The interactive media framework XIMPEL

an implementation in HTML5/JavaScript

Stefan Bruins

A thesis presented for the degree of
Master of Science

Computer Science
Vrije Universiteit
Netherlands

February 16, 2016

1. First Reader: Anton Eliens

2. Daily Supervisor: Anton Eliens, Winoe Bhikharie

3. Second Reader: Victor de Boer

Abstract

This project focuses on the interactive media framework called 'XIMPEL'. This interactive media framework allows for easy creation of interactive media presentations. The framework was originally implemented using flash technology. In this project we created an implementation of the XIMPEL framework using the HTML5/Javascript technology. We take a practical approach by creating a proof of concept to proof HTML5 technology has matured enough since its initial release to support large media applications running inside of a web browser. As part of the project we determined the functional and non-functional requirements that XIMPEL must support. Additionally we have looked at what interactive media frameworks are and how they can be effective in educational settings in the form of a literature study. We have provided justifications for the requirement specifications, the architecture of the application and the implementation details.

Contents

1. Introduction

1.1 Introduction

XIMPEL is an interactive media framework and is currently implemented using flash technology. The framework allows authors to describe a media presentation using XML syntax. The media presentation can include different types of media (for instance YouTube videos, audio fragments or images). Interactive elements such as click-able overlays or questions can be added to the presentation to allow the user to interact and to influence the order of the media items that make up the presentation. By allowing the user to interact with the content of the presentation the users are more involved than with regular video. The main focus of this project is to create an implementation of the XIMPEL framework in HTML5 / Javascript. We explore the possibilities and limitations of using HTML5 technology. HTML5 is the most recent HTML specification and has been around for several years. Support for HTML5 has been growing but may still not be consistent enough across different browsers.

1.2 Overview

This project is an exploratory research focused on determining whether the HTML5 / Javascript technologies have matured enough to support a large media based framework like XIMPEL. Additionally we explore the relevance of XIMPEL as an educational framework and which future extensions may be relevant.

In this chapter (chapter 1) we first describe the main research problem, the research questions and our research aproach. In chapter 2 we provide some context and background information on Interactive Media and the relevant technologies involved. In chapter 3 we desribe the planning of our framework and the design decisions that were made. In chapter 4 we discuss the architecture and technical details of our implementation. In chapter 5 we discuss a presentation that has been created with the new implementation of XIMPEL that forms our proof of concept. In chapter 6 we describe relevant features that have not yet been included in this version of the new implementation but may be interesting and relevant to add in the future. We end with the conclusion in chapter 7.

1.3 Research problem

The current implementation of the XIMPEL framework is written in Actionscript / Flex. In order to run applications written in Actionscript a flash player is required. In most cases the flash player must be installed in the form of a plugin and is not natively supported by the browser. Additionally, these plugins are mainly available for the Microsoft Windows platform. Especially mobile platforms often lack support for flash applications. The use of the flex/as3 tool (flash) proved to be a significant burden for (beginning) developers. By using the HTML5 and ECMAScript 5 standard (Javascript) as a basis for the XIMPEL framework, the framework will be natively supported by all browsers that implement that standard. Additionally, since no special tools or compilers are required to work with HTML5/Javascript it will be easier to use for the beginning programmer.

The current version of XIMPEL supports certain features. There are ideas for new features which may require significant changes to the existing framework. Example features are parallel media playback, the use of mini-games within a presentation and google maps as a media type that can also be included in your presentation. Since the re-implementation of XIMPEL in HTML5/Javascript will be developed from scratch it is useful to inventorize what features will be supported right away and which features might be implemented later on and take this into account when building the XIMPEL framework.

1.4 Research question

In this research we will look at the usefulness of the XIMPEL framework and its re-implementation in HTML5/ECMAScript 5. Additionally we will look at how the framework should be implemented and what features it should include. We formulated two following research question:

Question: Has HTML5/Javascript technology matured enough to support the XIMPEL educational interactive media framework?

The following elements are in-scope for the research question:

- What are interactive media frameworks, why are they useful and which ones already exist?
- What javascript libraries are available to support the implementation of the XIMPEL framework and what are the pros and cons for using them?
- How can ECMAScript 5 as a language be used to build a structured framework?
- How can extensibility of the framework best be achieved?

1.5 Research approach

A practical approach is used in this project. By developing an implementation of XIMPEL in HTML5/Javascript technology we try to prove that HTML5 has become powerful and mature enough to support a big media framework. Since the framework already exists in the form of flash technology we can compare whether the functionality of the flash version also works in the HTML5 version.

Additionally we will do a literature study on interactive media (frameworks) and related topics in general. Then based on the findings and based on the views of the original author of the XIMPEL framework we create a requirements specification that describes what will be included in XIMPEL. Based on these requirements and based on best practices and programming patterns we describe the design choices for the architecture and implementation details of our XIMPEL framework. The justification for our choices is based on our literature study, best practices, patterns and personal experiences.

2. Background

2.1 Interactive video

2.1.1 What is interactive video/media?

The term interactive video (or interactive media) is an ambiguous term. In [1] several different interpretations of interactive video are mentioned. In one interpretation interactive video is viewed as a way of allowing the user to interact with elements within the video that link to external media sources that are not part of the video itself. This can be used to enrich what is conveyed within the video with additional external information sources. For instance, clicking at certain parts of the video will open a website in a browser with related information.

Another interpretation focuses on the user interaction with the video to control the playback of that video. This could be simple enhancements beyond the basic playback controls such as jumping to different predefined chapters. It may also be more advanced playback control. For instance, being able to search the next occurrence of a certain actor within the video. Such information is usually implicit in the sense that there is no structure that holds information about the different actors that play a role in the video. To support this type of advanced playback control an explicit structure of the information that is conveyed within the video is required. This structure can either be computer generated by complex algorithms or the structure can be manually created.

In [2] the focus is on non-linear interactive video. Several definitions are given, all referring to video where the user does not only control the playback within one fixed video sequence, but it can actively influence the sequence of videos (or other types of media) that will be played. The interactive video consists of multiple videos which are played in a dynamic order based on the input from the user at run-time. This can, for instance, be used to branch to a specific story line based on the user's input or to repeat a cycle of videos until the user entered the right answer to a quiz question.

Additionally, [3] identified four common terms from the literature on interactive video: "hypervideo", "interactive video", "annotated videos" and "non-linear video". The distinction between those terms is subtle and as such they are often used interchangeably. Figure 2.1 shows the elements that are mentioned in the literature for each of these terms. Considering figure 2.1, the term "hypervideo" seems to best reflect the functionality of XIMPEL. However the item "modification

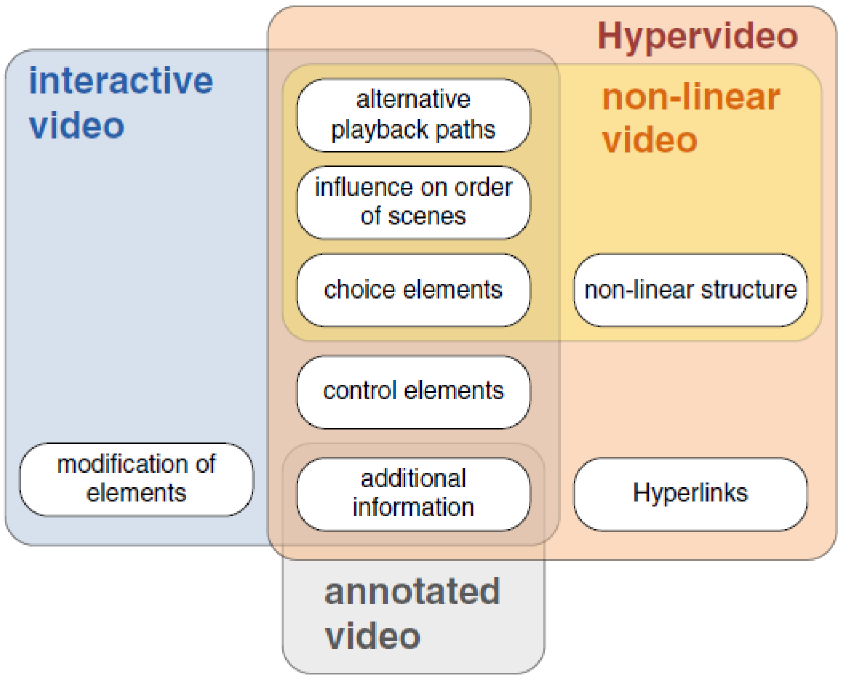

Figure 2.1: Overview of different terms used in relation to "interactive video" [3].

of elements", which means "adding to or changing the content of the video or its annotations", is not something that is ruled out for future versions of XIMPEL. Consequently, whenever we talk about interactive video we are referring to all the elements in figure 2.1. The XIMPEL framework does not only support video but can support other types of media as well. In the literature this is sometimes referred to as "interactive media presentations".

To summarize, we will consider an interactive video (or media) presentation to be a presentation which may have (non-linear) alternative playback paths where the user can influence the order in which media items are shown by using hyperlinks, control buttons or choice elements and may possibly link to external content. Whenever we use the term "interactive video", we are talking about interactive media presentations.

An example scenario for interactive media Suppose we are scientists and discovered that if we do not fundamentally change the way humans interact with the planet, the planet may become inhabitable within several centuries. Having become slightly worried because of this prospect we decide to shoot a video to raise awareness for the situation. Unfortunately, the video did not have the intended impact on the awareness of the people. We decide maybe we need to involve the viewers more, so we create an interactive media presentation. In this interactive media presentation people first get to see some videos, audio fragments and images that look at environmental issues in general and how it may affect earth. After these videos, questions will be asked to the viewer about their personal habits that influence the environment. For some questions, when the user chooses an answer that is harmful for the environment he or she is directly confronted with his or her bad choice. For other questions, the user is not confronted instantly but one or more scores are tracked in the background. At the end of the presentation the user gets feedback about the effects of his or her choices based on the scores for different variables. All of this all happens within the interactive media presentation.

2.1.2 Relevance?

There are many possible applications for interactive video. For entertainment purposes, educational purposes and convenience purposes. Examples of applications for interactive media presentations are:

- Math lessons with instructional videos and quizzes that must be passed in-order to continue to more advanced concepts.
- Providing a virtual tour through a building where the user can decide which route to take.
- A presentation aimed at creating environmental awareness through instructional videos and giving feedback on the effects of the viewer's personal habits.
- Showing a different story line based on the user's choice.
- Providing the ability to select different videos of the same scene taken from different angles.
- An image of the world map where clicking a country will show the crime statistics for that country, including links to external data sources

An important contribution of interactive media is in the field of education. In [4] the effectiveness of e-learning is measured when using interactive video as opposed to using normal video or no video at all. This research shows that regular video does not always have a positive effect in e-learning environments. It may lead to a better learning outcome but it depends on the way video is used. The research also shows that the satisfaction among the users of interactive instructional videos is higher than the satisfaction among the users of regular videos or no videos. Additionally the research claims that the usage of interactive videos can increase test scores. This finding is in line with previous research [5].

Another research [6] mentions that the differences in cognitive characteristics between people causes people to learn in different ways. In order to create a better learning experience, the information should be shaped, arranged and optimized for a specific viewer rather than for a general public. Examples of differences between individual viewers are cognitive skills, prior knowledge, interests and learning strategies. While these elements are difficult or impossible to address with traditional media such as video, audio and images, it is to some extent, possible to address them using interactive media presentations. Such interactive media presentations can enable users to adapt the presentation to their individual needs by giving them control over the "what" and the "how" of the presentation. The research also mentions that viewers of traditional video typically face the problem that they must rapidly organize information that is presented to them at a rate that they cannot change. The lack of adaptability of a presentation may lead to shallow processing of information and cognitive overload. Adaptability of a presentation can be improved by using interactivity.

In [7] research is done on interactivity in multi-modal environments where the information is transferred both verbally and visually. They mention that the following elements can improve learning performance:

- Guided-activity
- Reflection
- Feedback
- Pacing
- Pretraining

By guiding the learning process and reflecting upon correct answers the learner can better organize and integrate new information. Feedback reduces extraneous processing by providing students with better schemas to repair misconceptions in their mental models. Controlling pacing can benefit the student because it allows the students to process smaller chunks of information in working memory and pre-training can help students to relate new information to prior knowledge and existing mental models. Interactive media presentations can, to some extent, provide a guided learning experience, for instance by using quizzes to determine if a student understands the required information for the upcoming material and give appropriate feedback. It is also possible to allow the user to control the pace in which he or she steps through the learning material.

Interactive video seems to benefit the learning experience as well as improve learning satisfaction.

2.2 Interactive video frameworks

2.2.1 The need for a framework

An interactive video presentation can be represented in different ways and stored in different formats, interaction methods can vary from mouse clicks to drawing figures on a touch screen, supported media types can range from videos to images or google-maps instances, the presentation can be displayed in many different ways and there are endless possibilities for additional functionality that can be added to an interactive presentation. The different forms that a presentation can take is dependent on the functionality within the framework used to create and run it.

In principle, interactive presentations can be defined by code without using a framework. However, since different interactive presentations are likely to have a large amount of overlap in code, it is sensible to create a framework to aid development of interactive presentations. The main reasons for using a framework to create interactive presentations is to:

- Hide the programming complexity from the author of the presentation.
- Increase the authoring speed of interactive presentations.
- Offer a toolbox of functionality that can be re-used in different presentations.
- Allow external applications to communicate with the presentations using a fixed public interface.

2.2.2 Descriptive formats

The programming complexity can be hidden by implementing an interpreter that interprets a descriptive format of the interactive presentation. The descriptive format describes how the presentation looks, behaves and reacts to user input. This presentation description can be any format, for instance a format based on XML or JSON. The presentation description includes all the required information for running the interactive presentation. This required information includes but is not limited to the source location of the used media items, the possible user interactions, the default playback order of media items and the visual properties of the elements within the presentation. By using a descriptive format to describe a media presentation, the author of the presentation can focus on describing what he or she wants the presentation to do and look like, rather than focussing on how this should be done in code. Based on a presentation description the interpreter knows how the run the presentation and deal with user interactions.

The usage of a descriptive format makes it easier to create presentations, but at the same time it imposes limitations. There is a balance between expressiveness of the format and simplicity. The interpreter may support a format that is as expressive as a fully fledged programming language but then there is no point of having a descriptive format. The other extreme is to have a very clean, elegant and simple format that allows to express only the most basic functionality. In that case the usability of the format is very high but the expressiveness is low. The exact balancing of simplicity and expressiveness is a choice made by the designer of the framework.

2.2.3 Players

Besides a descriptive format, a framework may offer a player which knows how to play a media presentation. The player handles the media playback and the user interaction as well as any other functionality that is supported, such as displaying quiz questions. Players may support one or more descriptive formats.

2.3 XIMPEL in a nutshell

Now that we have discussed what interactive video presentations and frameworks are, we can discuss the XIMPEL framework, the main topic of this research. XIMPEL is a framework for creating interactive media presentations. The main focus of XIMPEL has been to provide an open educational multimedia platform, where the main philosophy is to keep the authoring process of XIMPEL presentations simple and fast. Using XIMPEL, authors with a limited technical background can easily and rapidly create interactive multimedia presentations. The key focus points of XIMPEL are:

- Simplicity (easy to author presentations)
- Rapid development (little time needed to author presentations)
- Extensibility (user defined media types)
- E-learning (quizzes and scoring mechanism)
- Interactivity (Click-able elements to control the presentation)

XIMPEL uses its own XML based descriptive format to describe media presentations and it offers a player to run the presentation based on what is specified in the presentation description. The current implementation is built using Flex/Actionscript. Because the prospects for HTML5 related technologies seem better than that off Flash related technologies, a new implementation of XIMPEL will be developed in HTML5/Javascript.

We will discuss XIMPEL in more detail in chapter 3, where we will look at the current functionality in XIMPEL and what functionality the new implementation of XIMPEL will include.

2.4 A brief overview of the solution technologies

In recent years, much of the functionality that was previously only available on the web using Flash has now become available in browsers natively as part of the HTML5 specification or other W3C specifications. HTML, Javascript and CSS are supported out of the box by all major browsers where as Flash applications require a browser plug-in. Adobe has stopped supporting Flash for mobile platforms, while HTML5, CSS3 and Javascript are commonly supported even by mobile browsers. Additionally, the HTML5, CSS3 and Javascript specifications are actively being developed and have growing support on all the major browsers. For these reasons we have chosen to use HTML, CSS and Javascript as the solution technologies for the new implementation of XIMPEL.

HTML5 The language used to structure and present content on the web is the Hypertext Markup Language (HTML). HTML5 is the fifth major revision of the core HTML language which is a W3C standard. The HTML5 specification describes the elements and attributes that can be used to structure and present content on the web and how browsers should interpret these elements and attributes. The specification also includes a number of scripting API's that can be used with Javascript. Many scripting API's that people commonly consider to be "HTML5" are not actually part of the HTML5 specification, but are rather part of the Web Hypertext Application Technology Working Group (WHATWG) specification or specifications maintained separately by the W3C.

Javascript ECMAScript is a scripting language specification that describes the core language features such as the syntax and data types [8]. There are many ECMAScript implementations including Javascript, JScript and ActionScript. These implementations follow the language specification of the ECMAScript standard but they may also have additional features not part of the ECMAScript specification. Even though strictly speaking Internet Explorer does not support Javascript but Jscript, this distinction is rarely made and people generally refer to both using the term Javascript. Throughout this paper we use the term Javascript unless we are referring to a specific implementation.

At the time of writing, ECMAScript 5.1 is the most recent completed specification. A final draft for ECMAScript 6 has been submitted but has not yet been approved. The most recent releases of all important browser vendors have support for nearly all features in the ECMAScript 5 specification [9]. The browser support for ECMAScript 6 features is still poor [10]. It is generally not the case that a browser supports one specific version of ECMAScript entirely and then transfers to the next version in one go. Usually supporting a new ECMAScript version is an ongoing process. The term Javascript does not say anything about what version of ECMAScript it is based on, because the Javascript engine that is used to interpret the Javascript code may support part of the ECMAScript 5 features and part of the ECMAScript 6 features.

CSS Cascading Stylsheets (CSS) are used to define how the content within XML based documents is displayed. Historically, the structure and presentation of content within web pages were both defined through HTML. The aim of CSS is specifically to separate the content structure (HTML) from the content presentation (CSS). CSS styling rules are defined through embedded or external stylesheets and can be dynamically changed using scripting languages.

The DOM The Document Object Model (DOM) is a specification for representing objects in HTML and XML documents. The nodes (HTML elements) within these documents are organized in a tree structure. The DOM tree is the representation of these nodes. Web browsers use layout engines (also called rendering engines) to parse HTML into a DOM tree. Some layout engines, are used by multiple browser such as the Blink, WebKit and Gecko engines. The DOM has a formal W3C specification and exposes a public API which can be used by scripting languages such as Javascript to modify the DOM tree at runtime. Changes in the DOM tree are reflected to the user's browser window by the browser's layout engine.

Canvas The canvas is standardized by the Web Hypertext Application Technology Working Group (WHATWG). It allows dynamic scriptable rendering of 2d shapes and bitmap images. An earlier standard for drawing shapes in browsers is SVG, which is vector based. In vector based graphics the shapes are stored in a scene-graph in memory and subsequently rendered to a bitmap. The scene-graph is stored as part of the DOM where each object is a node in the DOM that can be accessed. As opposed to SVG, canvas consists of only one DOM node: the canvas HTML element. Canvas is raster based and does not keep track of what is drawn on the canvas. Instead, it directly updates a bitmap. As a consequence, when a shape is drawn on the canvas and its position were to change, then the entire scene would need to be redrawn. Furthermore, with SVG event handlers can be associated with objects, such that it becomes easy to listen for mouse-clicks on a specific object in the scene. To do the same thing in canvas, one must manually match the coordinates of the click event with the position of the object on the canvas. Canvas is a lower level API upon which a higher level engine can be built that may include scene-graph capabilities. There are libraries available that add scene-graph capabilities to the canvas (such as PaperJS, KineticJS, FabricJS and EaselJS) and it is also possible to paint a canvas in layers and then recreate specific layers.

Audio and Video HTML5 introduced the audio and video tags, offering the ability to easily include audio and video within a web page. The HTML5 specification does not specify which video formats should be supported, but the WHATWG initially recommended to support at least Theora video and Vorbis audio, as well as the OGG container format. However, this recommendation was later dropped and no concrete format has been chosen at this point. The main reason that there is no video format recommendation is due to patent and licensing issues and the lack of agreement between different players. As a consequence of this there is large polarisation of supported formats across different browsers and there is not one format that is supported by all the major browsers. In some cases browsers can support a particular format but only by using a plug-in. In order to still provide broad browser support for HTML5 video, HTML5 allows multiple source files to be specified. If the browser cannot play one of the source files it might be able to play one of the other source files that is stored in a different format. While this does provide better browser support, it is less convenient as the video must be stored multiple times and in different formats.

The video element can be manipulated in two ways. The first is by making use of CSS capabilities, such as transitions and transformations. The second way is to hide the video element and sample the video data to the canvas using the drawImage function of the canvas API. Before drawing the video data to the canvas, it can be manipulated in order to add different kinds of effects.

2.5 Javascript as a language

2.5.1 The global object and preventing namespace pollution

The global object in Javascript is the object to which global variables or functions are attached. A variable or function is global if it is not defined within a function or object. Within browsers the global object is usually called the "window" object, but in other environments it may be called something else. When building a framework it is common to expose as little as possible to this global object in order to avoid polluting the global namespace with all the variable and function names used within the framework. This reduces the risks that the framework has naming conflicts with other frameworks.

The "Immediately Invoked Function Expression" (IIFE) pattern in javascript can be used to prevent namespace pollution. IIFE uses the fact that javascript has first class functions. Rather than having all the code sit in the global scope it is wrapped within a function expression. The function expression is then immediately invoked which creates a new execution context with its own variable scope. All the code wrapped inside that function, ie. the framework code, is than ran. Any variable and function definitions are then attached to that execution context rather than the global namespace.

2.5.2 Inheritance

Javascript's inheritance model is different from that of languages such as Java or C++. In Javascript there are no classes, there are only objects. Everything that is not a primitive type is an object in Javascript, even functions. Objects are simply name value pairs. Even though there is no class concept as there is in many other OOP languages, Javascript still supports some form of inheritance where objects inherit from other objects. This inheritance is provided through the prototype chain. Each object points to another object as its prototype. When creating an object it is possible to specify which object should be the prototype for that object. The specified prototype-object itself may also have a prototype object. This chain of objects tied together through a hidden prototype property, form what is called: "the prototype chain". The prototype chain is used to implement prototypical inheritance. Whenever a property is referenced it is searched for on the object on which the property was referenced. If the property is not found there, it is searched for on the prototype of that object. If the property is also not found on the prototype-object then Javascript will look at that object's prototype. All the way up the prototype chain, until an object is encountered that has no prototype-object. An object has access to all the properties up its prototype chain.

This concept is often used to create multiple objects of the same type without having the code of the methods in memory for every created object.

2.6 Existing media frameworks and specifications

A lot of research has been done in the area of interactive media presentations.

2.6.1 SMIL

The most notable work is the Synchronized Multimedia Integration Language (SMIL). SMIL is a specification for an XML based markup language used for describing rich media presentations. In section 2.2 we talked about descriptive formats that may be used to hide programming complexity. SMIL is such a descriptive format, which allows users to describe a presentation through an XML document. It is a W3C recommendation for describing multimedia presentations. The SMIL specification is not a framework by itself, but merely a descriptive format which anyone can write an interpreter/player for. A few players exist that support the SMIL format, including: RealPlayer, AmbulantPlayer, Adobe Media Player, as well as some browsers that support it natively. Some advantages of SMIL are:

- The descriptive format is separated entirely from the media recordings. The entire presentation can be defined within the XML document.
- SMIL is a W3C recommendation and is the most standardized format available for describing multimedia presentations at this point.
- SMIL supports an extensive set of features to describe media presentations.

Some disadvantages of SMIL are:

- Poor community activity
- Very few learning resources available
- Development of SMIL seems to have stopped (the Synchronized Multimedia Working Group has been closed since 2012 and the last SMIL specification dates back to 2008).
- No visual editor exists for creating SMIL presentations.

2.6.2 CacophonyJS

Another development in the area of interactive media presentations is cacophonyjs, which is a javascript framework. Presentations in Cacophonyjs are described using a javascript array where each array item contains an effect or command. The author chooses a number of beats per minute and on every beat Cacophony runs the next command from the array. Interactivity can be used to jump back and forth in the array.

The descriptive format of CacophonyJS is much less expressive than that of SMIL. It is limited to use on webpages as there are no alternative players that support their Javascript based descriptive format.

2.6.3 PopcornJS

PopcornJS is a framework that is not specifically intended for creating interactive media presentations similar to XIMPEL or SMIL, but rather it is a Javascript code framework that makes dealing with media within the browser easier. PopcornJS offers functionality to let media items control parts of a website, for instance showing text on a part of the website at a specific time during the video. Popcorn creates a common wrapper around media items so that they can be controlled in a uniform way. It also makes it easy to respond to events such as the ending of a video or the start and end of loading a media item. Popcorn is mainly focused on controlling media items and reacting upon events comming from a media item. While PopcornJS does not provide a descriptive format for media presentations, it includes a lot of functionality that a player for a descriptive format could make use of.

PopcornJS is a widely used framework with many learning resources and active community. It is still being developed, but the documentation is not always up to date, especially the documentation on official plugins is outdated, where usage examples are listed in their documentation but are not working anymore.

3. Planning the XIMPEL framework

In this chapter we take a closer look at XIMPEL as a framework and what choices have been made for the new implementation of XIMPEL. Both the functional and non functional requirements will be specified.

3.1 XIMPEL as a framework

XIMPEL is not only an application that runs a media presentation but it also exposes an API that can be used by other scripts on the web page. This allows the web page to interact with the XIMPEL application, for instance by jumping to a specific part of the presentation when a link on the web page is clicked or by reading the current score within the XIMPEL presentation.

3.2 Functional requirements

In this section the functional features of the new implementation are discussed.

Playlists Playlists essentially define the interactive presentation. Playlists specify the different media objects (videos/images/etc.) that are part of the interactive presentation and they specify when those media objects will be "played". When a specific media object is being played, can depend on user input or it may be defined by a pre-defined playback order. The XIMPEL playlist is an XML file, but in principle it could be in any format.

Subjects Playlists define the media presentation. The main building blocks of a playlist are subjects. A subject defines a part of a presentation to which can be jumped. When one subject is done playing, either the next subject starts playing or the XIMPEL player waits for user interaction to decide which part of the presentation to play next.

Media types Even though we often talk about interactive video in this paper, a XIMPEL application does not need to include video at all. There are three media types supported by default in the old implementation, which are still supported in the new implementation. These are images, video and youtube-videos. It is possible to easily extend XIMPEL with more media types.

Overlays User-interaction is what makes XIMPEL different from regular video. User-interaction in XIMPEL is implemented through the use of overlays. Overlays are click-able elements that can be defined within a playlist as part of a media item. The XIMPEL framework spawns these overlays when playing that media item. The definition of the overlay within the playlist file specifies what happens when the overlay is clicked. The possible actions upon clicking an overlay are:

- Jump to another part of the playlist (a subject).
- Change a score variable.

Customizable overlays Overlays are automatically created by XIMPEL, but it should be possible to define the appearance and other properties of the overlay. The following properties are supported:

- Styling: shape, size, color, opacity, background-image and rotation
- Position (X and Y coordinates)
- Text (including the styling of the text)
- Start time (time at which to show the overlay relative to the start time of the media-object)
- Duration to be shown
- Delayed jump (delay the click action of the overlay until the video is finished)

Additionally the color, opacity, background-image, rotation and text (including text styling) must be definable for the default overlay and for when a user hovers over an overlay.

Questions and Scoring A key feature of XIMPEL is to be able to ask the user questions. Additionally, XIMPEL must also support a scoring system where based on the interaction with the user one or more different scores are being tracked. For instance the score can change based on the answers given to certain questions.

A method must exist to evaluate the scores, showing the user different messages based on their score(s) at the end of the video. Additionally, there must be a method for evaluating scores at runtime and provide custom score evaluation handlers. This provides a way to affect the playback order based on performance (score) of the user. For instance an interactive tutorial video could use this to repeat a quiz until the student has answered enough questions correctly.

Interface customization Besides the playing of media-objects and the overlays, there may be additional elements in the interface. For instance playback buttons (play, pause, next, previous) and score values. Which parts of the interface are visible must be customizable. Other interface customization options may be provided.

Configuration All configuration specific to a XIMPEL video is specified within a configuration file. This may included paths to the playlist and other files used by the XIMPEL interactive video and it includes all other possible configuration parameters. However, all the configuration options should have default values, such that a configuration file becomes optional and only needs to be used when the default configuration values do not suffice.

Audio Audio could be supported in three ways:

- Specifying background music for a media item, a subject or the entire XIMPEL app.
- Creating a media type for audio without supporting parallel media items.
- Creating a media type for audio and allow multiple media items to be played simultaneously.

The first method allows the author to use background audio on a media type, subject or throughout the XIMPEL presentation, but it does not provide fine-grained control over when the audio plays. The second method allows XIMPEL to play audio but not while playing other media items. The third method is most powerful as it allows video and audio to be played simultaneously. However, this makes the usage of XIMPEL more complex and may go against its philosophy. A combination of these options is also possible.

For the first version of the new implementation we will at least support the second option: a seperate media type for audio.

3.3 Non-functional requirements

3.3.1 Framework ease of use

The focus of XIMPEL is to easily create interactive videos. This denotes an important requirement regarding the ease of use of the framework. To guarantee ease of use, the XIMPEL application should adhere to the following conditions:

- An interactive video is defined entirely within the playlist and configuration file. (apart from XIMPEL extensions such as custom media types).
- XIMPEL is provided as one file to be included.
- The parser gives back information when a playlist or configuration file is not valid.
- The playlist syntax must be as minimal as possible.

3.3.2 Browser, platform and device support

The following browsers must at least be supported (the most recent versions at time of writing):

- Firefox

- Internet Explorer
- Chrome

Because the re-implementation is meant as a clean start for XIMPEL, it has been decided that we only focus on the most recent versions. The new framework is aimed at the future and as such there is no backwards compatibility with the old framework and older browsers are not supported.

3.4 Technical constraints and design issues

3.4.1 Third party libraries

Using libraries as part of the XIMPEL framework has clear advantages. However, when libraries are used, such as jQuery or PopcornJS, then a XIMPEL application may have conflicts on webpages that already contain another version of this library. Additionally, if the libraries become outdated or are not backwards compatible and use features that are no longer supported by browsers, it is required to make changes to the library, which may be a difficult task.

Decision criteria The following decision criteria are used to aid the decision making process for including third party libraries.

- **Development activity**: the degree in which the library is still actively being developed and supported.
- **Community activity**: the amount of activity within the community (forums, tutorials, examples)
- **Stability/Maturity**: The quality of the documentation, how well the framework has been tested and whether previous changes were often backwards compatible.
- **Usefulness**: the amount of functionality that is provided by the library and the difficulty of reproducing that functionality.
- **Coupling**: the degree to which using the library binds your application to the library. (ie. whether it is still possible to stop using the library in later stages).

Jquery

- **Development activity**: very active.
- **Community activity**: very active.
- **Stability/Maturity**: very stable and mature and good backwards compatibility.
- **Usefulness**: offers very general functionality useful in nearly every project.

- **Coupling**: low to medium coupling depending on the used functionality. It may be used throughout many parts of the application but it does not generally influence the structure of the application.

PopcornJS

- **Development activity**: moderately active.
- **Community activity**: moderately active.
- **Stability/Maturity**: Low to moderate backwards-compatibility as many online examples do not work anymore and support for several official plugins has been dropped. Additionally, the documentation is sometimes outdated.
- **Usefulness**: it makes cross-browser media support for media events and playback easier to manage.
- **Coupling**: low coupling. It just creates a wrapper around a media object offering a normalized API. By using a popcorn wrapper only through methods of a XIMPEL media type, the support for PopcornJS could be dropped in the future relatively easily.

BackboneJS, KnockoutJS, AngularJS and EmberJS (MVC frameworks)
Because these frameworks would fulfil a similar role they are evaluated together.

- **Development activity**: high development activity for each of the frameworks.
- **Community activity**: high community support for each of the frameworks.
- **Stability/Maturity**: each of the frameworks is mature and stable.
- **Usefulness**: each of those frameworks bring a model-view-controller (or similar) structure to the application. Can be very useful but also limits the freedom of application structure.
- **Coupling**: tightly coupled. The frameworks to a large extend dictates the structure of the application, which is their main reason of existence. However, this means that once the application uses the framework, it is hard to stop using it in the future.

4. Technical specification

4.1 Architectural overview

4.1.1 XIMPEL high level structure

The XIMPEL application takes input and based on that input it outputs a presentation. There are two types of inputs involved:

- The XML documents (playlist and config file)
- User interaction

The first type of input, the presentation description, is a static input and passed to XIMPEL before the presentation is started. The second type of input is dynamic input passed by the user during the playback of the presentation. Because the playlist and config files are static inputs they can be transformed into a different format that is better manageable by our XIMPEL application before the presentation starts running. Figure 4.1 illustrates the high level overview of what XIMPEL does. The architecture of our application is based on this high level overview.

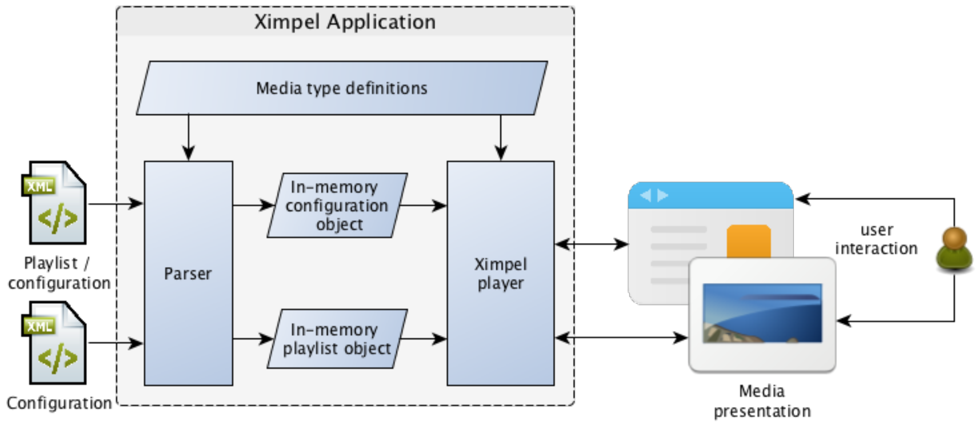

Figure 4.1: An overview of XIMPEL.

The playlist XML file and the configuration XML file are passed to the parser. The parser takes the playlist and configuration data and transforms it into a model more convenient to use by the rest of the XIMPEL application. These models are then given to a player which runs the presentation based on the information from the models. The player outputs a presentation by producing HTML that is appended to the DOM tree of the web page. The player also attaches event handlers to some HTML elements to listen for user interactions and act upon these interactions appropriately. All of these components combined form our XIMPEL application. We will later discuss the media type definitions that are mentioned in figure 4.1.

4.1.2 Components

The high level overview given in section 4.1.1 shows the rough structure of our application. in this section we will further specify each of these elements.

Our implementation consists of three main components. These are the XimpelApp component the Parser component and the Player component.

4.1.2.1 XimpelApp

The XimpelApp component orchestrates all the tasks involved in getting the player up and running. Every XIMPEL presentation on a web page has its own XimpelApp instance. An instance of a XimpelApp object is created by the web page. The XimpelApp instance than orchestrates the whole process from loading the playlist and configuration files to starting the Player component. This process involves the following tasks:

- Accept the information about which playlist and config file should be loaded.
- Load the playlist and config files from the server.
- Create a Parser instance.
- Pass the loaded playlist and configuration data to the Parser and receive the resulting Playlist-Model and ConfigModel.
- Create a Player instance
- Pass the PlaylistModel and ConfigModel to the player component.
- Start playing the Player component.

4.1.2.2 Parser

The parser is a stateless component that has only one public method. This method takes two XMLDoc objects, one XMLDoc that contains the content of the playlist file and one XMLDoc that contains the content of the configuration file. The parser traverses the tree structure of the XMLDoc and builds a PlaylistModel and ConfigModel object based on these XMLDocs. The PlaylistModel itself consists of many other types of models such as SubjectModel, OverlayModels and many more. These models contain the information extracted from the XMLDoc object.

4.1.2.3 Player

The Player component is the component that uses the models produced by the Parser to determine when to show what. The Player is the most complex component and because of that it is split into several sub-components or objects that each handle a small task.

A XIMPEL playlist consists of a list of subjects. These subjects are wrappers around small parts of the presentation and function as anchor points to jump to. Each subject consists of a sequence of items that are to be played one by one. In the current implementation the items within this sequence are always media definitions, but in future versions this sequence may consist of other sequences or groups of items that should be played in parallel.

The parser transforms all the subjects in the playlist XML file to SubjectModels. A SubjectModel contains one SequenceModel that specifies the sequence of items that should be played for that subject. For instance, the following example snippet from a playlist contains a subject that has a main sequence containing two media definitions:

```
<subject>
    <media>
        <image src="test.png">
            <overlay x="0" y="0" width="0" height="0" text="Example" />
        </image>
        <image src="test2.png">
            <question>Do you see a cow on this picture?</question>
        </image>
    </media>
</subject>
```

The media definitions are stored in MediaModels. A sequence of items is stored within a SequenceModel. An item within a SequenceModel can be a MediaModel, a SequenceModel or a ParallelModel. In the example above the main SequenceModel will contain two MediaModels (one for each of the images).

When the player wants to play a subject, it takes the SubjectModel corresponding to that subject and retrieves the SequenceModel stored within it. The Player then tells the SequencePlayer component to start playing that sequence. The Player will get notified when the SequencePlayer finished playing.

Figure 4.2 indicates what the content of a SequenceModel may look like. It is a SequenceModel containing 5 items. The first two items are MediaModels (one for the Audio and one for the Video). The third item is a ParallelModel which in turn contains three MediaModels. The fourth item is a MediaModel again and the fifth item is another ParallelModel. The ParallelModel contains two MediaModels and a SequenceModel. The SequenceModel has three MediaModels within it.

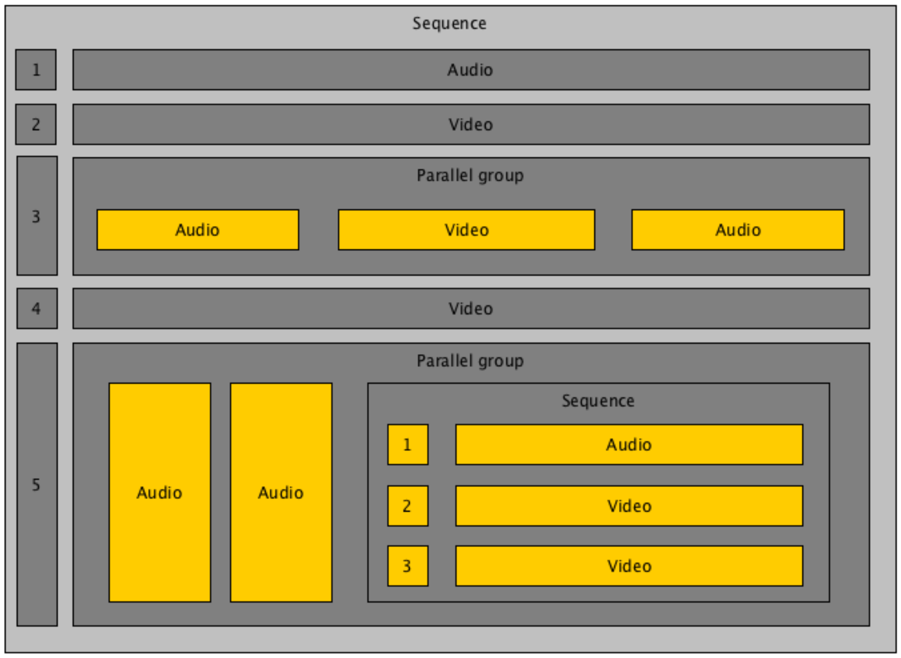

Figure 4.2: An example structure of a SequenceModel. A SequenceModel may contain MediaModels, ParallelModels or another SequenceModel

The order of playback for the SequenceModel in figure 4.2 is as follows:

- Audio
- Video
- The Audio, Video and Audio within the ParallelModel all play at the same time.
- Video
- The Audio, Audio and sequence all play at the same time, where the Audio, Video and Video within the sequence play one by one.

Figure 4.3 gives an overview of the different sub-components used by the Player to play the SequenceModels, ParallalModels and MediaModels.

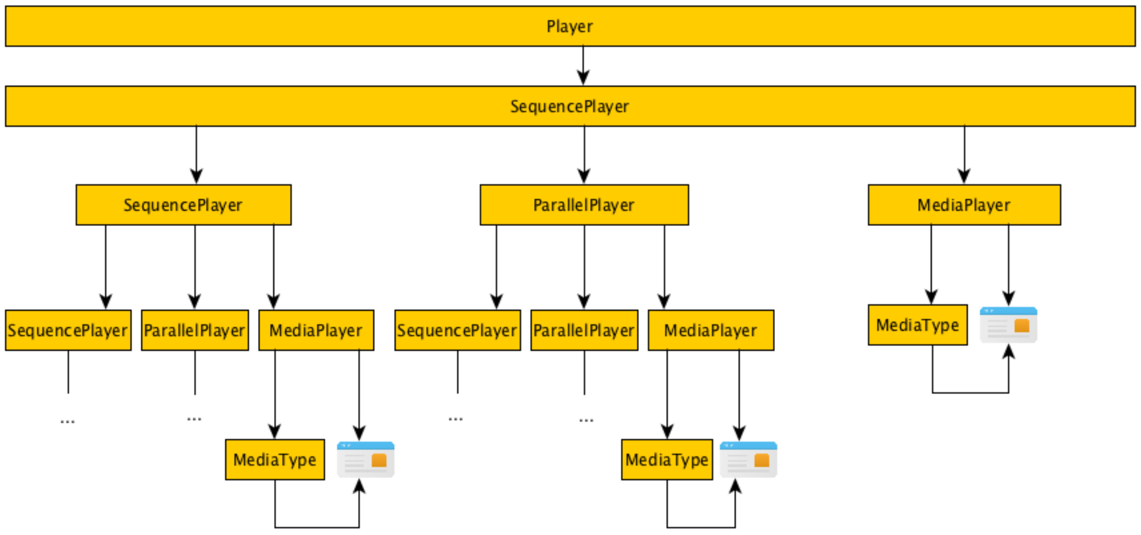

Figure 4.3: SequenceModels are played by the SequencePlayer, ParallelModels are played by a ParallelPlayer and MediaModels are played by a MediaPlayer

SequencePlayer The SequencePlayer is responsible for playing the list of items specified within a SequenceModel. It takes the first item and plays it. As soon as the item is finished it starts playing the next item. When the SequencePlayer encounters an item that is a MediaModel it will use a MediaPlayer to play that item, when it encounters a ParallelModel then it will use a ParallelPlayer and when it encounters a SequenceModel then it will use another SequencePlayer.

Note that the SequencePlayer does not really play anything as it does not produce an output. However, if one of the items within the sequence is a MediaModel then the SequencePlayer will use a MediaPlayer to actually start playing the media.

ParallelPlayer Eventhough the ParallelPlayer has not been created yet we discuss it here because it has been a part of the design of XIMPEL. The ParallelPlayer uses a ParallelModel and plays each of the items in the ParallelModel simultaneously. If there are five MediaModels specified within the ParallelModel then the ParallelPlayer will use five MediaPlayers to play the MediaModels simultaneously. If the ParallelModel contains two MediaModels and a SequenceModel then it will use two MediaPlayers and a SequencePlayer and play all three at the same time.

Note that like the SequencePlayer, the ParallelPlayer does not really play anything as it does not produce an output. However, if one of the items within the ParallelModel is a MediaModel then the ParallelPlayer will use a MediaPlayer to actually start playing the media.

MediaPlayer The MediaPlayer object plays one single media definition. A media definition specifies a media item that is to be played including all the presentational elements (overlays, questions, etc.) that are to be shown during the playback of that media item. All the information of a media definition is stored within a MediaModel.

The MediaPlayer uses a MediaModel to determine what to play and which presentational elements to show at which times. The MediaPlayer can play different media types (images, videos, etc) by creating an instance of a media type and then tell that instance to start playing. For example if a MediaModel indicates that a video is to be played, then the MediaPlayer will create an instance of the Video media type and tell that instance to start playing. The MediaPlayer also makes sure other presentational elements are shown at the right time. It shows and hides overlays at the correct time and it uses a QuestionManager object to display questions at the appropriate time.

Media types Media types are added to XIMPEL using a plug-in based system. Such a plug-in is a Javascript object that uses a MediaType object as its prototype (ie. its parent). The custom media type must implement a number of methods to be a valid media type. Instances of the media type object can be created and each instance then contains the required methods, such as playMedia() and pauseMedia(). We call the instances of such a media type a media item. The MediaPlayer discussed earlier is responsible for starting, pausing and stopping these media items when necessary.

4.1.3 Custom media types

We have implemented custom media types in the form of a plug-in system. The custom media types shipped with XIMPEL have no special integration with XIMPEL in any way. Anyone can add media types to the framework and can make use of the same capabilities as the build-in media types. The following requirements should be fulfilled to define a custom media type.

- Constructor function: instances of the media type are created by a constructor function (a function that is used with new keyword).
- Media type API: the object created by the constructor function implements the media type interface (ie. the object contains a number of functions that each media type should have).
- Media type registration: the media type has to be registered with XIMPEL.

At this moment the interface that the media type should implement consists of the following methods:

- mediaPlay: starts/resumes the media item
- mediaPause: pauses the media item

- mediaStop: stops the media item entirely (ie. it will start playing from its original starting state when play() is called)
- mediaIsPlaying: should return true if the media item is playing, false otherwise
- mediaIsPaused: should return true if the media item is paused, false otherwise.
- mediaIsStopped: should return true if the media item is stopped, false otherwise.

Note that other methods are available for each media type by default. These methods are currently:

- addEventHandler() (currently only supports the 'ended' event)
- removeEventHandler()
- destroy()

4.2 Design choices and issues

4.2.1 Resizing presentations / fullscreen

One of the requirements for the XIMPEL framework was to be able to display the presentation in different dimensions without having to adjust the positioning of elements within the playlist. This requirement has been fulfilled by using the transform capabilities of CSS, in particular the scaling capabilities of CSS. While support for this feature was lacking in the past, support now is much better with all the newest versions of the major browsers supporting it. The XIMPEL framework currently uses a native resolution of 1920*1080. The content of the presentation has dimensions and positions defined assuming the presentation is displayed in 1920*1080. However, when the container in which the XIMPEL presentation is ran is given other dimensions than 1920*1080, then the content is scaled to fit within the container while maintaining the aspect ratio. If the container does not have the same aspect ratio as the aspect ratio of 1920*1080, which is 16:9, then black bars will appear around the presentation.

This same technique is used to support fullscreen presentations. When going to fullscreen mode using the HTML fullscreen capabilities, what actually happens is that a container is given the dimensions of the entire screen. XIMPEl detects this and scales the content to fit within the new size of the container.

4.2.2 Video on canvas or the HTML5 video element

In our implementation we have chosen to use the HTML5 video element directly to display the videos in XIMPEL. An alternative could have been to use the video element in the background and draw the frames to a canvas element. An advantage of using this technique is that the data on a canvas can be manipulated allowing you to change the video at run-time. For instance by changing the data such that the video is shown in gray scale.

We have chosen not to use the canvas for showing video because at this moment we do not support filters in XIMPEL. Additionally, using the video element directly, we can conveniently re-size the video without worrying about the aspect ratio of the video. If we resize the video to dimensions with a different aspect ratio, then the video will automatically adjust and show black bars around the video. If we would draw the frames on a canvas we would have to deal with this logic ourselves. If we ever need to directly manipulate video data we only have to change the Video media type implementation by making it use a canvas instead of a video element.

4.2.3 Rendering overlays

Overlays are click-able regions within the XIMPEL presentation. There are different possibilities for implementing overlays. One method is to draw the overlay shape on a canvas and manually calculate whether a mouse click is on that part of the canvas. In that case, we need to manually check for the mouse click position because the HTML5 canvas is a single DOM element regardless of what has been drawn on it. As a consequence, it is not possible to attach event handlers to specific drawings on the canvas. The advantage of drawing overlays on a canvas is that more complex shapes can be used.

Another option is to use regular HTML elements as overlays, which means that event handlers can be attached to them directly without having to manually calculate whether an overlay was clicked. By using DOM elements less complex shapes can be used (only shapes supported by CSS). We have chosen to use regular DOM elements as overlays because at this moment there is no requirement for complex shapes.

4.2.4 Preloading

To have smoother playback even in bad network environments, pre-loading could be used. However, since we do not know how big the presentation is we cannot pre-load all media items by default. Additionally, since the order of the presentation is non-deterministic we cannot know which item will play next. For this reason we have decided not to pre-load media types. This keeps the architecture of the application simple.

4.2.5 Conditional jumps

With conditional jumps the author can define a branching in the flow of the presentation based on a condition. We have implemented conditional jumps in the new implementation. However, the exact syntax in the playlist that should be used to specify a condition was not yet known or determined. For that reason we have chosen to use a very liberate condition specification which just accepts a string where XIMPEL variable names can be templated into. After replacing the variable templates with their respective value, the string is passed to the javascript eval() function. The eval() function has security risks and may cause Javascript code minimization to fail. For that reason, the current method for specifying a condition is just a temporary solution and should probably be changed in the future. However, we have made it easy to change the way a condition is specified in the future.

In the parser, we translate a condition to a ConditionModel object. This object is stored within the XIMPEL player and whenever the condition needs to be tested it is passed to an evaluation function. If in the future we want to change the way that the condition is specified in the playlist then we only need to change the parser to deal with the new playlist syntax and possibly construct a different ConditionModel object. If the structure of the ConditionModel object is changed than the evaluation function should also be changed to make use of that new condition model. The rest of our program doesn't need to know anything about the contents of the condition model and the properties and methods within it.

4.2.6 XIMPEL Namespace

In javascript there is always a global object. In browsers this global object is the window object. Whenever a variable or function is declared within the global scope (ie. outside any other function) then these variables or functions are stored int he global object, the window object. It is considered bad practice to add variables and functions to the global object because this may create conflicts between different javascript libraries included on the same page. Since our XIMPEL framework will be used within existing websites, we do not want to add all our variables and functions to the global object.

Instead we created one object named 'ximpel' and we attached all our objects and functions to that namespace. This has a slight performance drawback because for every function reference we now need to do an additional lookup for the 'ximpel' object (ximpel.doSomething() instead of doSomething()). However, the XIMPEL framework is not a heavy application in the sense that most of the load is caused by the media types rather than the code of our framework.

4.2.7 Classical Prototypical Inheritance

Javascript offers inheritance in the form of prototypes. We make use of function constructors to construct object instances. These instances all have access to a prototype object. By placing the methods for our instances on the prototype object we only have to store the methods once and they can be referenced by all of our instances. This saves memory because Javascript will not have to load all the methods in memory each time a new instance is created. The technique that makes this possible is prototypical inheritance which is based on prototype chains. How this works exactly is not in the scope of this document.

4.2.8 Polyfills

Since many browsers do not support all features of ECMAScript 5 yet we have created polyfills for a number of features. Polyfills are custom implementations of functionality that is expected to be supported by a browser but are in fact not supported by all browsers. The polyfill is only used when the feature is not supported by the browser natively, if the feature is already supported then the native implementation is used. This is accomplished by using feature detection in javascript.

5. Proof of concept

As part of our exploratory research we created two proof of concept presentations. These proof of concept presentations are used to verify if and how well the HTML5 implementation suffices as an alternative to the flash implementation. The first proof of concept presentation is a presentation that is a re-creation of an existing presentation that was created for the flash implementation of XIMPEL. The second proof of concept presentation is a dummy presentation that covers most functionality of the new implementation of XIMPEL.

5.1 Presentations

The presentation called "Zaanse schans" is a presentation originally created with the flash implementation of XIMPEL by a master student at the VU in Amsterdam. This is a large presentation that makes use of the core functionality of XIMPEL. This presentation has been chosen as a proof of concept presentation for our implementation because it uses many large video files and includes the very core functionality of XIMPEL. Since the HTML5 implementation is not backwards compatible with the flash implementation due to changes in the playlist structure and differences in the used resolution and aspect ratio, we had to re-write the playlist to adapt to the changes.

The second proof of concept is not a presentation with sensible content. Instead it uses arbitrary content and makes use of all the features that the new implementation supports. It is solely aimed at illustrating that the functionality in the framework works. It does not cover all possible test cases because this is something that should be done using test code. It is rather meant as a means to illustrate that the concept of an HTML5 implementation for XIMPEL works.

5.1.1 Result

The first proof of concept called "Zaanseschans" shows that a presentation with basic functionality that was created for the flash implementation is still possible using the HTML5 implementation. The presentation runs smoothly and has the same flow as the flash version. There are some minor differences in the positioning of overlays because the x and y coordinates of the overlays were changed to accommodate for the different aspect ratio that is in the new version.

The second presentation which we will call the "feature test" shows the different features that XIMPEL offers through a playlist document. The following features are covered:

- Playing videos, audio fragments, images and Youtube videos
- Setting a startTime for videos, audio fragments and youtube videos.
- Setting a duration for a media item (this feature is not specific to a media type)
- Linking subject using the leadsTo attribute on the subject element, on the media type element and on the overlay element.
- Declaring and changing variables when a subject is started and when a media item is started.
- Using conditional jumps based on XIMPEL variables and the leadsTo attribute.
- Using a custom media type "message" (note that in principle a custom media type is no different from a core media type except that it doesn't ship with XIMPEL by default)
- Asking single questions and asking lists of questions.
- Setting start times and timeouts for questions.

Note that this list of functionality is not exhaustive. There are many more parameters for the different elements that can be used, but these cover the most important functionality. The remaining options are mostly options to change the appearance of the different presentational elements, such as the width, height, position, color and size of overlays or media items.

6. Future extensions

We re-implemented the main features of the existing XIMPEL framework and focused on some important extensions. Some lower priority features have been left out due to time limitations but will probably be added in future versions. In this chapter we will discuss some possibly interesting features that were not included in this version of XIMPEL but maybe added in the future. Additionally this chapter will include a list of improvements upon existing features.

6.1 Additional features

6.1.1 Parallel media playback

One of the future extensions that is not available in the new XIMPEL implementation but has been thought about during the design process is the possibility to play media items in parallel. Showing multiple media items at the same time allows for more complex presentations. For instance authors will be able to create presentations where multiple images can be played at the same time or videos can be shown with related images next to it.

The architecture of the new XIMPEL implementation has been designed to be easily extended with the possibility of parallel media playback. As described in previous chapters the new implementation uses a MediaPlayer to play one single media definition from the playlist. The media definition includes the media item that is to be played and the additional presentational elements that need to be shown during that media item. The media player handles the playback of all these presentational elements. The visual elements that should appear during the playback of a media definition are all attached to the main DOM node of this XIMPEL player. In the current situation where there is only one MediaPlayer playing at the same time, all the DOM nodes within the player element will belong to that one MediaPlayer object.

The SequencePlayer plays a sequence of media definitions. It picks the first media definition in the sequence and plays it. When that media definition has finished playing, the next media definition is played and this process continuous until all media definitions in the sequence have been played. We call this sequential playback. To support parallel playback, we will have to write a ParallelPlayer which functions similar to the SequencePlayer except that it plays the media definitions in parallel rather than in sequence. This can be done by playing multiple media players at the same time. Each of those media players would add its DOM nodes to the player element.

It is the responsibility of the author to not provide media definitions that interfere with each other (for instance, overlapping questions or overlays). In most cases it will be best to add presentational elements (except for the media itself) to only one media definition in a parallel group. If media items that play in parallel overlap, then there must be a way of managing precedence. If no precedence rules can be defined then the media item that plays first will always be covered under media items that are started later. To define these precedence rules the MediaPlayer can be extended such that it can deal with z-index settings. The author can then define a z-index for a media item within the playlist file. The MediaPlayer will show media items with a higher z-index on top of media items with a lower z-index.

Example In the current implementation a sequence of media definitions can be defined like this.

```
<subject>
    <media>
        <video ... />
        <audio ... />
    </media>
</subject>
```

These media definitions play one after the other. The sequential order is implicitly defined here. The above playlist can be interpreted like this:

```
<subject>
    <media>
        <sequence>
            <video ... />
            <audio ... />
        </sequence>
    </media>
</subject>
```

Everything within the "media" tag is considered to be played sequentially. To support parallel playback the syntax could be similar to this:

```
<subject>
    <media>
        <parallel>
            <video ... />
            <audio ... />
        </parallel>
        <picture .... />
    </media>
</subject>
```

In this case everything within the "media" tag is still played in sequence. First the parallel group is played and after that the picture is played. The above example can be interpreted like this:

```
<subject>
    <media>
        <sequence>
            <parallel>
                <video ... />
                <audio ... />
            </parallel>
            <picture .... />
        </sequence>
    </media>
</subject>
```

Whenever the parser encounters a "sequence" tag or a "parallel" tag it would create a SequenceModel or a ParallelModel. These models would contain a list of all the items defined between these tags. The XIMPEL player always starts with a sequence player that plays the main sequence. The main sequence is the sequence defined in the "media" tag. The sequence player plays each of the items within this main sequence one by one. When it encounters a media definition in its list then it starts a media player to play that item. When it encounters a SequenceModel within its list then it starts a sequence player. When it encounters a ParallelModel in its list then it starts a ParallelPlayer. This behaviour also applies to the ParallelPlayer. This means that a playlist like this is possible:

```
<subject>
    <media>
        <sequence>
            <parallel>
                <video ... />
                <sequence>
                    <video ... />
                    <video ... />
                    <video ... />
                    <parallel>
                        <video ... />
                        <video ... />
                    </parallel>
                </sequence
            </parallel>
            <picture .... />
        </sequence>
    </media>
</subject>
```

However, it may be evident that this becomes confusing and because of that multiple levels of nested parallel and sequence tags should be avoided and are probably not required in the first place.

6.1.2 Mini-games

XIMPEL is a framework that focuses on e-learning. An important research area in e-learning is Serious Games. An interesting extension to the XIMPEL framework may be to add Serious Games as part of a XIMPEL presentation. The Serious Game implementation itself would not be part of XIMPEL but XIMPEL should allow these games to be started as part of the presentation. To include such mini-games within a presentation the following is required:

- XIMPEL should be able to start, stop and pause the mini-game.
- The results of the mini-game should be able to influence how the presentation proceeds.

It is possible to create and register media types to the XIMPEL framework easily. The only requirement is that the media implements a by XIMPEL required API. A mini-game could be added to XIMPEL in the form of a MediaType. This requires a media type implementation that implements the media type API. This API is quite simple and includes methods such as play(), pause(), stop() and isPlaying(). The play() method of the media type would start the mini-game, the pause() method would pause it and the stop() method would stop it. By considering the mini-game as a media type, we covered the first requirement listed above.

The second requirement listed above is not yet handled by the XIMPEL implementation. The media type that implements the API that XIMPEL requires will do the communication with the mini-game implementation. XIMPEL starts the media type by calling the play() method. The play() method will start the mini-game (which is just Javascript code). The media type implementation must also ensure that it somehow gets the result of the mini-game when the mini-game finishes. Because we don't want XIMPEL to be bound to specific media types in special ways that only apply for that media type, we must make the communication between the MediaType and XIMPEL generic.

By allowing media types to throw pre-defined events for specific actions that XIMPEL should perform we can allow mini-games to communicate with XIMPEL without creating an explicit binding between the two. To implement mini-game support the MediaPlayer should listen for a pre-defined event. For instance an event "VariableUpdate" that tells XIMPEL to update a specific XIMPEL variable/score. The MediaType throws the event and includes the information about which variable/score should be updated and how it should change. The media type gets this information from the playlist and from the result of the mini-game. This allows the MediaType to communicate a score change to XIMPEL in a generic way (ie. each other media type could use the "UpdateVariable" event. Several other events could be be supported as well, for example an event that makes XIMPEL jump to a specific subject.

6.1.3 Global overlays

Currently overlays are always defined as part of a media definition. This means that when a media item has finished playing, the media player stops and all the overlays disappear. There may be cases where it is useful to be able to show overlays during several media items or throughout the entire presentation. We could define overlays within a subject such that the start time and end time of overlays are relative to the subject's playback time and the overlays disappear when the subject finishes playing. Additionally we could define overlays within the playlist element such that the start time and end time of overlays are relative to the entire presentation.

In order to do this some changes need to be made. First of all XIMPEL should keep track of how long a subject has been playing and how long the entire presentation has been playing. Additionally there are changes required to the OverlayManager as this component is currently bound to the media player but should be made generic such that the Player object could also make use of the Overlay manager. In that case we may also want to create a SubjectPlayer object which is in charge of playing subjects which would then include handling the playback of subject-overlays. Right now there is no separate SubjectPlayer yet.

6.1.4 Additional media types: Google Maps

More available media types allow for a richer experience for both developers and users of the applications. There are many possibilities for media types. For instance Google Maps. A Google Maps media type could show a Google Maps instance within a XIMPEL application including markers, routes and public transport information. The Google Maps media type is just like any other media type and will use the Google Maps API to draw google maps instances within our presentation. The media type only needs to implement a number of basic methods so that XIMPEL knows how to start and stop it.

6.1.5 Generating playlists

Another feature that may be interesting in future versions of XIMPEL is to automatically generate playlists. This can be done in two ways. First it can be done by generating the actual XML content and secondly it can be done by generating the PlaylistModel javascript object that the Player object uses to determine how to play the presentation.

This content can be automatically generated before the presentation is running, but it could also be generated at runtime. If it should be possible to add content at runtime, then the XIMPEL player should be extended with methods to add this content to the PlaylistModel object.

6.2 To do

This project was focused on creating an HTML5 version that was similar to the flash version of XIMPEL. We added some new functionality and we looked at how some interesting features could be supported. Unfortunately there was not enough time to also implement all of these interesting features as part of this project. Additionally, there are still some development goals that were not fully addressed yet. In this section we provide a to-do list which can serve as a reference to future developers that proceed with the development of XIMPEL.

Input validation Right now the playlist XML document is passed to parser in the form of an XMLDoc object. The parser will traverse the XMLDoc like a tree and extract information from it to build models containing the information needed to play the presentation. However, the parser does not validate the information that it puts into the model. This has two disadvantages:

- If the playlist comes from an unreliable source then it may be a security threat.
- If the author of the playlist makes mistakes then these are not detected by the Parser. This may have undefined consequences during the playback of the presentation.

To handle the first point of concern the parser should check whether the content that it puts into the models doesn't contain harmful data to prevent cross-site scripting attacks. The exact filter methods to prevent this is beyond the scope of this project. To handle the second point of concern the parser should check if the data supplied in the playlist has a valid structure. Take, for instance, the following example:

```
<image width="150" .... />
```

In this case, the parser should know that the value of the "width" attribute requires a unit to be specified (for example "px" or "%". It should either give an error or it should assume a certain unit and add that to the value. Currently the player simply accepts the input value, which may or may not cause problems during the playback of the presentation. Besides checking for wrong values it also a good idea to check for invalid playlist structures and present errors to the user if the structure is invalid.

This kind of feedback can help the author a lot and thus make the usage of XIMPEL much more user friendly.

Loading dynamic playlists Currently the XimpelApp object has a load() method that loads a playlist and a config XML document from the server, uses the parser to create a PlaylistModel and ConfigModel, creates a Player object, passes the models to this Player object and then starts playing the presentation by calling play() on the Player object. However, there are cases where you don't want to load the playlist document from the server but you want to pass the playlist as an XML string. The XimpelApp object should be adapted such that it offers a way to specify where and how to get the playlist.

Note that jQuery's parseXML() function can be used to parse an XML string into an XMLDoc object.

Player events There was a wish for the new implementation to be able to be integrated with external elements on a webpage. Right now the integration options are limited. However, they can easily be added by making the player throw certain events, which external components can listen to by calling the addEventHandler() method on the Player object. Additionally, external components have access to a specific player API. Right now this API is limited to methods such as: getScore() or goTo(), but this may be extended in the future.

The basic structure for integrating with external components is there but the number of events produced by the player should be increased and the number of API methods exposed to external components should increase as well. Examples of events that may be useful for external components are events for:

- Subject change
- Score/variable change
- Subject start / ending
- Presentation start / ending

HUD Right now the XIMPEL presentation only shows control buttons (unless disabled in the configuration file) but no meta data of the presentation. A HUD could provide additional information such as current scores or the current subject name.

Text messages In the old implementation of XIMPEL there was a <branchquestion> tag that could be used in the playlist. This tag would show the user a message so that the user was aware a choice had to be made. This could be generalized by providing a <message> tag that can be used to show a user a message at any given time. These messages could be created by using overlays, however for overlays you need to specify the location and dimensions and their appearance is not optimal for longer text messages. It may be a useful feature to provide a <message> tag that shows a message in a consistent way. This message tag can then be used both for providing related information during a presentation as well as for providing a question text when the user is expected to click one of several overlays.

Youtube loading Right now when a Youtube media item is started by the media player, the first frame of the YouTube video is shown for a fraction of a second with a "loading"-icon. This only occurs when the Youtube video is started from a different starting point than the beginning. What happens is that Youtube is buffering to start playing from the given playback point. We have not yet discovered a way to start playing only after the Youtube video has finished buffering from the given starting point onwards. However, the media type may be able to detect when the Youtube video starts playing and hide the element until then.

7. Conclusion

7.1 Summary

We have looked at the benefits that interactive media frameworks have in learning environments. Using interactive media in learning environment can increase satisfaction among students and has been shown to increase test scores as opposed to non-interactive media such as videos and audio fragments. Additionally, people learn in different ways. In order to create a better learning experience, the information should be shaped, arranged and optimized for a specific viewer rather than for a general public. This is something that can be done with interactive media frameworks much more easily then it can with traditional media.

To prove it is possible to offer an educational interactive media framework on the web using HTML5, we have created an implementation of such a framework called XIMPEL. We described the requirements for the framework based on the functionality that was already present in the existing flash implementation and based on the ideas from the original authors of XIMPEL. Creating the new implementation was part of this exploratory research in which we have tried to answer the following question:

- **Question**: Has HTML5/Javascript technology matured enough to support the XIMPEL educational interactive media framework?

To answer this question we have created a XIMPEL implementation in HTML5 and built two proof of concept presentations with it as discussed in chapter 5. With these proof of concept presentations we have shown that the core functionality is certainly possible using today's HTML5 technology. The requirement was for XIMPEL to be able to run in the most recent version of the major browsers which are Firefox, Internet Explorer and Chrome. Since this project was considered as a new clean start for XIMPEL, backwards compatibility and older browser support was not required.

The proof of concept presentations have shown that working with many media files works fine in HTML5. Even though support for video formats may differ between browsers it is possible to specify multiple sources and let the browser pick the source it can support. Additionally, the

MPEG-4/H.264 video format is now supported in all newest versions of all the major browsers. In some cases some non essential Javascript features were not supported by all browsers even though the ECMAScript 5 specification includes the feature. In this case we have used polyfills to implement the feature for browsers that do not support it.

Unfortunately we have not had the time to optimize the framework for mobile usage. However, a quick test of the "feature test" proof of concept presentation on the mobile chrome browser shows that the presentation runs largely as intended with the exception that fullscreen mode does not cover the entire screen but has large black bars around the video which wastes screen space and YouTube videos do not automatically start playing. Aside from those two quirks the presentation runs as intended. These quirks may have easy fixes. It may be the case that the width of a fullscreen element has to be determined differently in the mobile version of the chrome browser. Unfortunately, this does show that HTML5 still has some inconsistencies across browsers.

We conclude that HTML5 with its current support is mature enough to support the XIMPEL interactive media framework at least on desktop platforms where support for the current most recent versions of the browsers Internet Explorer, Firefox and Chrome is enough. Additionally, the re-implementation of XIMPEL in HTML5/Javascript offers promising extensions, by allowing to make direct use of Javascript libraries that can be used for example in integrating games or data visualisations.

References

[1] R. Hammoud, *Interactive Video: Algorithms and Technologies.* Springer, 2006.

[2] B. Meixner and H. Kosch, “Interactive non-linear video: Definition and xml structure,” in *Proceedings of the 2012 ACM Symposium on Document Engineering*, ser. DocEng ’12, Paris, France: ACM, 2012, pp. 49–58, ISBN: 978-1-4503-1116-8. DOI: `10.1145/2361354.2361367`. [Online]. Available: `http://doi.acm.org/10.1145/2361354.2361367`.

[3] B. Meixner, K. Matusik, C. Grill, and H. Kosch, “Towards an easy to use authoring tool for interactive non-linear video,” English, *Multimedia Tools and Applications*, vol. 70, no. 2, pp. 1251–1276, 2014, ISSN: 1380-7501. DOI: `10.1007/s11042-012-1218-6`. [Online]. Available: `http://dx.doi.org/10.1007/s11042-012-1218-6`.

[4] D. Zhang, L. Zhou, R. O. Briggs, and J. F. N. Jr., “Instructional video in e-learning: Assessing the impact of interactive video on learning effectiveness,” *Information & Management*, vol. 43, no. 1, pp. 15–27, 2006, ISSN: 0378-7206. DOI: `http://dx.doi.org/10.1016/j.im.2005.01.004`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0378720605000170`.

[5] T. Phillips, M. Hannafin, and S. Tripp, “The effects of practice and orienting activities on learning from interactive video,” English, *ECTJ*, vol. 36, no. 2, pp. 93–102, 1988, ISSN: 0148-5806. DOI: `10.1007/BF02766617`. [Online]. Available: `http://dx.doi.org/10.1007/BF02766617`.

[6] S. Schwan and R. Riempp, “The cognitive benefits of interactive videos: Learning to tie nautical knots,” *Learning and Instruction*, vol. 14, no. 3, pp. 293–305, 2004, Dynamic Visualisations and Learning, ISSN: 0959-4752. DOI: `http://dx.doi.org/10.1016/j.learninstruc.2004.06.005`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0959475204000337`.

[7] R. Moreno and R. Mayer, “Interactive multimodal learning environments,” English, *Educational Psychology Review*, vol. 19, no. 3, pp. 309–326, 2007, ISSN: 1040-726X. DOI: `10.1007/s10648-007-9047-2`. [Online]. Available: `http://dx.doi.org/10.1007/s10648-007-9047-2`.

[8] “Ecmascript language specification 5.1 edition,” ECMA International, Final ECMAScript 5.1, 2011. [Online]. Available: `http://www.ecma-international.org/ecma-262/5.1/`.

[9] “Ecmascript 5 compatibility table,” Kangax & webbedspace. [Online]. Available: `http://kangax.github.io/compat-table/es5/`.

[10] “Ecmascript 6 compatibility table,” Kangax & webbedspace. [Online]. Available: `https://kangax.github.io/compat-table/es6/`.