time	1		1	1	2

META statements

META statements hold information that describes the X3D scene that follows. They are optional and if used they are placed inside of head. They are comprised of a name and a value (called content). META data can contain all sorts of information, such as copyright, references etc.

Table 17 Example of META statements in X3D

```
<head>
        <meta name='description' content='*enter description here, short-sentence summaries preferred*'/>
        <meta name='author' content='*enter name of original author here*'/>
</head>
```

X3D scene graph

After defining all the necessary information regarding the X3D file (profile, components, version etc.) we insert the scene node. The scene contains all the nodes that create our virtual world.

Scene Access Interface (SAI)

There are many times that developers want to give users the ability to interact with the 3D scene not just by linking sensors to 3D objects, but using custom code, either from external applications or using the X3D scripting node. For this purpose, X3D defines an interface that

enables the manipulation of the 3D scene without being “directly part of the scene graph itself” [19]. SAI gives five different types of access into the scene:

1. Access to the various functions of the X3D browser/player
2. Access to notifications about events that the browser/player raises (for example starting up, shutting down, not finding URLs etc.)
3. Permission to send events to nodes inside the 3D scene that allow such input
4. Permission to read values from nodes inside the 3D scene that allow such output
5. Access to notification about alterations made to values inside the 3D scene by events.

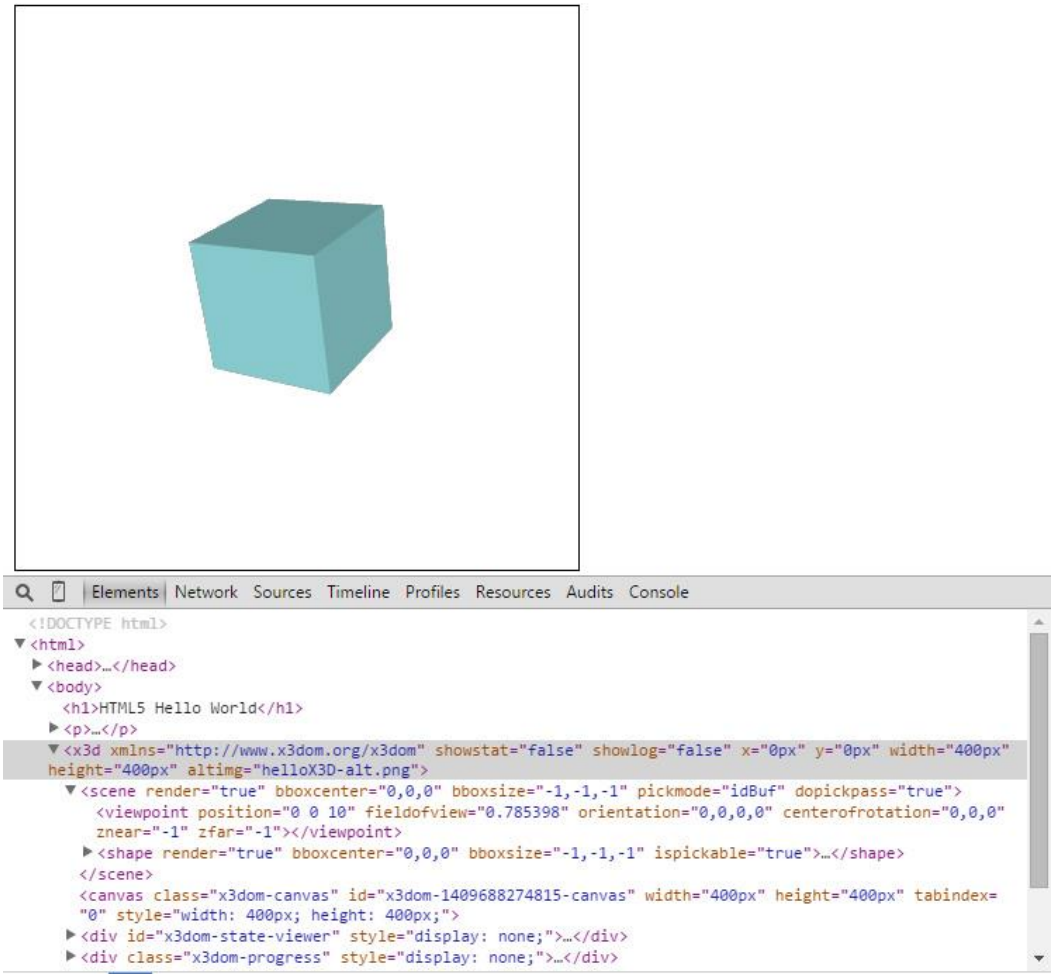

Figure 2-9 X3DOM example running in web browser with its code visible

The X3DOM framework

As mentioned above, X3D is the ideal infrastructure to display 3D content on the web, but it has a drawback. It needs some form of player or browser plugin in order to be played back. However, users are now accustomed to just open their web browser and be able to view and playback the majority of web content. Moreover, HTML5 declares that X3D can be used to display 3D content inside of web pages, but does not define the way that the DOM can be connected with the 3D content for further manipulation[12].

With this in mind, the Fraunhofer IGD created X3DOM (pronounced X-Freedom). X3DOM is a framework that gives the ability to web developers to embed X3D elements and scenes into the HTML5 DOM tree. Using this framework, a developer can create an HTML5 web page that can accommodate live interactive X3D scenes that can be manipulated (add, remove or change elements) easily. Technically, X3DOM fills the gap between the X3D content and the HTML5 DOM and connects them.

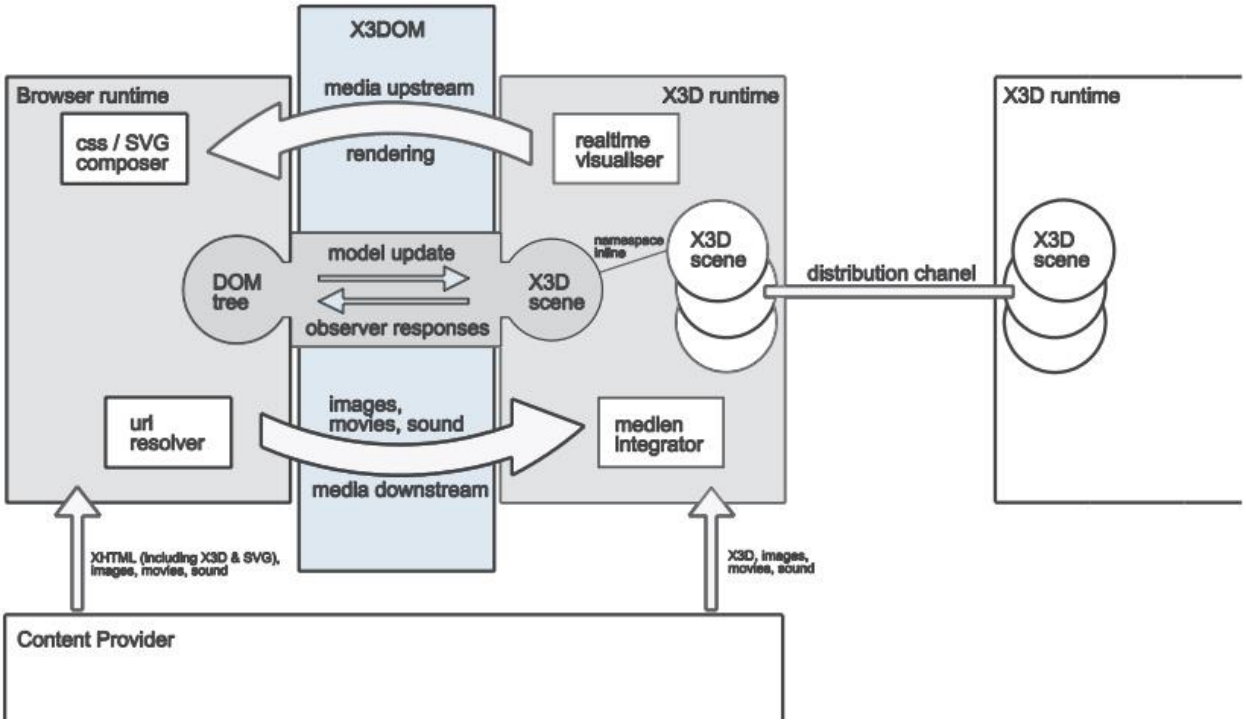

Figure 2-10 Diagram showing the X3DOM components [15]

Basic X3DOM operations

The “Connector Component” is essentially the heart of X3DOM. It is what connects the web browser with the underlying X3D scene. X3DOM doesn’t manipulate the HTML5 DOM but rather it uses an X3D tree representation. Every alteration made to the X3D tree is being reflected/synchronized to the DOM in order to be rendered in the browser. Likewise, any modifications that are made on the DOM nodes (like creating new nodes or removing old ones) are reproduced on the X3D tree. All these operations and “communication” between the two systems are performed by the X3DOM frontend and backend components.

Additionally, it is connector’s duty to handle all media (images, movies and sounds) upstream and downstream, in order to integrate them successfully into the X3D representation, and to be able to pass those media streams to the backend for further processing.

Another important component of X3DOM are the “Observer Responses”. An observer has the ability to manipulate the X3D tree depending on time or user interaction (for example, Sensors produce events based on time or user interaction).

One more feature that makes X3DOM so powerful is the fact that it supports segmented 3D scenes. This is very useful because authors can create very complex 3D scenes that are divided into segments that can be loaded just when they are needed. This results in small initial scenes that can be loaded more easily and fast. Additionally, the different parts of the “big” scene can support different X3D profiles. All this functionality is provided by the Inline node.

Finally, The X3D node that resides inside of the HTML5 web page has to provide a “single point of access”, meaning that it must be the node from which developers gain access to all the properties and functions that the X3DOM provides for that 3D scene. This is done by defining specific attributes that the X3D element holds that configure various aspects of the scene. Additionally, this node is “connected” to an X3D object that allows access to the SAI (Scene Access Interface) [16] of the X3D in order to manipulate the scene via code.

Chapter 3 - HTML5 video player with In-Stream 3D Interactive Scenes display capabilities

Aim of this thesis is the integration of XMT-A scenes within web contexts and their combination with HTML5 video, for the creation of a framework that will be able bring MPEG-4 scenes natively to the web. We chose to use the WebM video format to this end, as it demonstrates greater compatibility with most modern browsers today.

The methodology we introduce is based on embedding MPEG-4 (XMT-A) 3D content within WebM files, using the *Attachments* feature of the Matroska standard. Using our approach, 3D content transmitted in this manner can then be displayed over the video playback at a given time point and for a limited duration (both of them are specified in the 3D content). In this chapter we will present the full process, from the collection of the XMT-A files used in our experiments, to the merged playback of video and 3d within web environments.

3.1 – Process overview

The operation of the system can be broken down to a number of discrete steps, each with its own considerations:

1. 3D content in XMT format is integrated in a WebM file, and the composite multimedia file, still valid under the Matroska standard, is uploaded on a server to be accessed through the Web.
2. A client requests the WebM files from the server through the interface of the WebM player.
3. The server extracts the XMT file from inside of the WebM file and sends it to the client.
4. When the client receives the whole XMT file, it converts the 3D scene content to X3D using XSLT transformations, while storing independently the additional multimedia information (such as start-end times, or meta-information) contained in the XMT multimedia object, in relevant variables.
5. When the 3D content is ready to be displayed, the WebM player starts the playback of the video stream.

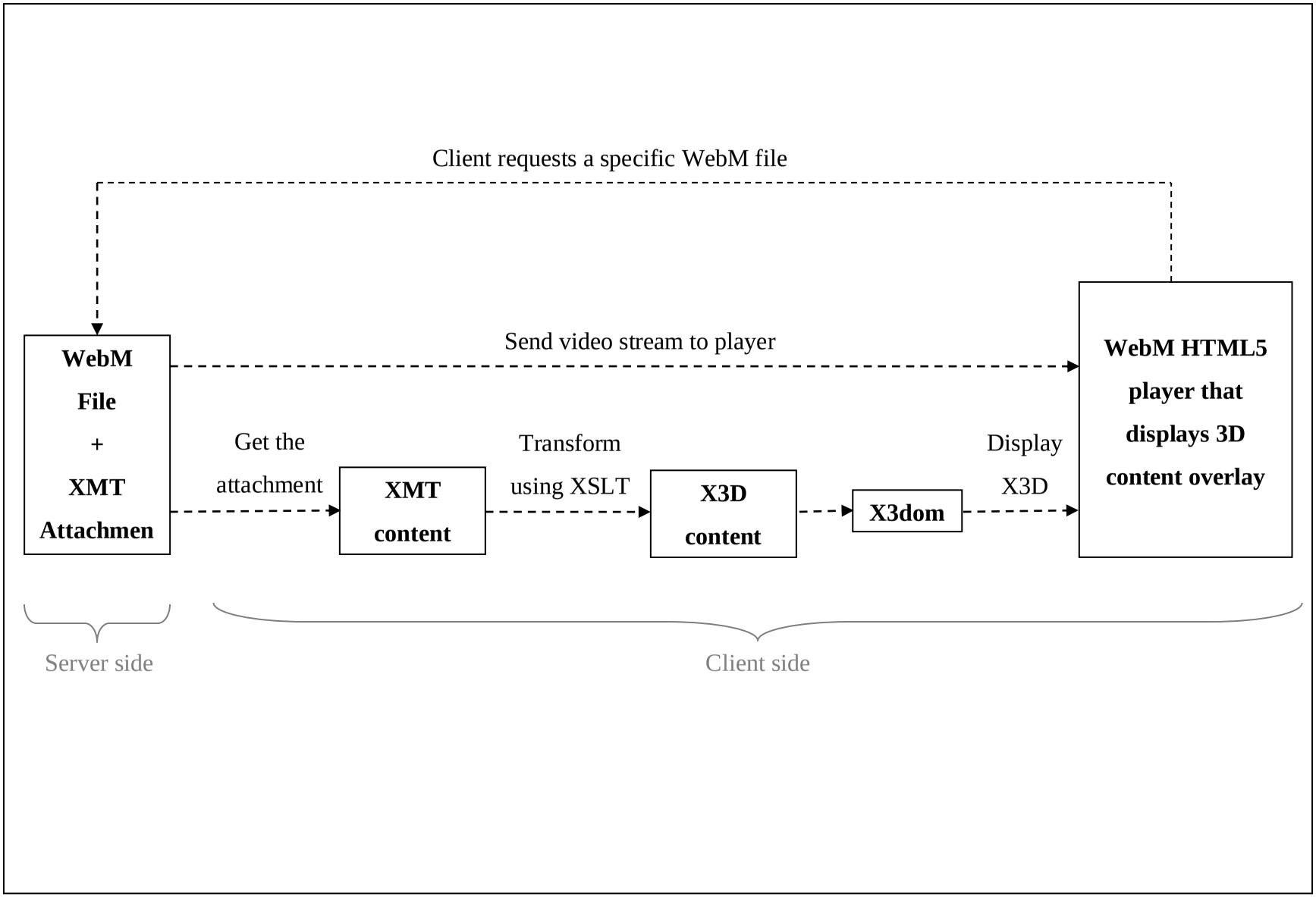

Figure 3-1 Operation of the HTML5 WebM player

6. When the video playback time reaches the value of the start value for the 3D scene, the X3D scene is displayed (using the X3DOM framework) overlaid on the video playback.
7. When the time reaches the duration defined in the XMT-A structure, the X3D content is removed from the overlay.

Figure 3-1 shows an overview of the architecture of the entire process. The rest of this section is dedicated to describing each step of the process in detail, presenting the problems and considerations faced at each stage, and the proposed and implemented solution.

3.2 – Creation of integrated XMT-WebM files

Collecting XMT-A files

The XMT-A standard has seen little use by developers so far. This can, in part, be attributed to the absence of popular players, as well as its incompatibility (up to this point) with web applications – even before the advent of HTML5-compliant X3DOM, there existed a number of browsers / players, as well as web browser plugins for X3D. While the need to install a specialized plugin did restrict the broad applicability, because of its reduced user-friendliness, at least among researchers and professionals, the availability of players and plugins contributed to the creation of a tight X3D community and the creation of large, supported datasets. XMT-A, in contrast, had almost nothing comparable in terms of software and web integration and, as a result, saw much less development.

As a result, there is a notable absence of XMT-A material to be found today. Even for research purposes, it was impossible to find XMT-A scenes with which to work and test our system. The approach followed instead was to create such files using the GPAC suite of tools [10][13][27][28]. GPAC offers -in theory- conversion capabilities from X3D to XMT-A, and vice versa. However, both directions of the conversion process are flawed due to software errors. The scenes resulting from any direction of these conversions using GPAC are usually syntactically wrong and cannot be played by any player. In order to generate a test set of XMT-A scenes, the solution we chose was to pick a number of indicative X3D scenes, convert them to

XMT-A using GPAC, and then manually correct them to have them properly conform to XMT-A. The resulting collection of XMT-A files derived from this semi-manual process was used as a test dataset for experimentation. The difficulty in even acquiring a basic XMT-A dataset to experiment with is indicative of the significance of our work. While XMT-A is an important part of MPEG-4, bringing together 3D scene structure and video, the lack of web integration, and, in fact, even stand-alone players, has left the standard in relative neglect. Through our work, we expect to give XMT the broad applicability it was designed to have.

Creating a WebM file with 3D content attached

As explained in the previous chapter, WebM is a subset of the Matroska multimedia container. Matroska supports the integration of any file (of any data type) into the file data stream via the Attachments mechanism. On the other side, WebM doesn't officially support this functionality. Due to its open nature, however, it is fully possible to insert into the WebM file structure all the Matroska elements that enable the Attachments functionality, without actually invalidating the WebM file itself. So, the first step is to create a WebM file that has a valid XMT file attached and upload it on a server.

Table 18 Part of an XMT that was embedded in the demo WebM video file

```
<XMT-A>
<Header>
 ...
</Header>
<Body>
<Replace>
<Scene>
<par begin="4s" duration="30s">
<Group>
<children>
                ...
<Transform DEF="N1" rotation="0 0 0 0" scale="1 1 1" scaleOrientation="0 0 0 0" translation="0 -19.118 -29.118">
<children>
<Shape>
<appearance>
<Appearance DEF="N2">
<material>
<Material diffuseColor="0.882 0.776 0.341"/>
</material>
</Appearance>
```

```
</appearance>
<geometry>
<IndexedFaceSet DEF="N3" coordIndex="..." solid="false">
<coord>
<Coordinate point="..."/>
</coord>
<normal>
<Normal vector="..."/>
</normal>
</IndexedFaceSet>
</geometry>
</Shape>
</children>
</Transform>
    ...
<TimeSensor DEF="N13" cycleInterval="3.33333" loop="true"/>
<PositionInterpolator DEF="N14" key="0 0.01 0.02..."/>
<OrientationInterpolator DEF="N15" key="0 0.01 0.02 ..."/>
    ...
</children>
</Group>
</par>
<ROUTE fromNode="N13" fromField="fraction_changed" toNode="N14" toField="set_fraction"/>
<ROUTE fromNode="N14" fromField="value_changed" toNode="N1" toField="translation"/>
  ...
</Scene>
</Replace>
</Body>
</XMT-A>
```

In order to create the WebM files, we used MKVToolNix, described as “a set of tools to create, alter and inspect Matroska files”[9]. MKVToolNix (among many things) creates MKV and WebM files that contain files in the Attachments section. Essentially, we present a variant of the typical WebM files which, while retaining compatibility with the HTML5 <Video> element, can carry the additional MPEG-4 compliant interactive 3D information we want for integration of XMT-A, HTML5 and web video. At the same time, the resulting file is a valid WebM video, playable as an HTML5 Video element. Essentially, we have discovered that the video players are blind to the Attachment part of WebM files, and, as a result, they can be used as a carrier for any form of data, even possibly information that is not directly related to the multimedia content. In our case however, they are used for transferring 3D data which are to be merged with the video during playback.

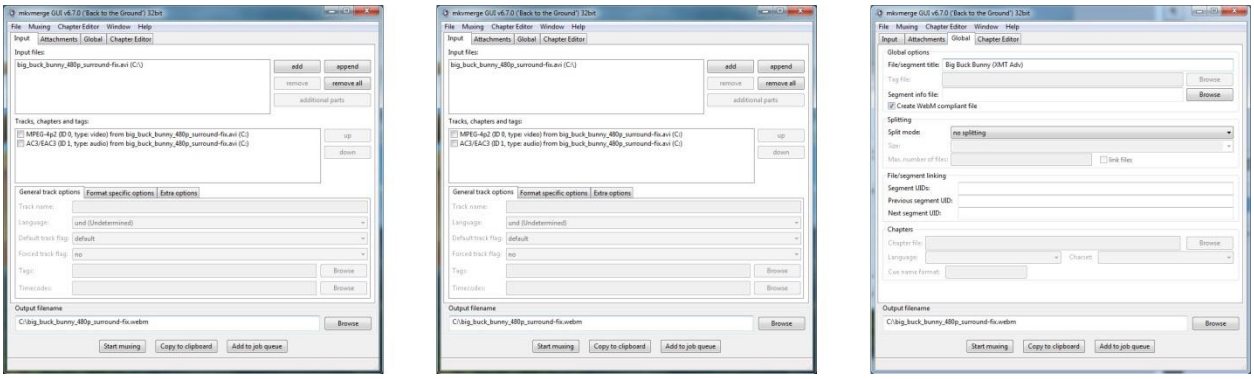

Figure 3-2 MKVToolNix Screenshots

3.3 – Client request for WebM file

In order for the client to receive and playback WebM files that have XMT content attached, it has to request it from the server using a simple AJAX POST request through a service. In the request, the client must define the filename of the WebM file, so that the server knows which file to deliver.

The fact that the entire framework is based on native HTML5 functionalities means that it can be easily integrated in any web platform with minimal adaptations. To demonstrate this, and to present a minimal working example for testing, we developed a test implementation of the system. The experimental/demonstration page we developed includes a text field, a button, a video element and an X3DOM scene. The text field and button are used in order to request a WebM video file from a web server. The video element is used to display the requested video stream, whereas the standby X3DOM scene is used to display the 3D content that is attached into the WebM file. Figure 3-3 The simple web page used to demonstrate the system shows the graphical user interface of the experimental client page/player we developed, while Table 19 contains a part of the HTML5 code behind the client, showing the input controls and the video/3D player.

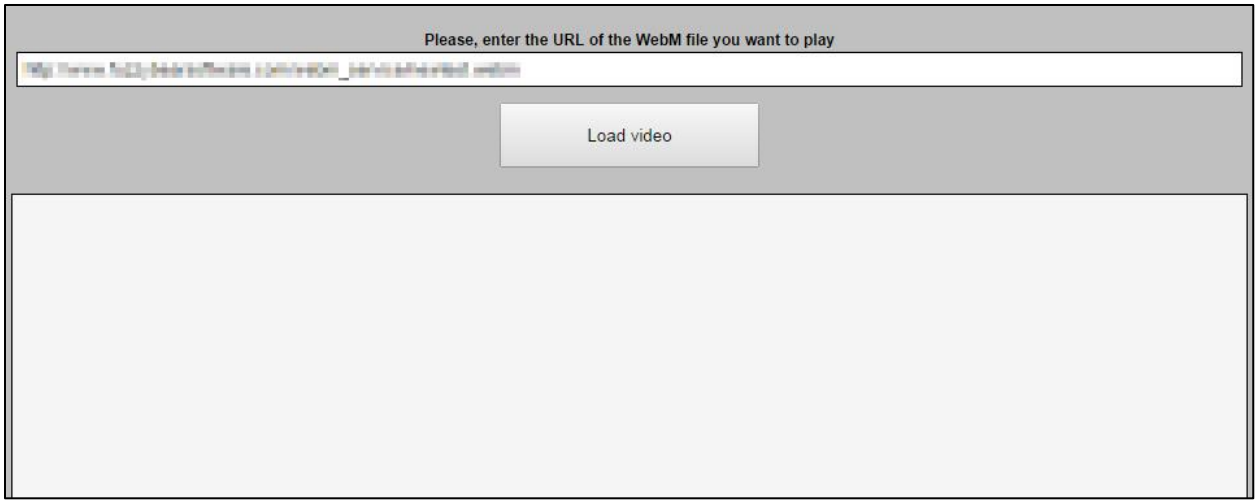

Figure 3-3 The simple web page used to demonstrate the system

Table 19 Part of the HTML5 code of the demonstration web page

```
<!DOCTYPE html>
<html>
	<body>
		<div>
			<!-- URL text box and button -->
			<div>
				Please, enter the URL of the WebM file you want to play
			</div>

			<div >
				<input id="video_url" type="text"></input>
				<input type="button" value="Load video" id="cmdLoadVideo"></input>
			</div>
			<!-- /URL text box and button -->

			<!-- Video player with 3d content overlay -->
			<div>
				<video id="videoPlayer"></video>

				<div id="3d_content">
					<x3d>
						<scene id="the_scene" ></scene>
					</x3d>
				</div>
			</div>
			<!-- /Video player with 3d content overlay -->
		</div>
	</body>
</html>
```

3.4 - Extraction of the XMT content from the server

When the web service receives the client's request, it searches for and retrieves the requested WebM file from its storage database. The file contains both the video and the XMT-A

information. Upon retrieval, the server proceeds to read the contents of the WebM file and isolate the XMT content within it. To this end, it uses the getID3 php library [14], using which the XMT code is isolated. This code is packed in an MPEG-4 XMT-A document containing all the interactive 3D information and all related metadata. The document thus created is then sent back to the client for processing and integration of the XML 3D scene into the page. Table 20 shows part of the PHP code we use to isolate the 3D scene information from the WebM file prior to transmitting it to the client.

Table 20 Part of the PHP code used in the server side to extract the XMT from the video

```
for($i = 0; $i < count($variable["matroska"]["attachments"]); $i++)
{
	if($variable["matroska"]["attachments"][$i]["FileMimeType"] == "application/xml")
	{
		$returnstring = $variable["matroska"]["attachments"][$i]["FileData"];
		break;
	}
}

return $returnstring;
```

3.5 – Transforming XMT to X3D

Relation between XMT-A and X3D

We have already explained that, in order to display MPEG-4 XMT content in HTML5 environments, we chose to exploit the inherent similarities between XMT-A and X3D, and use the existing X3DOM framework in order to display XMT code within HTML5 web pages, over WebM video. As we previously mentioned, there exists a particularly close relationship between XMT-A and X3D. Essentially, XMT-A presents itself as a subset of the X3D language, offering almost all functionalities that X3D does, using the same XML elements, and most importantly, the same scene tree structure architecture.

However, a number of subtle differences exist between the two standards, which render them, for all practical purposes, incompatible. This means that there is a significant chance that an XMT-A scene will be seen as syntactically wrong by an X3D player, and vice versa.

In the process of attempting to create a player for XMT files based on X3DOM, we had to implement a converter from XMT documents to X3D. As a first step, we proceeded to create the first documented comparison of the syntactic differences between the two standards. Although the reasons why XMT-A did not follow the exact complete structure of X3D are unknown, our work is, to our knowledge, the only attempt so far to index the differences between the two, towards unification of the two standards.

The differences consist of two general categories: one is the presence of syntax differences in various nodes (for example, in XMT-A, between a <Group> element and its Scene Tree children, there has to exist an additional <children> node, in contrast to X3D, where the children of a Group node are placed directly within the <Group> element), and the other is the absence of certain nodes from one or the other standard (e.g. X3D does not support <DiscSensor> elements, while XMT-A does not support <TriangleFanSet> elements). As part of our research effort to make X3D and MPEG-4 XMT-A compatible and interchangeable, we proceeded with an exhaustive analysis of the two standards, and the identification of their similarities and differences. This is the first step of our work, and the first contribution of our work, corresponding to a design phase in the project. Table 21 displays an organized listing of all the major similarities and differences between the two standards.

Table 21 Similarities and difference between XMT-A and X3D

Node type	**X3D**	**XMT-A**
Root node	<X3D>…</X3D>	<XMT-A>…</XMT-A>
Header node	<head>…</head>	<Header>…</Header>
Scene description node	<Scene>…</Scene>	<Body> <Replace> <Scene> … </Scence> </Replace>

		</Body>
Grouping node	<Group>…</Group>	<Group> <children>…</children> </Group>
Viewpoint configuration node	<Viewpoint />	
Navigation config node	<NavigationInfo />	
Shape definition node	<Shape>…</Shape>	
Appearance definition node	<Appearance> … </Appearance>	<appearance> <Appearance> … </Appearance> </appearance>
Material definition node	<Material /> 2D material node not supported	<material> <Material /> <Material2D /> </material>
Texture definition nodes	<ImageTexture /> <MovieTexture /> <PixelTexture /> <TextureTransform /> <TextureCoordinate /> <TextureCoordinateGenerator/>	<texture> <ImageTexture /> <MovieTexture /> <PixelTexture /> <TextureTransform /> <TextureCoordinate /> </texture>
Transform node	<Transform>…</Transform> 2D transform not supported	<Transform>…</Transform> <Transform2D>…</Transform2D>
Geometry definition node	There is no specific node for geometry definition in X3D	<geometry> … </geometry>

Geometry nodes	<Sphere />	<Sphere />
	<Text />	<Text />
	<Box />	<Box />
	<Cone />	<Cone />
	<Cylinder />	<Cylinder />
	<PointSet />	<PointSet />
	<IndexedLineSet />	<IndexedLineSet />
	<LineSet />	<LineSet /> ??
	<IndexedFaceSet />	<IndexedFaceSet />
	<ElevationGrid />	<ElevationGrid />
	<Extrusion />	<Extrusion />
	<Arc2D />	Arcs not supported
	<ArcClose2D />	Closed arcs not supported
	<Circle2D />	<Circle />
	<Disk2D />	Disks not supported
	<Polyline2D />	Polylines not supported
	<PolyPoint2D />	PolyPoints not supported
	<Rectangle2D />	<Rectangle />
	<TriangleSet2D />	2D triangle sets not supported
	<TriangleSet />	Triangle sets not supported
	<TriangleFanSet />	Triangle fan sets not supported
	<TriangleStripSet />	Triangle strip sets not supported
	<QuadSet />	Quad sets not supported
	<IndexedTriangleSet />	Indxd triangle sets not supported
	<IndexedTriangleStripSet />	Indxd triangle strip sets not supported
	<IndexedQuadSet />	Indexed quad sets not supported
Sensor nodes	<TimeSensor />	<TimeSensor />
	<TouchSensor />	<TouchSensor />
	<PlaneSensor />	<PlaneSensor />
	<CylinderSensor />	<CylinderSensor />

	<SphereSensor /> <KeySensor /> <StringSensor /> <LoadSensor /> <ProximitySensor /> 2D proximity sensors not supported <VisibilitySensor /> Disc sensors not supported	<SphereSensor /> Key sensors not supported String sensors not supported Load sensors not supported <ProximitySensor /> <ProximitySensor2D /> <VisibilitySensor /> <DiscSensor />
Interpolator nodes	<ScalarInterpolator /> <ColorInterpolator /> <PositionInterpolator /> <OrientationInterpolator /> <NormalInterpolator /> <CoordinateInterpolator /> <PositionInterpolator2D /> <CoordinateInterpolator2D/>	
Script node	<Script>…</Script>	
Routing node	<ROUTE />	
Lighting nodes	<DirectionalLight /> <PointLight /> <SpotLight />	
Triggering nodes	<BooleanTrigger /> <IntegerTrigger /> <TimeTrigger />	No triggering node are supported in XMT-A
Scene environment nodes	<Background /> <TextureBackground /> <Fog />	<Background /> <Background2D /> <Fog />
Audio nodes	<Sound /> <AudioClip />	

Prototyping node	<PROTO>	No prototyping supported

Transforming XMT to X3D

When the client receives the XMT file from the requested WebM file, it starts the transformation process. This process relies on the eXtensible Stylesheet Language Transformations (XSLT) language, an XML-based framework for converting XML documents from one schema to the other. The next subsection presents the XSLT language in short. Our choice of XSLT as the transformation framework is based on the important observation that both standards we are interested in are XML-based. Even if they did not demonstrate the huge similarities we know they have, to the extent that an XMT document could uniquely lead to an X3D document, XSLT would be our framework of choice for the task. In our implementation, XSLT "translates" the XMT elements into their corresponding X3D elements. Since it is possible that the XMT-A document will contain elements that have no corresponding X3D nodes, we operate under the concession that any elements that are not supported by the X3D language and don't have corresponding elements are discarded.

Table 22 Example XSLT code for transforming XMT to X3D

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" omit-xml-declaration="yes"/>

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

    <!-- Remove <scene> elements -->
    <xsl:template match="scene">
        <xsl:apply-templates select="node()"/>
    </xsl:template>

    <!-- Remove <coord> elements -->
    <xsl:template match="coord">
```

```
                    <xsl:apply-templates select="node()"/>
                </xsl:template>

                <!--More xslt code… -->

                <xsl:template match="/">
                    <xsl:copy>
                        <xsl:apply-templates select="XMT-A/Body/Replace"/>
                    </xsl:copy>
                </xsl:template>
</xsl:stylesheet>
```

The XSLT language

As mentioned above, the XMT file needs to be converted into X3D in order to be passed to X3DOM. This process is done using XSL Transformations (XSLT). XSLT is a mechanism that accompanies the XSL (the XML stylesheet language) and allows conversions of XML files into other XML documents [25]. It defines special XML nodes (as seen in Table 22) that select, copy, modify, remove or style elements inside of XML documents.

As stated in [25], "The transformation is achieved by associating patterns with templates". In other words, XSLT nodes search in the source XML document for very specific parts of it (patterns of nodes) and then apply to the matched nodes the defined transformations and styles (the template). The template creates a new node tree that is not related in any way with the tree of the matched nodes in the source XML file. In fact, the resulting node tree, as stated in [25], "can be completely different" from the originating tree because of the modifications and transformations that can add, remove, alter, and reorder the original nodes. Finally, XSLT patterns, for selecting nodes from the source XML file, use the XPath expression language.

Analysis of the XSLT transform

At first, the XSLT parses every element node that the XMT file contains, along with every attribute that every element could have.

Table 23 Parsing every XMT node, along with every attribute

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

XMT file contains a number of nodes that are specific to the XMT-A standard (like <children>, <material> and many more). Along with these nodes, in the XMT file the corresponding X3D nodes are present as well (like <Material> etc.). So we end with a file that has duplicate nodes which also have the wrong case (X3D is case sensitive). In order to correct this problem, the XSLT has to remove any unwanted nodes from the XMT file.

Table 24 Removing unwanted and invalid nodes from the XMT file

```
<xsl:template match="children">
        <xsl:apply-templates select="node()"/>
</xsl:template>

<xsl:template match="material">
        <xsl:apply-templates select="node()"/>
</xsl:template>

<!-- This continues until all unwanted nodes are removed  -->
```

From this point on, the system needs only to parse the nodes that contain the transformed X3D scene and pass it on for further process.

Table 25 Parsing the transformed X3D scene

```
<xsl:template match="/">
        <xsl:copy>
            <xsl:apply-templates select="XMT-A/Body/Replace"/>
        </xsl:copy>
</xsl:template>
```

The following table shows the exact alterations that the XSLT file applies to the original XMT file, in order to produce valid X3D content.

Table 26 Alterations made by XSLT to the XMT file

XMT node	X3D
<XMT-A>	Ignored
<Header>	Ignored
<Body>	Ignored
<Replace>	Ignored by JavaScript
<Scene>	<Scene>
<par>	Used to define starting time and duration, then removed by JavaScript
<Group>	<Group>
<children>	Removed
<appearance>	Removed
<material>	Removed
<texture>	Removed
<geometry>	Removed
<fontStyle>	Removed
<coord>	Removed
<source>	Removed
<color>	Removed
<normal>	Removed
All the other nodes are passed as is, without making any kind of alterations on them.	

3.6 – Additional XMT information

The XMT document contains the scene information, but also contains a wealth of metadata that, for the most part, do not correspond to X3D nodes, and cannot be incorporated in

X3D documents. We should keep in mind that XMT is a lot more than a schema for describing declarative 3D scenes. Instead, as the XML-based successor to BIFS, it contains entire Multimedia Objects, possibly consisting of multiple interrelated Streams, which can include anything from audio and video to rights and content descriptions.

The framework we are proposing aims to take into account the entire content of the XMT document, and not only the 3D scene content. One very important such piece of information is the timing information concerning the time of appearance of the 3D scene in the video, with respect to the running time of the video itself, and the length of time it is to remain visible (i.e. the "duration" of the 3D scene, after which it will disappear again). These two values per scene (start time and duration) are defined in the attributes of the <par> element under the *begin* and *duration* attributes, and are extracted from the document and stored in JavaScript variables to be used when appropriate. However, we can similarly extract other types of information from the scene. For example, a player can seek and isolate all rights and DRM-related information on the content, and behave correspondingly. Or, an MPEG-7 description could be embedded in the XMT scene, and it could be isolated for a retrieval task.

Overall, the adaptation of the XMT-A content for display within HTML5 is a process consisting of multiple steps, with the end result of an X3D scene ready to be processed by a web browser using X3DOM, and a number of variables containing further information from the XMT-A multimedia object.

3.7 – Video playback and 3D content overlay

Following the transformation of the XMT-A scene into X3D, and having extracted from the XMT-A scene the additional parameters – most importantly the start time and duration of the 3D scene, but also potentially rights information, for example specifying whether the player should be allowed to play the material, or how to decode it -, the page is ready to play the integrated Video-VR scene. The playback framework consists of a standard HTML5 video element with a JavaScript event listener attached to it, alongside an X3DOM scene embedded in the HTML code, which is initialized as empty of all content (i.e. containing only the <X3D> and <Scene> elements with no children). With respect to the implementation, we treat this X3DOM scene as essentially being in standby mode and awaiting for the DOM to be updated by adding child elements, which are to be displayed.

The first step is to load the video stream from the WebM file into the Video element. This can simply be achieved by providing the URL of the WebM file containing the video and the XMT attachment. While the specification of WebM does not take the possibility of an attachment into account, its innate relation to the Matroska standard means that the element can be placed there, and will not interfere with the video data during playback, in any player, HTML5 or otherwise. Instead, it will be bypassed and the video will play normally.

As the video begins to play, the JavaScript listener constantly examines the elapsed time of the video playback and compares it to the *begin* attribute of the <par> element of the XMT-A scene. As soon as the listener returns that the video playback time matches the start time contained in a scene in the XMT-A file, the corresponding 3D scene is loaded, and overlaid on the video, allowing interaction and navigation functionalities, and partially covering the video content. The user can at this stage experience the video with the addition of the 3D material, although it is also quite possible to pause the video for a duration of time, according to the needs of the application. The extent of navigation and interaction options allowed to the user are, as always in the context of X3D, defined by the scene author and embedded in the XMT-A code.

With respect to the duration for which the scene appears over the video, in our current implementation the listener continues to count the playback time and to compare it with the duration defined in the XMT structure. When the expected duration passes, the content is removed from the scene by removing it from the DOM, leaving the X3DOM scene empty and in standby again. However, this setup is not necessarily the only option. Alternatively, we could be pausing the video and counting time using an independent clock, and removing the 3D content and resuming the video after the specified duration passes. Although the two types of content (3D and video) appear interlocked, they are essentially independent, which allows us any number of implementation choices with respect to their timing and synchronization.

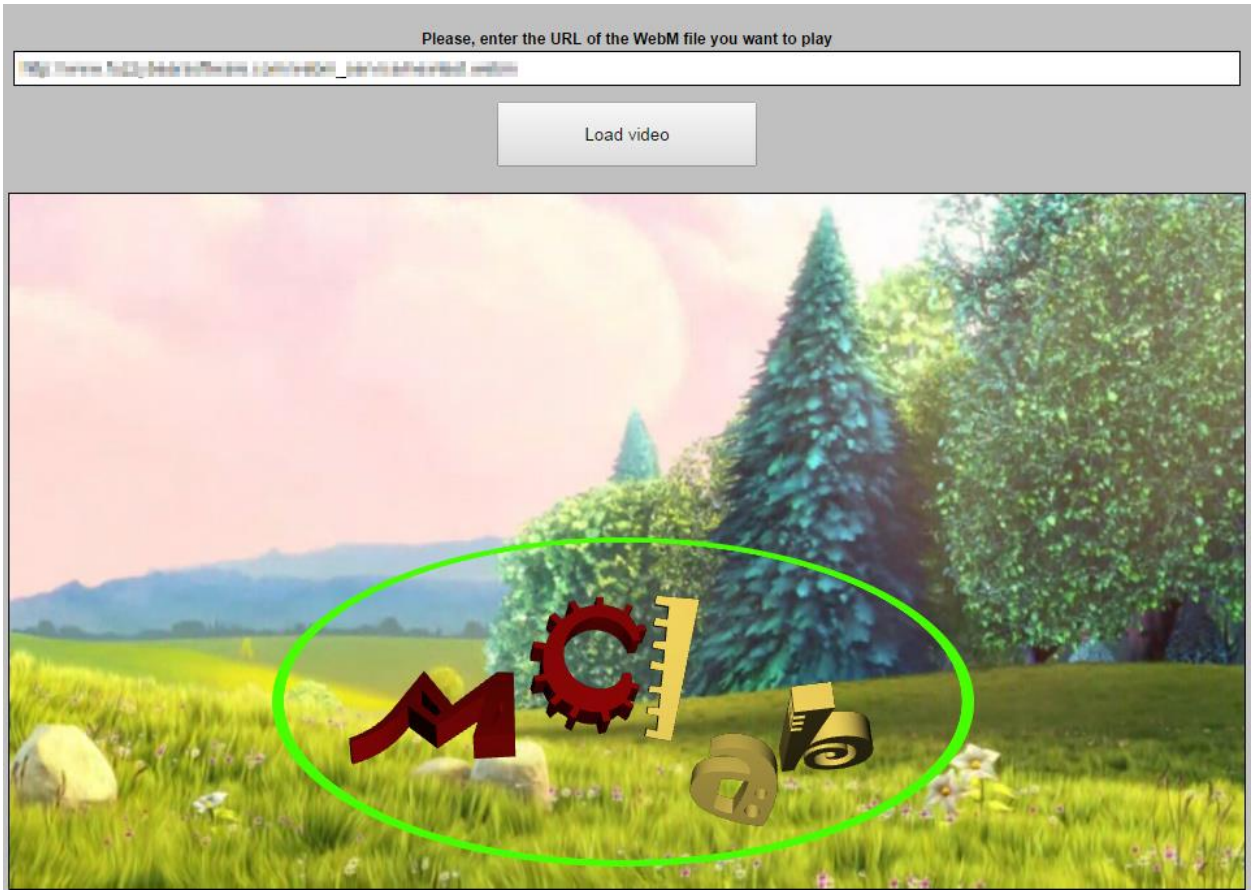

Figure 3-4 3D content being displayed on top of the video stream

Furthermore, up to this point we have only discussed examples in which a single scene is contained in the XMT-A document. However, the internal structure of XMT-A is such that it can allow multiple different scenes at different time intervals to be contained in the same file, in a simple, explicit, sequential manner. The framework developed in our work can easily be extended to handle such a case, by inserting and removing the corresponding scenes throughout the duration of the video contained in the WebM file. This means that multiple, radically different scenes with different material could be contained, from limited duration advertisements at different time points, to 3D scenes that are part of the user experience, designed to be played over the video, and changing in time as the video shots or scenes change.

Conclusions, future steps

In this thesis we created a framework that gives the ability to developers to integrate 2D or 3D MPEG-4 XMT interactive content into the WebM video format, using only HTML-5 and JavaScript technologies. We explored the relationship between XMT-A and X3D, created a mechanism to automatically convert content between them, a system that displays XMT-A scenes in HTML5 and a transfer and display mechanism of XMT-A scene through and alongside WebM video files.

Although MPEG-4 3D scene description technologies have been pushed aside (by other technologies) in the recent years, the system we created put it again into the spotlight, because it offers flexibility, wide applicability, ease of authoring and interpretation, the declarative aspects of X3D as well as the standardization, ubiquity and prestige that MPEG-4 offers. The reason for this is that MPEG-4 3D scenes are ultimately made web-compatible, a feature that makes it very attractive for authors and developers. Anyone can use the power and flexibility of XMT to create interactive 3D content that can be viewed by the millions of users across the web. Additionally, we created the 3D player that enables web developers to use this technology to incorporate WebM videos that display 3D interactive content into their web pages.

Furthermore, WebM is a new but very promising multimedia container format that is designed specifically for fulfilling the needs of the web for a totally free and open source multimedia file format. The open source nature of WebM renders developers able to customize every aspect of it in order to improve it or bring it to their needs. Moreover, because it is designed for the web, it requires very small amounts of computational power. This enables WebM to be played back in a vast range of devices.

For this reasons, we brought together this predominantly web-compatible video format with the interactive MPEG-4 3D scenes. This means that not only a 3D scene can display video streams inside it, but a video stream can contain and display 3D content as well. And because the XMT-A content is conducted by temporal rules, the 3D scenes can be displayed, hidden or modified based on time, something that gives very powerful control, to developers and authors, on how they manage their content.

This technological fusion opens the door to a vast field of applications. Such video streams(that have intercalary 3D scenes) can be used, for example, in the field of advertising and

promotion. Such a system, that leverages the benefits of XMT-A is iPromotion [29]. The described platform is used to deliver rich and interactive advertisements through the web. iPromotion uses the XMT-A file structure in order to define Multimedia Objects that consist of X3D scenes, along with MPEG-7 descriptions, DRM information etc. The system extracts every data requested on the server from the XMT-A file and sends to the client pure X3D content. iPromotion could be extended to support our proposed framework, so that it uses XMT-A notation to describe 3D scenes (instead of X3D inside of XMT-A). Additionally, an advertisement could be a video (that catches the eye of the user more easily) that features some interactive 3D scene on top of it.

Another example of applications could be systems that offer the ability to playback videos can use this technology to display 3D interactive ads that are overlaid on the video. Streaming video providers (such as YouTube and Vimeo) or even Video on Demand and Internet Television provides (such as Hulu or NetFlix) can take advantage of such technology. Movies that can interact with user by displaying 3D content that can (possibly) change the script flow, displaying eye catching advertisements before or during video playback are some of the potential uses.

Taking such an application one step further, it could be combined with a system like MiSPOT that is proposed in [26]. MiSPOT uses XMT-A to describe and create advertisements, called hooks (although, when the hooks are compiled for transmission, BIFS is used). Then, the system retrieves a list with the topics of advertisement that each specific user is more receptive and chooses the right hooks to display. This would give the ability to our framework to deliver specialized and targeted interactive 3D advertisements to the viewers, fused with the environment that is being displayed.

Another great example of usage is merging educational games or applications with this interactive 3D technology. Such applications may rely on narrative videos (for example a tutor explains the structure of an atom)that are able display in interactive 3D the subjects that they present. This way, users are more prone to be engaged (without getting bored) and understand better the subject of presentation. Furthermore, users can be introduced with interactive 3D VR quizzes in order to evaluate their level of comprehension and to practice.

Concluding, this system can be further improved by supporting more nodes and extra functionality, even those that are not supported by X3D at all. Thanks to the PROTO node

supported by X3D it is relatively simple to create new nodes that will function as the counterpart of some of the XMT-A nodes that X3D doesn't support. Another important addition to the framework is Scripting. Both X3D and XMT support the Script node, so it is very easy to extract code from there and execute it on the client. Such a feature will boast the power of the framework and will make it much more flexible and interactive. This kind of functionality, though, may raise the need to tweak or modify X3DOM itself, in order to support the extra nodes.

Furthermore, a tool that will allow the complete and correct conversion from X3D to XMT-A can be created. This encourages authors and developers to use MPEG-4 XMT content in HTML5 web pages because they can convert existing X3D scenes to XMT, something that will give them the ability to leverage its power. All this future work can consolidate the acceptance of MPEG-4 XMT as a major standard for displaying video and interactive 3D scenes on the web.

Overall, while in this work we have explored the background, brought together all related technologies, and laid the groundwork for the fusion of two major industry standards (MPEG-4 and X3D) with a dominant video format (WebM), the possibilities of future extensions and applications remain promising, and many directions exist for exploration from this point forward.

References

[1] EBML Official Website, EBML, http://ebml.sourceforge.net

[2] Matroska Official Website, What is Matroska?, http://www.matroska.org/technical/whatis/index.html

[3] Matroska Official Website, Diagram, http://www.matroska.org/technical/diagram/index.html

[4] WebM Official Website, About WebM, http://www.webmproject.org/about

[5] WebM Official Website, Frequently Asked Questions, http://www.webmproject.org/faq

[6] WebM Official Website, WebM Container Guidelines, http://www.webmproject.org/code/specs/container

[7] WikiPedia, WebM, http://en.wikipedia.org/wiki/WebM

[8] Michelle Kim, Steve Wood, Lai-Tee Cheok (2000). Extensible MPEG-4 Textual Format (XMT)

[9] MKVToolNix – Cross-platform tools for Matroska, http://www.bunkus.org/videotools/mkvtoolnix/

[10] GPAC, Multimedia Open Source Project, http://gpac.wp.mines-telecom.fr

[11] Don Brutzman, Leonard Daly (2007), "X3D: Extensible 3D Graphics for Web Authors", Elsevier

[12] X3DOM – About x3dom, http://www.x3dom.org/?page_id=2

[13] ISO/IEC, "Information technology – Coding of audio-visual objects – Part 11: Scene description and application engine"

[14] getID3() – The PHP media file parser, http://getid3.sourceforge.net

[15] Behr, Eschler, Jung, Zollner (2009), "X3DOM – A DOM-based HTML5/X3D Integration Model"

[16] Web3D, "Extensible 3D (X3D) – Part 2: Scene Access Interface (SAI), http://www.web3d.org/documents/specifications/19775-2/V3.3/index.html

[17] Fernando Pereira, Touradj Ebrahimi (2002), The MPEG-4 Book, Prentice Hall Professional

[18] Young-Kwon Lim, Cyril Concolato, Jean Le Feuvre, Kyuheon Kim (2012), "MPEG Multimedia Scene Representation" in "The MPEG Representation of Digital Media" (pp. 177 – 202)

[19] Wojciech Cellary, Krzysztof Walczak, "Interactive 3D Content Standards"

[20] The Moving Picture Experts Group, "Overview of the MPEG-4 Standard", http://mpeg.chiariglione.org/sites/default/files/files/standards/docs/w4668.zip. Accessed 01 September 2014

[21] Aaron E. Walsh, Mikael Bourges – Sevenier, "The MPEG-4 Jump-Start", Prentice Hall Professional Technical Reference

[22] C. Concolato, J.-C. Dufourd, "Comparison of MPEG-4 BIFS and Some Other Multimedia Description Languages", Ecole Nationale Superieure des Telecommunications, ENST

[23] Mojtaba Hosseini, Nicolas Georganas, "MPEG-4 BIFS Streaming of Large Virtual Environments and their Animation on the Web", Web3D '02 Proceedings of the 7th International Conference on 3D Web Technology

[24] Web3D, "What is X3D and Supported Features", http://www.web3d.org/realtime-3d/x3d/what-x3d/, Accessed 01 Septemeber 2014

[25] James Clark (1999), XSL Transformations (XSLT) Version 1.0, http://www.w3.org/TR/xslt, Accessed 01 September 2014

[26] Martin Lopez-Nores, Yolanda Blanco-Fernandez, Alberto Gil-Solla, Manuel Ramos-Cabrer, Jose J. Pazos-Arias (2011), "Exploiting MPEG-4 Capabilities for Personalized Advertising in Digital TV", "The Handbook of MPEG Applications" (p. 103 – 123), Wiley

[27] Jean Le Feuvre, Cyril Concolato, Jean-Claude Dufourd, Romain Bouqueau, Jean-Claude Moissinac (2011), "Experimenting with Multimedia Advances using GPAC"

[28] GPAC (2012), "GPAC, Toolbox for Interactive Multimedia Packaging, Delivery and Playback", ACM SIGMM Records Vol. 4, No. 4, Open Source Column

[29] Markos Zampoglou, Athanasios G. Malamos, Kostas Kapetanakis, Konstantinos Kontakis, Emmanuel Sardis, George Vafiadis, Vrettos Moulos, Anastasios Doulamis (2014), "iPromotion: A Cloud-Based Platform for Virtual Reality Internet Advertising", In "Big Data and Internet of Things: A Roadmap for Smart Environments", Studies in Computational Intelligence, 546,(pp. 447-470). Springer International Publishing.

[30] Hee-Sun Kim, Ilgun Ko (2011), "A Generation Method of MPEG-4 3D Scene for Mobile Environments", "International Journal of Multimedia and Ubiquitous Engineering" Vol. 6, No. 2

[31] Blagica Jovanova, Ivica Arsov, Denis Conan, Doan-Tuan Anh, Alain Ozanne, Marius Preda (2011), "Mobile Mixed Reality Games Creator Based on MPEG-4 BIFS"

[32] Koong, Chorng-Shiuh, Lee, Bo-Huan, Wang, Yu-Hsiang, Chang, Chih-Hung, William C. Chu (2012), "A Component-based Authoring Tool and Script Language for MPEG-4", 12th International Conference on Quality Software

[33] Alexander Uherek, Sonja Maier, Uwe M. Borghoff (2014), "An Approach for Long-Term Preservation of Digital Videos based on the Extensible MPEG-4 Textual Format"

[34] Qonita M. Shahab (2005), "A Study on Transcoding of MPEG-4 BIFS Scenes to MPEG-4 LASeR Scenes in MPEG-21 Digital Item Adaptation Framework, School of Engineering, Information and Communications University

[35] Bruno Carpentieri, Roberto Iannone (2012), "Building an Impress Extension for Interactive MPEG-4 Video Conversion"

[36] Kim, J., Lee, C. G., Kim, Y., & Ryu, J. (2013). Construction of a haptic-enabled broadcasting system based on the MPEG-V standard. *Signal Processing: Image Communication*, *28*(2), 151-161.

[37] Mathieu Robart, Mauro Plumari, Danilo Pau (2013), "Extended MPEG-4 3DGC for Real-time Rendering on Embedded Platforms"

[38] Sasko Celakovski, Danco Davcev (2010), "Multiplatform Real-Time Rendering of MPEG-4 3D Scenes with Microsoft XNA"

[39] Launchpad6, "Online Video Marketers Guide: What's the Difference Between In-Stream and In-Banner Video Ads?", http://launchpad6.com/en/online-video-marketers-guide-whats-the-difference-between-in-stream-and-in-banner-video-ads/, Accessed 15 December 2014

[40] Houston Chronicle, "What is In-Stream Advertising?", http://smallbusiness.chron.com/instream-advertising-36316.html, Accessed 15 December 2014