Integration Of Web Browsers and Interactive TV

A dissertation submitted for the degree of
MSc in Distributed Systems and Networks

Computing Laboratory
University of Kent at Canterbury

Michael Pediaditakis

August 31, 2001

Abstract

Interactive TV(ITV) technologies enhance the conventional TV services and enable user–content interaction. The World Wide Web is currently the most popular approach to information exchange. The "Integration of Interactive TV and web browsers" project studies the convergence of the two areas which is important since both areas can benefit each other.

We focus on an ITV extension for an existing web browser. The "Multimedia and Hypermedia information coding Experts Group" part 5 (MHEG-5) standard is selected as representative for the ITV area, and the Extensible Markup Language (XML) and Document Object Model (DOM) standards for the web browsers area. We present the design and implementation of an MHEG extension for a web browser and integrate the MHEG and DOM event models in order to achieve an easier and more compact MHEG event model implementation.

Our study provides the foundation for the convergence of the two technologies by a thorough study, implementation and evaluation of ITV and browser integration.

Acknowledgments

I would like to thank my family for their support and guidance during this year of study in the University of Kent at Canterbury and during my whole life in general.

I would also like to thank my tutor and project supervisor Dr. David Shrimpton for his guidance throughout this project, his inspirational ideas and his support during the last difficult days of the dissertation preparation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Interactive TV (ITV) has received major attention since it is a relatively new technology that enhances the TV experience and offers a brand new way of information exchange using the TV set. Moreover, the World Wide Web (WWW) can be considered as the dominant means of communicating information. "Integration of Web Browsers and Interactive TV" project's principal aim will be to investigate the convergence of these two areas.

1.1 The two areas and integration benefits

ITV technologies enhance the conventional TV services by allowing the user to interact with the presented content. The required additional functionality for interaction support is currently provided by a set-top box (STB). The "Multimedia and Hypermedia information coding Experts Group" part 5 (MHEG-5)[14] standard has been accepted as part of the Digital Audio and Visual Council (DAVIC) ITV specification and for the U.K. terrestrial ITV for interactive content representation and handling. The MHEG-5 standard provides a means for representing, transferring and handling data for interactive multimedia client server applications.

Web content is processed and presented by a "web browser" application. Web browsers are mainly focused on the support of W3C international standards which provide a standard and internationally recognized way to represent the information available through the web. Nevertheless, there are always some platform-specific or non-standard extensions that are needed in order to support content types which are not described by international standards and are based on proprietary data formats.

We will mainly focus on the integration of MHEG-5 functionality into the web browsers. This integration will be beneficial since it will provide web browsers capable for interactive multimedia and ITV content handling and will also allow users to seamelessly change between ITV and web content. Moreover, we will see how web technologies can be used in order to support an interactive multimedia application.

1.2 Related work

There have been several proposals for web browser and interactive TV domains convergence. In this section we will present three different approaches which look at the problem from a different perspective.

One of the first proposed solutions[2], when there was no support for web scripts and dynamic HTML, used gateways that emulated MHEG behaviour through dynamically generated HTML pages. The principal aim was to use an MHEG-unaware web browser for presenting MHEG content. The basic problem was the user interaction handling, since, HTML was not adequate for handling MHEG applications. The proposed solution was based on "image-maps" which displayed MHEG content and provided feedback on the user interaction. The proxy was processing MHEG data and generated an "image-map" representing the presentation screen content. User mouse-interaction was fed back to the proxy in terms of mouse "clicks" over the image map. Then, the proxy processed the user interaction and returned a new image map representing the new screen layout. This solution was obviously slow, non-scaleable, required heavy network support and the user experience was less than satisfactory. However, it was the only way to represent interactive multimedia content without modifying the web client.

After scripting support and additional functionality was added to the HTML standard, a different approach became feasible. The basic concept was the conversion from MHEG to HTML content and vice-versa[11]. In order to achieve the conversion, an investigation of the common aspects of the two standards was made and a mapping was provided for the corresponding concepts of both standards. For the MHEG to HTML conversion, in order to support the MHEG features which could not be directly mapped to HTML concepts, JavaScript and Cascading Style Sheets (CSS) were used. This approach allows representation of both content types in a platform that is designed to handle only one of them. Moreover, it provides a standard and platform independent way to integrate the two areas since there is direct translation between the standards and there is no need to focus on specific platforms. The basic drawback is that some MHEG concepts could not be efficiently translated. Additionally, the differences of the display and user interaction capabilities of the corresponding platforms introduce even more problems which are addressed in [23].

The third proposal[8] makes use of downloadable applets that provide the MHEG functionality. Applets allow the development of an MHEG engine in a way similar to a stand-alone application. The only disadvantage of this solution seems to be the fact that it is difficult to use existing browser functionality for supporting MHEG handling and is simply a way of "attaching" an MHEG engine to a browser.

1.3 Our approach

Our approach is to extend a web browser architecture in order to make it MHEG-aware by modifying its implementation. This would require an open source browser and probably more effort than the related work discussed in Section 1.2. The latter is true because it is usually more difficult to support the new content by modifying an existing architecture than to independently implement the new functionality or to convert the new content to an already handled document type. However, this approach would allow the use of existing browser features and already implemented Internet standards to support the MHEG engine extension.

What we want to avoid is a platform and browser dependent MHEG extension. If that was the case it would be easier to develop a "plug-in" that implements an MHEG engine. This approach is not of major interest since it provides no more than an MHEG engine implementation. Therefore, we will focus on using standard browser and international standards functionality for supporting the MHEG extension.

Our goal is to find ways of making a web browser MHEG-aware by using as many already implemented features as possible. This would normally end up to a more compact and efficient MHEG extension implementation as opposed to one which does not use the existing features.

1.4 Document layout

The document is divided into three basic parts: the background information, the integration process description and the evaluation. Moreover, additional informative material that provides details which are not essential for our study is included in the appendices.

Background information is included in the next three chapters. Chapter 2 provides information on the ITV domain and an overview of the MHEG standards. The description of MHEG is essential for the rest of the project since it will be our main point of interest. Chapter 3 is concerned with the other area of the integration, web browsers. It presents the current browsers situation and the XML and DOM standards which will be used extensively for the integration process. Finally, Chapter 4 is concerned with the identification of the requirements for the target browser platform. Several different alternatives are examined and X-Smiles is selected as the most adequate one for our research. Then, an overview of X-Smiles architecture is given in order to provide the required context for our discussion on the extension of the browser.

After the background information, Chapter 5 describes the integration process and our achievements. It describes how the integration process is structured and contains the design, implementation and a first level evaluation of the two main parts of the integration: the MHEG extension and the event models integration.

An evaluation of the whole project and further research ideas are included in Chapter 6. The evaluation follows a bottom-up structure where we start from the implementation evaluation and continue up to a general discussion about the project contribution and the standards used. Further research follows a similar structure by starting by minor implementation problems and ending up with research ideas in the wider area of the web and interactive TV.

Finally, the appendices contain additional information on the abbreviations and the MHEG application domain used, the original project description, the browser alternatives, the examples and the output for the evaluation and some illustrative code examples.

Chapter 2

Interactive TV – MHEG

In this chapter we will give a brief description of the Interactive TV (ITV) domain, its relation to the ISO/IEC MHEG standards[14, 15] and an overview of MHEG parts 5 and 8. Our aim is to signify the importance of MHEG-5 for ITV and to describe the basic MHEG aspects which are important for the rest of our study.

2.1 Interactive TV

Interactive TV technologies enhance the conventional TV services by allowing the user to interact with the TV set.[1] For instance, the user may request information on the TV programe or take part in a multi-player game using the TV set. The ITV information would normally be displayed on top of the conventional TV programe, while the interaction might take place with an enhanced remote control.

In order to support the enhanced ITV functionality, a set-top box (STB) is used[3]. STBs are currently used in order to provide the "missing intelligence" of TV sets. In the future, STB functionality might be integrated into the stock TV sets. STBs need a well defined application programming interface (API) for supporting the interactive services. Currently, there are

[1] In this context the term "interaction" is used as in "interactive multimedia". In other words, it is not meant for conventional TV interactions such as increasing the volume or changing the contrast.

several proposals for the STB API but there is still no international agreement. However, UK terrestrial interactive TV and the Digital Audio and Video Council (DAVIC) have accepted MHEG-5 as the platform for ITV support. Based on these two examples, we will consider MHEG-5 as an important platform for ITV, and our study on the integration of web browsers and ITV will focus on the integration of MHEG-5 functionality into the web browsers.

2.2 MHEG standards

MHEG stands for "Multimedia and Hypermedia information coding Experts Group". The MHEG group of standards is collectively called "Coding for multimedia and hypermedia information", and aims to provide international standard specifications for the encoding of different kinds of multimedia and hypermedia information.

Currently there are 8 parts, which are shown in Table 2.1. The core standard is MHEG-1, which is a generic standard for multimedia object representation that introduces as less constraints as possible in order to support a wide range of multimedia platforms.

Part	Description
MHEG-1	Base notation (ASN.1).
MHEG-2	Object alternate notation (*withdrawn*).
MHEG-3	MHEG script interchange representation
MHEG-4	MHEG registration procedure
MHEG-5	Support for base-level interactive applications
MHEG-6	Support for enhanced interactive applications
MHEG-7	Interoperability and conformance testing for MHEG-5
MHEG-8	XML notation for MHEG-5

Table 2.1: MHEG family of standards

MHEG-2 was intended to be an alternative representation of MHEG objects in SGML, but is now withdrawn. Parts 3 and 4 introduce scripting and identifier registration extensions respectively.

MHEG-5, “Support for base-level interactive applications”[14] can be considered as a specialization of part 1 which is focused on simple client-server interactive multimedia applications. It addresses limited resource terminals such as STBs. It is not strictly backwards compatible with part 1 because of some optimizations due to the restricted application domain range. MHEG-6 introduces scripting extensions (procedural code) to part 5. Part 7 addresses interoperability and conformance of part 5 applications and engines.

Recently, MHEG-8 “XML Notation for ISO/IEC 13522-5 (MHEG XML)” [15] was introduced. It provides an alternative interchange representation for MHEG-5 using XML. It defines an XML language for representing MHEG-5 information in a device independent manner.

2.3 Overview of MHEG parts 5 and 8

MHEG-5 is a specification of objects and of an interchange format, based on MHEG-1, for use in simple client-server [2] interactive multimedia applications across platforms of different types and brands [14].

In order to support a wide variety of platforms, MHEG has optional and non-standardized features. It introduces the “Application domain” concept which must be precisely defined (in a standard manner) and specializes all the abstract parts of the standard. An example of an application domain is the UK terrestrial ITV domain. Examples of application domain constraints are the set of multimedia objects supported, the transmission protocols used and the support of a “free-moving cursor”. The conformance of an MHEG application or an engine is always based on the definition of the application

[2] In this case “client-server” is used in the wide sense that there might not be a direct “request” path between them. For instance, there can be an *object carousel* based architecture in which the client waits for the transmission of the required multimedia application elements.

domain.

MHEG follows the object oriented paradigm by defining a set of classes, instantiations of which are transferred to the MHEG engine. The latter is located at the client and it processes and renders the multimedia presentation. The MHEG classes are defined in terms of their attributes, the actions that can be performed on them and the events that might be generated. An MHEG presentation is represented by an *Application* object. Each application consists of a set of scenes that control what is presented to the user. Scenes support spatio-temporal composition of presentable objects. Events and Links describe the behavior of a multimedia application. Links can specify the action(s) to be executed when a corresponding event is generated.

The MHEG-5 objects are transferred in well defined device independent encodings. The MHEG-5 standard defines two of them: *ASN.1* and the *textual notation*, while the MHEG-8 standard defines an additional XML representation. These notations give the "Author once, run everywhere" property to MHEG-5, since they are independent of implementation architectures and transmission protocols.

Content encoding schemes (e.g. MPEG for video) and transfer protocols (e.g. HTTP) are not specified and must be defined for each application domain.

2.4 MHEG-5 object model

In this section we will describe the MHEG-5 object model and the interchange representation of objects as specified by the MHEG-8 standard.

2.4.1 Introduction to MHEG objects

MHEG defines a set of *abstract* and *concrete* classes for the interactive multimedia content description. Only the concrete ones can be instantiated and represented in MHEG notations. In a typical MHEG system, clients present instantiated concrete classes sent by the MHEG server.

An MHEG class is described by its base class, its exchanged and internal attributes, the possible emitted events, its internal behaviours and the elementary actions that affect it. Using object oriented design terminology, exchanged and internal attributes correspond to public and protected attributes respectively. Similarly, internal behaviors and *elementary actions* correspond to protected and public class members respectively.

Finally, an object is able to emit *synchronous* and *asynchronous* events. Each event has a source object, a type and may have an associated data value. MHEG link objects can associate events with actions and are used to react to the emission of events. When an event is generated, all the corresponding (in terms of source, type and data) links are said to "fire" and the specified action is executed. The exact time where this execution takes place depends on the implementation of the engine. Generally, synchronous events should be executed as soon as possible while asynchronous ones may be queued. We will further examine events in Section 2.5.

2.4.2 The MHEG classes

In this section we will describe the functionality and the representation of the most important MHEG classes. The core object model hierarchy is represented in Figure 2.1, where the concrete classes (e.g. *Application*) are represented using a normal font, while abstract ones using a lighter one (e.g. *Root*). The *ElementaryAction* is not an MHEG class,[3] but it is included here in order to show its relation to the *Action* class.

The *Root* class defines the basic MHEG objects functionality. It provides object identification by its exchanged attributes *group identifier* and

[3] *ElementaryAction* is not an MHEG class since it is not described as such by the standard (e.g. description of internal behaviours, events etc). However, it is a concrete entity (i.e. it is represented in the notations) that can be part of an MHEG class description.

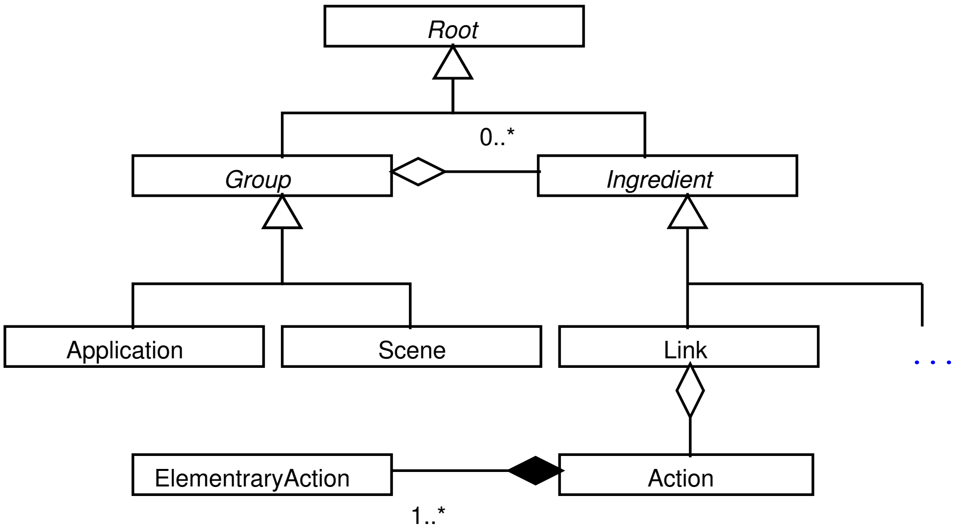


Figure 2.1: MHEG core classes

object number. The former is a string that must conform to the application domain defined encoding, where the latter is an integer (0 for group objects and non-0 for others). *Root*'s elementary actions, internal behaviours and attributes control the four possible states of an MHEG object, which are shown in Figure 2.2. Even if there is no defined encoding for the *Root* class, it introduces the `groupid` and `objnum` attributes, which are inherited by all of the sub-classes. For instance, for an *Application* object:

```
<application groupid="app.xml" objnum="0">
...
</application>
```

Most of the MHEG classes can be categorized as containers or ingredients. The former ones are the subclasses of *Group*, namely the *Application* and the *Scene*, while the latter ones are the subclasses of the *Ingredient* class. The "grouping" behaviour is represented by the *Group*'s exchanged attribute "`items`", which describes the contained ingredients, as shown in the example group representation below:

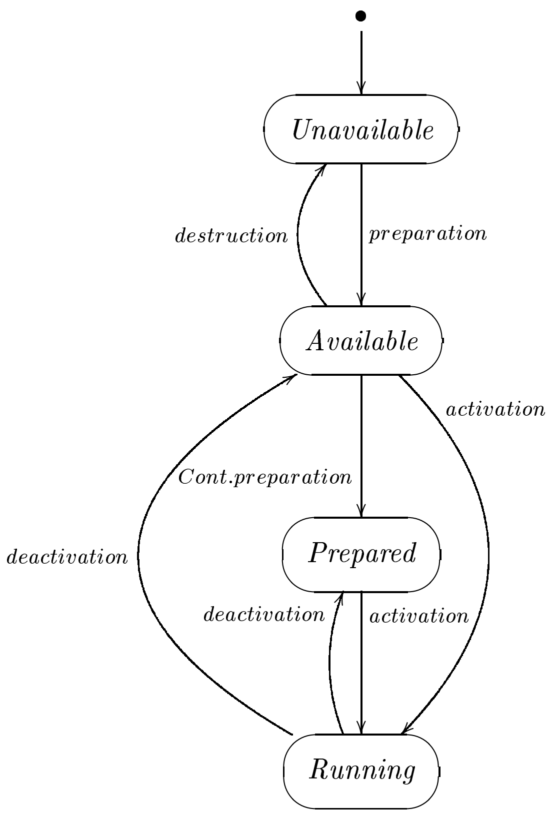


Figure 2.2: MHEG class states

.

```
<scene groupid="scene1.xml" objnum="0">
   <onstartup>
       <action>
           ...
       </action>
   </onstartup>
   <items>
       <link>
          ...
       </link>
       ...
   </items>
   ...
</scene>
```

In order to present an ingredient there must be an active *Application* and an active *Scene*. The *Application* provides the generic engine functionality, is responsible for initializing the presentation and groups the ingredients that are shared among presentation scenes. On the other hand, *Scene* allows the spatio-temporal coordination (coordinate system, timers etc) of ingredients and is responsible for the user interaction (user input events). Usually, there is an application object which launches the first scene in order to initiate the presentation, as shown in the following extract of an example in [15].

```
<application groupid="app1.xml">
  <items>
    <link objnum="1">
      <linkcondition>
        <eventsource objnum="0"/>
        <eventtype type="isrunning"/>
      </linkcondition>
      <linkeffect>
        <action>
          <transitionto>
            <objref objnum="0" groupid="scene1.xml"/>
          </transitionto>
        </action>
      </linkeffect>
    </link>
  </items>
</application>
```

Most of the MHEG classes inherit the abstract class *Ingredient* which provides the common functionality for objects that can be included in *Group* objects (e.g. if they will be shared among scenes, their content etc). Some of the most important ingredients are shown in Figure 2.3. *Variables* are used for exchanging values of different data types and are essential for the data associated with the elementary actions (e.g. parameters) and the events. Classes that inherit *Presentable* are the ones that can be "presented" (e.g. an audio clip). Similarly, classes that inherit from *Visible* have a visual representation (e.g. a bitmap) and the ones that inherit from "Interactible" can interact with the user (e.g. a push button).

The *Link* class can be considered the most important ingredient because in collaboration with the *Action* class lays the foundation for the behaviour

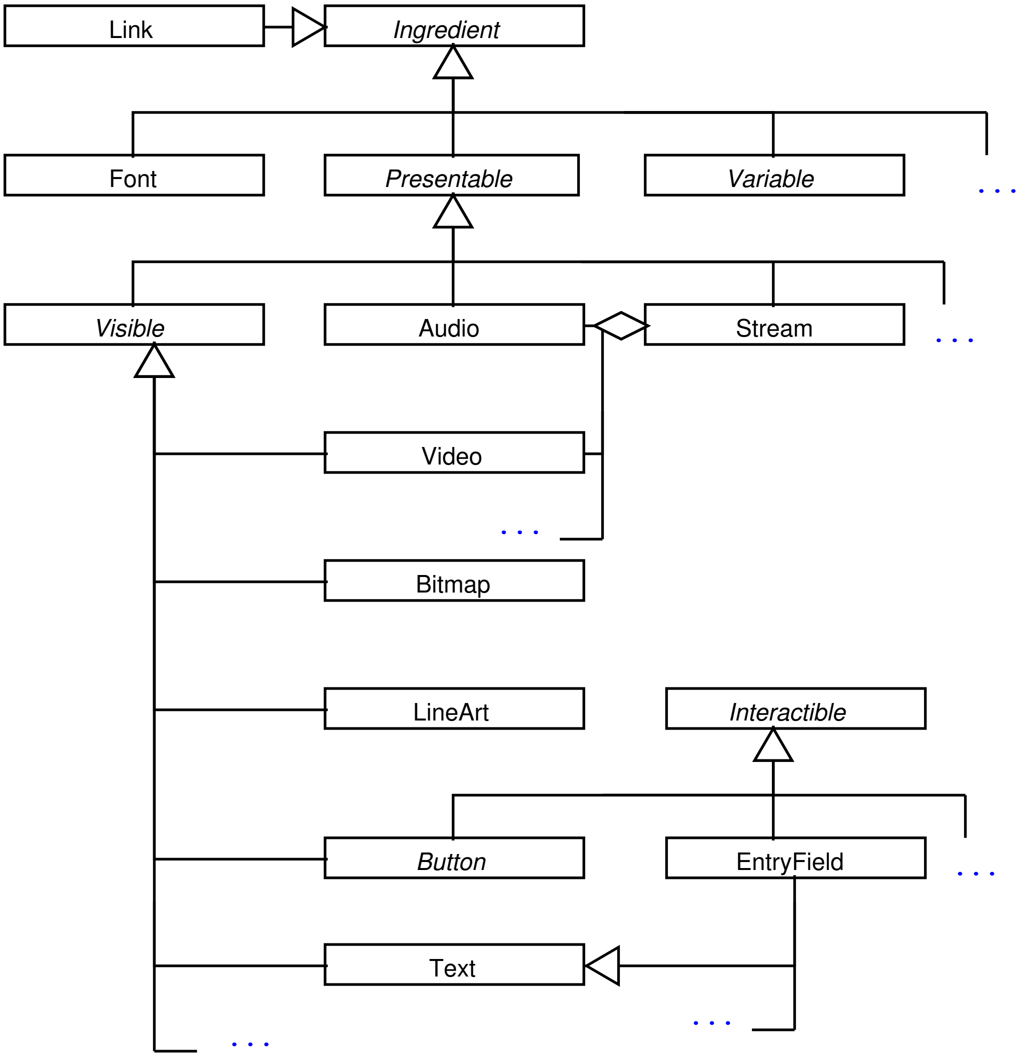


Figure 2.3: MHEG ingredient hierarchy

of an MHEG presentation. It contains a *link condition* (event source, type and associated data) and a *link effect*. When an event is emitted, the specified effects of all the active links with a matching link condition are executed. The link effect is described by an *Action* object. The representation of a simple link with a corresponding action is shown in the previous XML example.

2.5 MHEG-5 event model

We have already mentioned that objects may emit events, which in turn may cause links to "fire" and result in the execution of a sequence of elementary actions. In this section we will investigate the restrictions set by the standard and how the actual execution could proceed.

Firstly, we assume a sequential execution queue (e.g. Figure 2.4) which holds the elementary actions to be executed. When there are no actions, the engine is said to be idle. When an event is emitted, the MHEG engine must check all active (i.e. with their $RunningStatus$ = true) links condition in case it is satisfied. In that case, associated elementary actions can be "pushed" to the execution queue. The standard does not restrict the order in which simultaneously fired links should be handled, and all possible permutations are considered acceptable.

There are two types of events, synchronous and asynchronous. The first ones occur "synchronously" to the execution and must be handled as soon as possible. Asynchronous events occur "asynchronously" to the execution by timers that expire, successful content retrieval et cetera. They should also be handled in a timely manner but they are not allowed to preempt other asynchronous events (and they should be queued). However, synchronous events can preempt the execution of an asynchronous one, and must be handled immediately. Generally, synchronous events should be handled in a similar manner to procedure calls, before the execution continues to the next elementary action.

Considering the above, depending on the "code" that is executed and the event that is generated, there are 6 possible behaviours which are depicted in Table 2.2.

	Event Type	
	Synchronous	Asynchronous
Normal Execution	Execute immediately	Execute immediately
Sync. event execution	Execute immediately	Execute immediately
Async. event execution	Execute immediately	Queue

Table 2.2: Event and execution types

However, the exact behaviour of a simultaneous "firing" of a synchronous and an asynchronous event, as well as the exact time that the queued events are processed is still not clear. We could give priority to asynchronous events since their timely handling enhances accuracy (in case of timers) and improves user's interactive experience (in the case of user input events). Therefore, if a synchronous event link fires and an asynchronous event link either fires or is pending (in the queue), we will execute the asynchronous one first. Figure 2.4 is a simple execution example that illustrates how all the restrictions we have set can be put into practice.

Finally, we have to point out that there are some elementary actions that may alter normal execution flow. These are the actions which "change the context of the current action processing" [14], which are *TransitionTo*, *Launch*, *Spawn* and *Quit*. When one these happens, the elementary actions waiting in the execution queue and the pending event links that do not correspond to the new context have to be removed. For instance, if an event for an expired timer is pending and there is a transition to a scene where this timer is not visible, the event must not be handled.

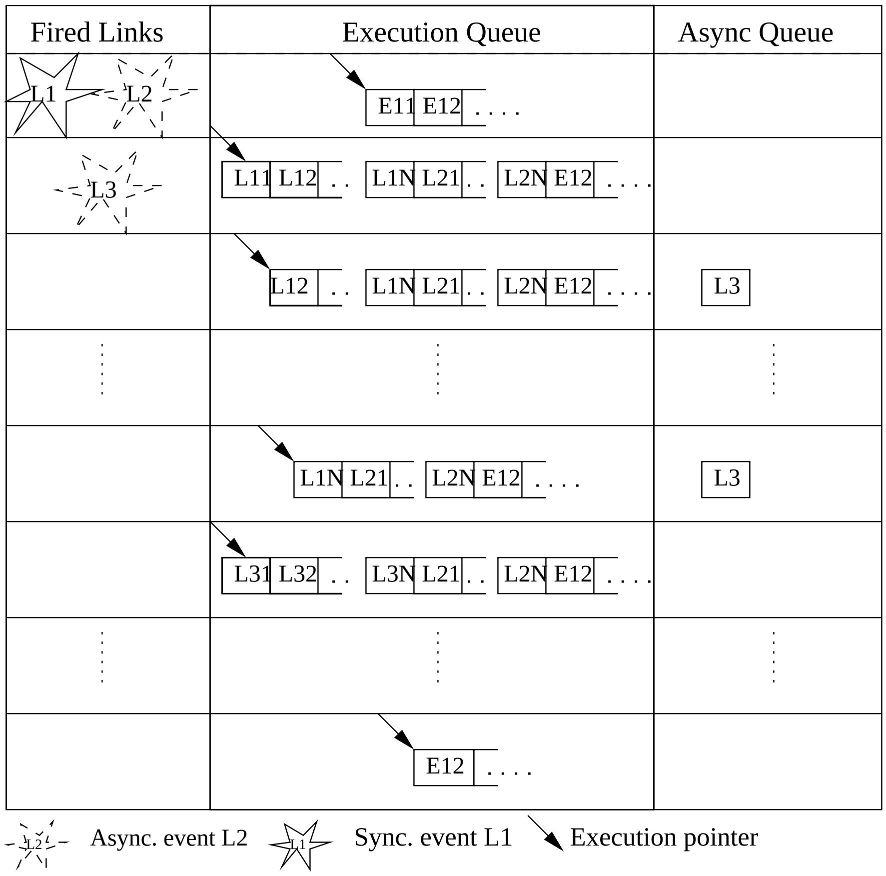


Figure 2.4: MHEG execution example

2.6 MHEG-5 conformance issues

MHEG-5 defines two types of conformance: Exchanged object conformance and MHEG engine conformance. The former is achieved when the representation of the exchanged objects is consistent with the encoding specified by the application domain. The latter is achieved when this representation is handled correctly by the MHEG engine with respect to the standards and the application domain definition.

An application domain definition should specify all the application area dependent parts that are either left unspecified or are overly abstract in the standard. Specifically it should define the:

- object exchange format
- set of supported MHEG classes
- set of optional features that will be implemented
- precise encoding formats and application specific extensions

All of these are explained in detail in section 4 of [14], however, we will give a brief description below..

The *object exchange format* specifies which of the three available encoding formats will be used for object exchange and is the basis for exchanged object conformance. In our study, it will be the XML notation described in [15].

The *set of supported MHEG classes* specifies the set of classes that have to be implemented by conforming engines. All the engines must support at least the core classes: *Root*, *Group*, *Application*, *Scene*, *Ingredient*, *Link* and *Action*. For each implemented class, all its attributes, events, internal behaviors and elementary actions which affect it should be supported.

The *set of optional features* specifies which of the optional features of the classes should be implemented by a conforming engine.

Finally, *encoding formats and extensions* deal with the specification of content encoding (e.g. the format of images), transfer protocols (e.g. HTTP), mapping of "raw" events to user input events and encoding of the group identifiers.

2.7 Conclusion

In this chapter we presented some aspects of the MHEG-5 standard with examples of the XML notation specified in MHEG-8. We have described the most important aspects of the object model, event support and how conformance is defined. In the next chapter we will investigate the other integration area, web browsers.

Chapter 3

Web browsers – DOM

After describing the ITV and MHEG standards, we will study the other part of the integration, web browsers. Firstly, we will describe the area of web browsers and how they relate to the XML and DOM standards. Then, we will provide a brief overview of XML and DOM, emphasizing on the DOM-2 event model[28] which might prove useful for the integration.

3.1 Web browsers

A web browser can be defined as "an application that provides a way to look at and interact with all the information on the World Wide Web".[1] What is meant by "...all the information ..." is not clear, but it should include the most commonly used web formats (e.g. HTML, XML, image formats etc).

Web browsers and web content are evolving rapidly since their first appearance in 1990. Initially, World Wide Web (WWW) contained only HyperText Markup Language (HTML) files and web browsers supported only the simplistic first HTML version. However, web community demands evolved and consequently a number of new formats have been used and new international standards have been developed. HTML version 4.0 is overwhelmed with new features because of the desired extended functionality.

[1] a *searchWebManagement* definition: `searchWebManagement.com`.

Moreover, the need for more attractive dynamic web pages, led browser providers to export (incompatible versions) of the browser internal document structure and event model.

In order to satisfy the demand for a more generic content language and a standard way to manipulate document structure, W3C produced (among others) the XML (eXtensible Markup Language) and DOM (Document Object Model) standards. XML and DOM will be our focus for the rest of this chapter.

3.2 XML overview

XML[30] is a standard markup meta-language, part of W3C's effort to overcome the problems of HTML and to provide a standard way to store and exchange information.

Specifically, XML is a subset of the *Standard Generalized Markup Language* (SGML) and supports a wide variety of applications by providing the means for defining new markup languages. The XML principal goals are to be a generic, easy to create and process markup language. It is generic since a new markup language can be defined for each application domain. It is easy to write since it is a text notation and can be created using a simple text editor. Finally, XML is easy (relative to SGML) to process since its syntax is concise and contains only the most important non-redundant structures of SGML.

An XML document should be *well-formed* and may be *valid*. "Well-formed" means that it follows the core syntax rules of the language (e.g. contains an XML declaration, has proper element nesting et cetera). Validity is concerned with structural and semantic correctness and relates to the optional *Document Type Declaration*(DTD). Valid XML must be well formed and must conform to the DTD. The latter defines the set of allowed elements, how they can combined and in general the form of the XML document. A simple example of a well formed XML document is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test>
<test attr="value">
        <child>Some text</child>
        <!-- A comment -->
</test>
```

XML describes the structure and not the semantics of the data. Consequently, additional information on how to represent and process the data is needed, and a number of complementary standards have been defined (e.g. XSL, XLink etc). However, they cannot offer full representation and handling information and usually a dedicated handler is needed for each XML document type.

3.3 DOM overview

The Document Object Model(DOM) is a W3C recommendation defined as "a platform- and language- neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents".[2]

The first DOM recommendation was DOM-1 which defined a platform- and language- neutral interface for manipulating XML and HTML documents structure. It was mainly used in the browser – scripting and browser – parser boundaries. The former provided a standard way of supporting dynamic web pages while the latter a standard parser interface. The current W3C recommendation is DOM-2 which consists of the 5 parts shown in Table 3.1. We are mostly interested in DOM-2 core and DOM-2 events because they are widely supported and their combination is essential for enabling dynamic document behaviour (an interactive multimedia application can be considered as a highly dynamic document).

[2]As defined by W3C at http://www.w3.org/DOM

Name	**Description**
DOM-2 Core	Dynamic document manipulation
DOM-2 Events	Generic event system
DOM-2 Views	Dynamic representation manipulation
DOM-2 Style	Dynamic style sheets manipulation
DOM-2 Traversal and Range	Traversal and content identification

Table 3.1: DOM-2 recommendations

3.3.1 DOM-2 core description

The DOM-2 core[27] defines an interface for dynamically accessing and updating the content and structure of documents. It represents the document structure using a tree-like composition of *Nodes*. Even if there is no restriction on the format of the represented document, the interface is defined in a way that fits nicely with the HTML and XML languages.

In order to achieve platform and language independence, all DOM interfaces are defined using the *Interface Definition Language* (IDL). This allows a standard interface for different DOM implementations (using the mappings between IDL and implementation languages). An important property of DOM-2 core is that the defined interfaces provide a rich set of functions for DOM tree creation and manipulation. Consequently, a typical DOM application does not need to use proprietary interfaces and will work on any standards conforming DOM implementation.[3]

All the nodes of a DOM tree implement an interface that inherits *Node* and corresponds to the document element type. An illustration of a partial *Node* hierarchy is shown in Figure 3.1. *Node* contains the generic document- and tree- node functionality by providing access methods to the type, name, value, children and parent of the node et cetera. *Document* is always the root of the tree and contains convenience and factory methods (for creating other document tree nodes). Similarly, the other interfaces contain convenience

[3]In DOM-1, there was no standard way to create the document tree, and applications had to use proprietary extensions.

and special methods that apply to the respective document element. An example of the document tree and the implemented interfaces for the XML code of Section 3.2 is shown in Figure 3.2.

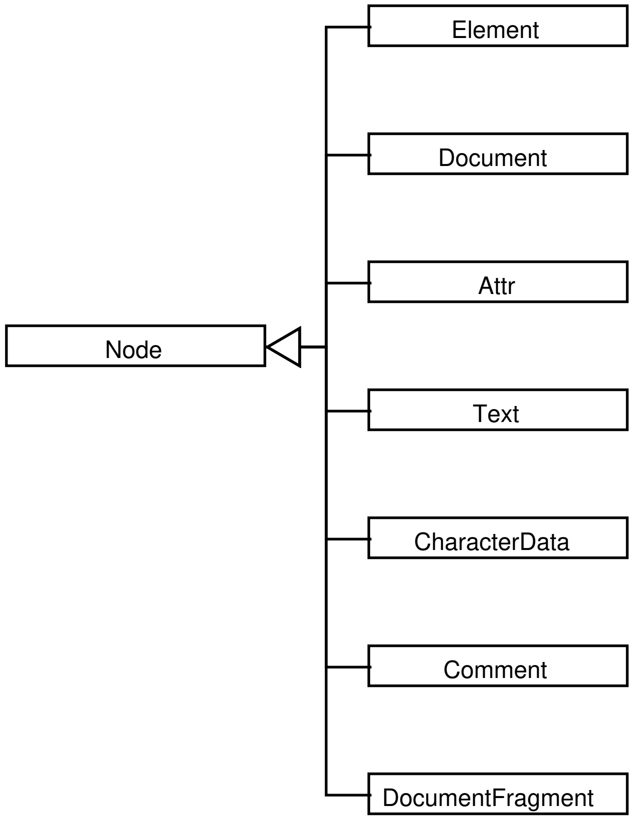


Figure 3.1: DOM core interfaces

3.3.2 DOM-2 event model description

The DOM-2 events recommendation[28] describes a standard event handling mechanism by defining the event flow in the document tree, the interfaces for event manipulation and the standard events for user interactions and document modification. It began as an effort to provide a standard subset of the proprietary event models that were used in web browsers for supporting dynamic document behaviour. The recommendation describes the event

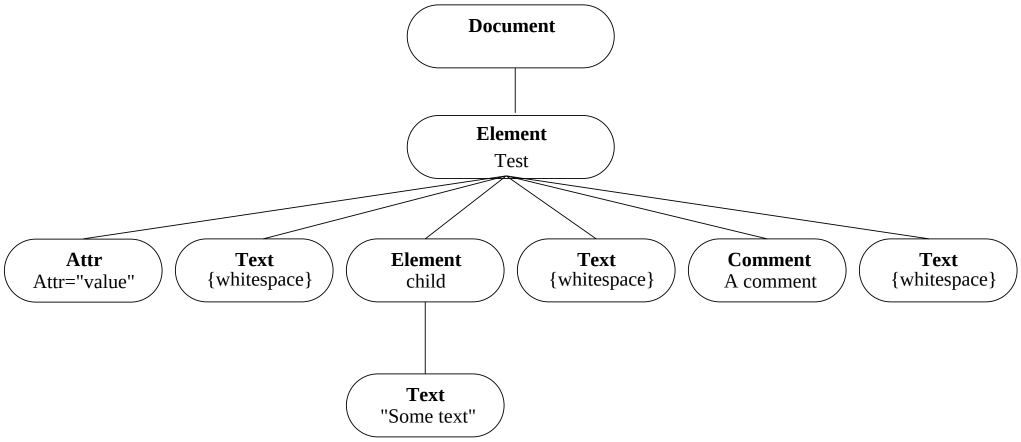


Figure 3.2: DOM tree example

flow procedure and defines a number of interfaces for event manipulation.

Each DOM-2 event has an event target, which is a node of the document tree. In a DOM implementation that supports events, all DOM tree nodes implement the *EventTarget* interface and can be used as event targets. In order to "handle" an event, an *EventListener* for that event must be registered for the appropriate event target. The most trivial case is a listener registered to the actual target of an event. Each *Node* can have more than one listeners for the same event simultaneously. In that case they are all invoked, but the exact execution sequence is not specified. Additionally, a listener can be registered to an "ancestor" of the event's target and handle the event during the *capture* or the *bubbling* stage.

A DOM-event can be handled in three different stages of its propagation through the DOM-tree. As we have already mentioned it can be handled when it has reached the event target by event listeners registered to that node. Moreover, *event capturing* allows handling of events that propagate from the document root to the event target. Similarly, *event bubbling* can be used to handle an event when it propagates upwards from the event target to the document root. Event capture is considered an event listener property and is defined when the listener is registered to the *EventTarget* (and, consequently, all the events flow from the document root to the event target in order to activate any capturing listeners). On the contrary, event bubbling is considered as an event property and is defined as part of the *Event* interface (and therefore, an event "bubbles" only if it is declared as "bubbling").

An event flow example is illustrated in Figure 3.3. The event is targeted to the text node and its propagation through the DOM tree is denoted by the dashed arrows. The diagram shows where and when capturing listeners, event target listeners and non-capturing ancestor listeners (for a bubbling event) are activated. This form of hierarchical event flow is especially useful when document structure corresponds to the spatial representation structure. In that case, event capture and bubbling simulates the actual graphical environments event flow. For instance, a "click" on a push-button is also a "click" in the region of the parent window.

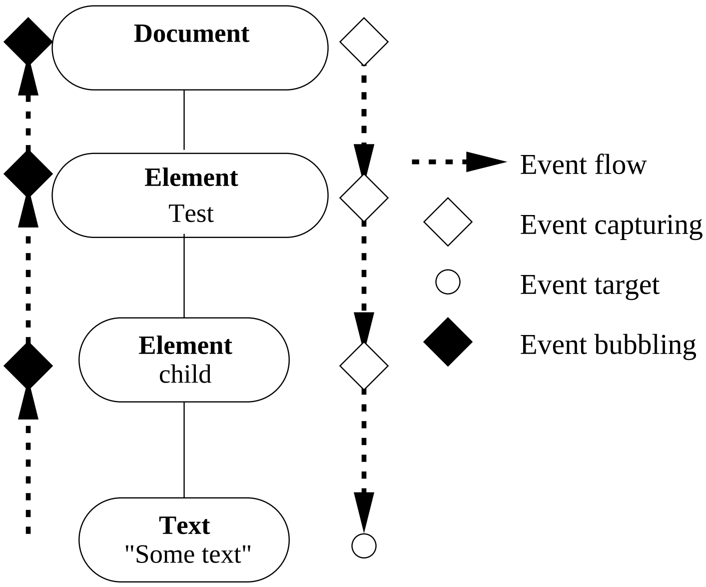


Figure 3.3: Event flow example

The DOM event model is extensible since there are no restrictions for the generated event types. However, a special set of events have been defined to support general graphical applications and HTML. The HTML specific events are widely used in combination with *JavaScript* for enhancing HTML web pages.

3.4 Conclusion

In this chapter we have briefly described the web-browser area, and gave an overview of XML and DOM-2. These have laid the foundation for our browser assessment and browser architecture discussion of the next chapter.

Chapter 4

Browsers assessment – X-Smiles

After studying the basic web and ITV technologies, it is important to select an appropriate browser platform for our experiments. Firstly, we will specify the browser requirements and study a number of alternatives. Afterwards, we will explore the architecture of the most promising one and comment on how it can be extended.

4.1 Browsers assessment

In this section we will describe the assessment process and the selection of a suitable browser to integrate MHEG with. This is of major importance because there is no commonly accepted browser architecture and, since we are interested in browser modification, we cannot rely solely on international standards. Consequently, the results of this section will significantly influence the rest of our study.

4.1.1 Browser requirements

Browser requirements can be divided in two categories: mandatory and optional ones. Mandatory requirements are those that are fundamental for integrating the MHEG functionality. Optional requirements are those that either are of secondary importance or are used as simple indicators of useful

Browser Requirements	
Mandatory	Available source code Object oriented implementation language Process, graphics and networking libraries Extensible user interface and rendering machine Support for basic media types XML and DOM support
Optional	Well documented modular object oriented design Support of XML related standards

Table 4.1: Summary of browser requirements

features.[1]

The requirements we have set are shown in Table 4.1. The very first requirement is the availability of the code. Since we will modify the browser's code, we need to have access to it. The second requirement has been set because the integration of the MHEG object-based architecture will be easier if the browser is developed in an object oriented (OO) language.

In order to avoid a platform dependent extension, we also require browser provided generic libraries for process management, graphics and networking. This requirement applies only when these are not supplied by the implementation language (e.g. Java).

The fourth mandatory requirement is a prerequisite for allowing an interactive multimedia presentation. The user interface should be highly customizable in order to achieve "interactiveness", and the rendering machine should be able to incorporate multimedia extensions. Additionally (5th requirement) basic media types (e.g. several image formats, audio etc) should be supported in order to straightforwardly present multimedia content.

The final mandatory requirement is XML and DOM awareness. XML awareness is essential since we are interested in MHEG-8 representation

[1] Since the project time table doesn't allow full investigation of all the alternatives' architecture, we use the optional features in order to get a "fast" insight into the browser properties that cannot be thoroughly investigated

which is defined as an XML meta-language. The DOM provides a standard foundation for dynamic internal document structures and will be of great help for MHEG support.

As we have already mentioned, web browsers have become fairly complex and difficult to handle applications. Therefore, it would be helpful if the selected implementation is well designed and well documented. Moreover, XML related standards support is a good indication for the extensibility of the browser. It should be easier to integrate a new XML language to a browser if it already supports a number of other XML applications and is designed with extensibility in mind.

4.1.2 Browser alternatives

We have taken into consideration six browsers which have their source code available: *Mozilla*[20], *X-Smiles*[32], *Amaya*[24], *HotJava*[22], *Arena*[1] and *Mosaic*[21].

In order to avoid a full investigation of the browsers, we created a "qualification" test in which we study a specific platform until we find that it does not comply to one of the mandatory requirements. Table 4.2 shows the results of the test, where only X-Smiles qualifies and also fulfills both of the optional requirements. Consequently, it seems to be the most appropriate platform for our study. The rest of the chapter will be devoted to the X-Smiles browser and its architecture. A brief description of the aforementioned browsers that justifies the results of Table 4.2 is included in Appendix D.

4.2 X-Smiles overview

In this section we will give an overview of the X-Smiles architecture. We will firstly describe the top-level design and then provide more information for each functional layer.

	Mozilla	X-Smiles	Amaya	HotJava	Arena	Mosaic
Open source	√	√	√	×	√	**A**
Non Obsolete	√	√	√		×	×
OO Implementation	√	√	×			
Networking, process and graphics libraries	√	√				
Extensible UI	√	√				
Media	×	√				
XML – DOM aware		√				
	Qualification Line					
OO design		√				
XML related standards		√				

A: Almost open source. It currently has no open source license. However, the source can be obtained and modified under some restrictions.

Table 4.2: Browser qualification information

4.2.1 Architecture overview

X-Smiles is composed out of three layers as shown in Figure 4.1. The "XML processing" layer is concerned with XML file processing, the "Browser core" ties everything together and contains core components like the event and MLFC (Markup Language Functionality Component) manager. The "User interface and interaction" layer consists of the browser user interface and the MLFCs.

In order to display a document, X-Smiles has to locate and activate the *primary MLFC*. The latter is the *MLFC* that handles this type of document (the document type is specified by the XML `DOCTYPE` declaration). MLFCs are responsible for the semantics analysis of the document, the presentation and the user interaction. The primary *MLFC* may use secondary MLFCs in order to display additional types of documents.

Before passing document information to the primary MLFC, X-Smiles parses the document and applies the (optionally) specified XSL transformation. This process is handled in the two lower layers and is illustrated in

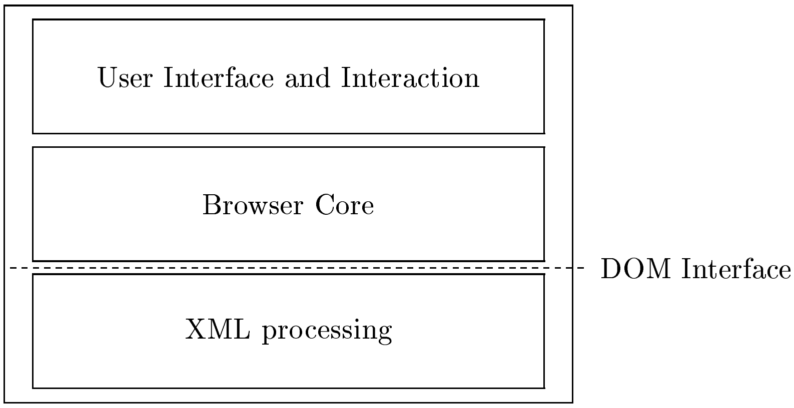


Figure 4.1: Top level browser architecture (based on [13])

Figure 4.2. The document text is firstly converted to a DOM tree. If there is an associated extensible stylesheet language(XSL) file, it is retrieved, converted to a DOM tree and combined with the document tree. The result is passed to the primary MLFC.

The most important components of X-Smiles are the MLFC manager and the event manager. The former, handles MLFCs while the later is responsible for dispatching internal events to all the interested parties. Events are, among others, used to inform browser components about the browser state. X-Smiles is always in one of 6 distinct states which are illustrated in Figure 4.3. Solid arrows show the usual transitions, while dashed arrows represent error conditions.

4.2.2 XML processing layer

The XML processing layer is responsible for the XML document parsing, DOM tree construction and XSL – Transformations (XSL-T)[26] application. X-Smiles can adapt to different external parsers, and currently *Xerces*[2] and *Docuverse* XML parsers are supported. The XSL-T engine can be customized in a similar way. The selection of the actual engines for parsing and

[2]Xerces parser belongs to the XML Apache project: `http://xml.apache.org`

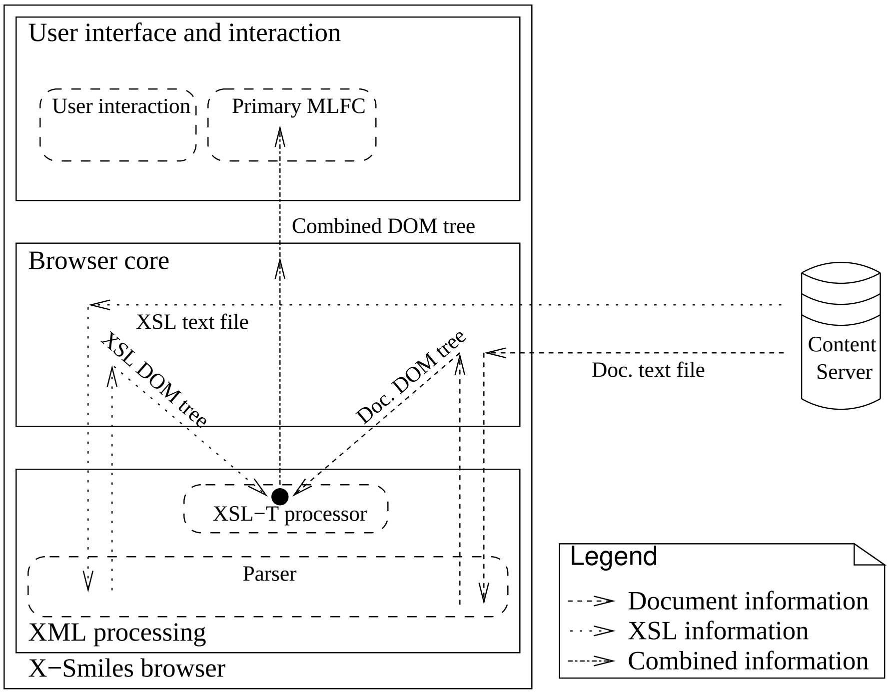


Figure 4.2: X-Smiles document flow example

XSL-T are chosen by the browser core and are determined by the X-Smiles configuration file.

4.2.3 Browser core layer

The browser core layer ties the browser components together and is responsible for the event mechanism, MLFC management and browser-wide shared information (e.g. the browser state).

The MLFC handling is based on the MLFC manager, which activates, deactivates and keeps track of MLFCs, attaches them to the event manager, throws MLFC specific events and handles the MLFC loader. There are two types of MLFCs: active and passive. Active are the ones used for the presented document and receive browser events while passive are the ones that are not used for that type of document.

The X-Smiles event model is based on a simple broadcasting object,

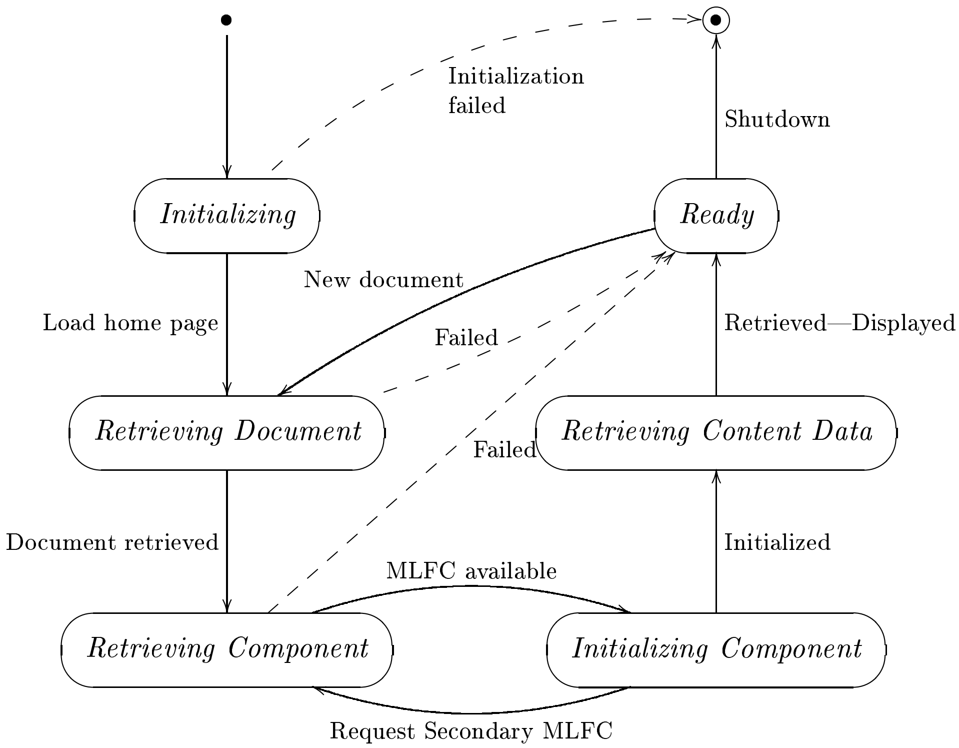


Figure 4.3: X-Smiles state model (based on the state model figure in [13])

the "event broker". When an event is sent, it is broadcasted to all the components that have registered to the event broker. Events are generated by event-specific functions which are part of the event broker and the event listener interface. The event model is overly simplistic and not extensible (because events are hard-coded) and we will therefore avoid using it for the MHEG integration process.

4.2.4 User interface and interaction layer

The upper layer of X-Smiles contains the MLFCs and the generic user interface functionality. We will not discuss the latter since we are only interested in the document contents representation, and not in the browser appearance.

MLFCs are responsible for the semantic analysis of the document DOM tree and the presentation of the information. The primary MLFC is given full control over the browser content window, so it is possible to represent the content and interact with the user independently of the other browser components.

There is a specific MLFC for each document type and two generic ones for the source and tree document views. The content specific MLFCs are independently designed according to the requirements of the corresponding document type. However, there is a generic shared MLFC functionality which is described below.

Firstly, there is the initialization phase in which the MLFC has to allocate required resources and initialize its internal variables. This happens immediately after the MLFC loader locates the MLFC. At this point the source document is not known to the MLFC. Next, we have the analysis of the document which might be combined with resource allocation and retrieval of content data. The MLFC examines the document source tree and builds the required internal structures for the presentation of the document. After the completion of this stage, the document is presented to the user and the MLFC handles the possible user interaction. Finally (whenever the component has to be removed), the MLFC must free all the allocated resources and revert to its initial state.

MHEG functionality should normally be included in an additional MLFC which will be associated with the document type "`mheg5`". Therefore, MHEG functionality will be a part of the user interface and interaction layer (similarly to all other MLFCs). When X-Smiles encounters an MHEG-8 file, it would pass the document DOM tree to the MHEG MLFC which in turn will render the presentation.

4.3 Conclusion

In the first part of this chapter, we described the browser requirements for our research and selected the most promising browser platform among six alternatives. The second part consisted of the description of X-Smiles and its architecture.

Based on the established foundation of the previous chapters and the results of this one we can now proceed to describe the actual integration part of the project.

Chapter 5

ITV – web browsers integration

Having established the required information on web browsers and ITV we will proceed to the integration of these two areas. Specifically, we will investigate the design and implementation of an MHEG extension for X-Smiles.

Section 5.1 describes the incremental process of the integration. Subsequent sections are devoted to the design, implementation and evaluation of the integration steps.

5.1 The integration process

Even if MHEG-5 is designed with simplicity in mind, a full featured engine implementation would be too time consuming for the available project time. Consequently, we will divide the integration process into a number of incremental steps. In the first one, we will lay the foundation for further research by focusing on a simple design and implementation of the core MHEG aspects. Further steps will build upon it in order to either illustrate an integration concept or to improve the engine functionality. We have designed and implemented the first two steps.

The first one is the "minimal conforming MHEG engine". Our goal will be to investigate how the core functionality of an MHEG engine can be integrated in a web browser. In order to accomplish this, we will define a restrictive application domain containing only the absolutely necessary elements for a working MHEG engine. We will not consider any presentables or optional MHEG features (e.g. caching, connections etc) and we will focus on the core aspects such as events, actions, fundamental classes et cetera. At this stage we will be able to study the integration of basic MHEG functionality, the MHEG event model and the processing and applicability of the MHEG-8 notation.

The second step is the "event models integration". The MHEG event model is based on the generated events, the links and the MHEG objects (as event sources). DOM-2 event model is based on implementation emitted events which are targeted to document nodes. Even if the two models are different, MHEG events and event handling concepts can be mapped to the DOM-2 event model. We will study this mapping, how it might be implemented and what are the benefits of such an implementation. Since the DOM-2 event model is a standard which is supported by most browsers (including X-Smiles) we expect to end up with an easier and more effective way to implement the MHEG event model using existing browser features.

The following sections are devoted to these two steps and their in-depth analysis. Further integration and functionality extensions will be described in the next chapter.

5.2 First step: minimal conforming engine

This section describes the first step of the integration, the "minimal conforming engine" which is based on an engine that supports only the mandatory MHEG features. We will study what has to be implemented, an implementation example and its evaluation.

5.2.1 Design

As we have already mentioned in Section 2.6, conformance is always defined in relation to an application domain. We have defined a minimal application domain that contains only the mandatory MHEG features and is described in Appendix C (page 93). According to this, in order to achieve conformance, the following functionality must be implemented:

- An internal object model representing the MHEG classes
- The object referencing mechanism
- The execution queue and the event handling
- The parsing of MHEG-8 documents and construction of the respective MHEG objects.

As far as the internal object model is concerned, we do not have to implement all the classes' functionality since some of the class features are optional[14]. Table 5.1 summarizes what has to be implemented for each of the minimal engine MHEG classes.

Class name	**Comments**
Root	Everything
Group	Everything except caching.
Ingredient	Content hook and original content are ignored.
Application	Defaults and locking are ignored.
Scene	Free moving cursor and next scenes are ignored
Link	Everything
Action	Everything
Variable	Everything
Variable sub-classes	Everything

Table 5.1: Minimal application domain classes features

Based on the above description, we can identify the *elementary actions* that have to be supported. The resulting set contains twenty four elementary actions which affect the implemented MHEG classes and do not relate to the specified optional features.

Internal object model

MHEG-5 applications are based on a collection of objects which interoperate in order to create a multimedia presentation and are described in terms of attributes, events, behaviours and the elementary actions that affect them (Section 2.4). An object's state consists of the values of internal and external attributes which must be stored. A straightforward approach is to represent MHEG objects as actual engine objects. The form of these objects and the way they are combined will be called the "*internal object model*". Below, we will investigate how such a model can be structured.

As far as the inheritance structure and the type of classes are concerned the mapping is trivial. MHEG classes inheritance will be represented by implementation classes inheritance and MHEG abstract and concrete classes will be represented as such in the internal object model. Consequently, we will have an implementation class hierarchy similar to the MHEG class hierarchy.

The MHEG attributes represent the object's state and can be implemented as attributes of the corresponding class. Exchanged attributes are used for object initialization and may be accessed by other classes. Therefore they can be implemented either as public attributes or by using member access functions. Internal MHEG attributes are accessed either internally or by sub-classes so they might be implemented as "protected" attributes. Similarly, internal behaviours manipulate internal attributes and are accessed the same way, therefore, they can be implemented as "protected" member functions. The interface of an MHEG class is based on the elementary actions that affect it. Consequently, we could represent them as public member functions.

The elementary actions interface design depends on the part of the engine that object dereferencing takes place, which can either be inside or outside the elementary action functions. In the first case the elementary actions interface will consist of MHEG references. In the latter case it will contain implementation dependent references. Since, we would like to separate action functionality from referencing mechanisms and since action's target has to be resolved before calling the respective member, we will choose the second solution.

Based on the arguments presented above we can derive a mapping of the MHEG object model to object oriented implementation concepts which is illustrated in Table 5.2. A simple example of this mapping is illustrated in Figure 5.1 where a partial definition of MHEG classes *Group* and *Application* are mapped to a class diagram.

MHEG term	**Object oriented term**
abstract class	abstract class
concrete class	concrete class
inheritance relationship	inheritance relationship
exchanged attribute	public attribute or protected attribute + public utility functions
internal attribute	protected attribute
internal behaviour	protected member function
elementary action	public member function

Table 5.2: MHEG - OO model mapping

Object referencing – handling

MHEG standard defines the notion of *Application namespace*, which identifies the set of an application's accessible objects. Referencing is based on the *object reference* which contains a group identifier and an object number. The group identifier is a string, that helps to locate a group object. The object number indexes an ingredient within the group (or the group object itself).

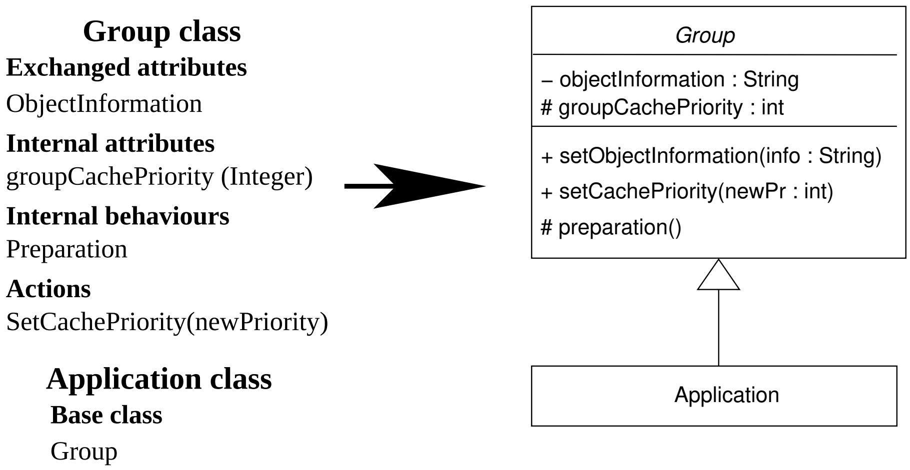


Figure 5.1: MHEG – OO mapping example

Object referencing should be implemented by the active application because the latter defines the application namespace. If the referenced object is already loaded (e.g. it belongs to the active scene or application) an implementation reference[1] can be derived. If the object is not loaded (e.g. a new scene) the application has to locate it and, if the reference is valid, create a new implementation instance. This process will most probably include the parsing of a new MHEG file.

The MHEG-5 standard[14] states that a reference should be resolved only when necessary. For instance, for an elementary action, the target object reference and possible parameter references have to be resolved immediately before the execution of the elementary action. This ensures correct handling of indirect references but requires storing of most of the source file information (since, reference information cannot be resolved during parsing). This can be achieved by either processing the source file during execution or by introducing appropriate structures that represent the required information (e.g. structures for representing object references, indirect references etc).

[1]As opposed to an MHEG object reference. For instance, it could be a Java reference or a C pointer.

The MHEG error-ignoring behaviour allows use of both of the solutions. However we will follow the second one since it allows easier processing of the actions, better error handling and separation of the parsing and execution steps.

Execution

MHEG presentation execution is the handling of a sequence of elementary actions as indicated by fired links and *Action* class attributes. In order to describe this process we have to derive a way to store and handle actions and events.

An *Action* can be represented as a sequence of elementary actions. Elementary actions are not specified as MHEG classes (Section 2.4.2) and there is a lack of guidance within the standard[14] on the most adequate implementation. Since, we have decided to store elementary actions information using internal structures, and because we need a simple and consistent way to call the respective public members of MHEG classes, representing each elementary action using an internal class is a good solution. For instance, we can have an abstract root class *ElementaryAction* and a subclass for each elementary action. Each subclass will hold information on the parameters of the action and will be responsible for resolving object references during execution and for invoking the appropriate method of the target MHEG object. This approach provides a simple way of handling elementary actions and separates the concepts of parsing, syntax checking and execution.

Elementary actions are always executed sequentially because of either a "fired" link or a well specified object condition (e.g. the *onStartUp* attribute). MHEG-5 does not specify any relation between these two, so the latter can be executed when the standard–specified condition arise while the former can be executed as specified by the event model below.

In order to handle MHEG events, an event model like the one described in Section 2.5 is needed. There will be a synchronous and an asynchronous event queue where corresponding fired events will be inserted. After each elementary action's execution, it is checked if there are any fired links that must be executed, and in that case, the corresponding elementary actions are inserted in the execution queue.

However, if this design is directly implemented, it is difficult to mark the asynchronous event boundaries (i.e. when an asynchronous event is executed) and to handle event priority as expected. For instance, lets assume that a synchronous (S_1) and an asynchronous (A_1) event fire simultaneously. If we give priority to asynchronous events, elementary actions of A_1 will be pushed to the execution queue. However, after the execution of the first elementary action, S_1 will still be pending and therefore executed immediately. Consequently, we will have an interleaving of link execution where the synchronous event finishes before the asynchronous one.

A simple solution is to introduce the concept of a stack, where each handled event introduces a new level of execution. Each execution level can contain information on the elementary actions and the pending asynchronous events as well as information on the state of execution (e.g. asynchronous or not). This approach can be implemented by using a stack of pairs of queues (for per level execution and synchronous event queue) and an asynchronous event queue. Each time an event is handled, a new level will be created. A simple example of this concept is illustrated in Figure 5.2. We have to point out that even if the simple model of the previous paragraph could be used for a conforming engine, the one specified here seems to be closer to the intentions of the standard.

Finally, as far as "firing" of links is concerned we just have to check all *active* links when an event is emitted. If a link "fires", the corresponding event is placed in either the asynchronous or the top synchronous queue.

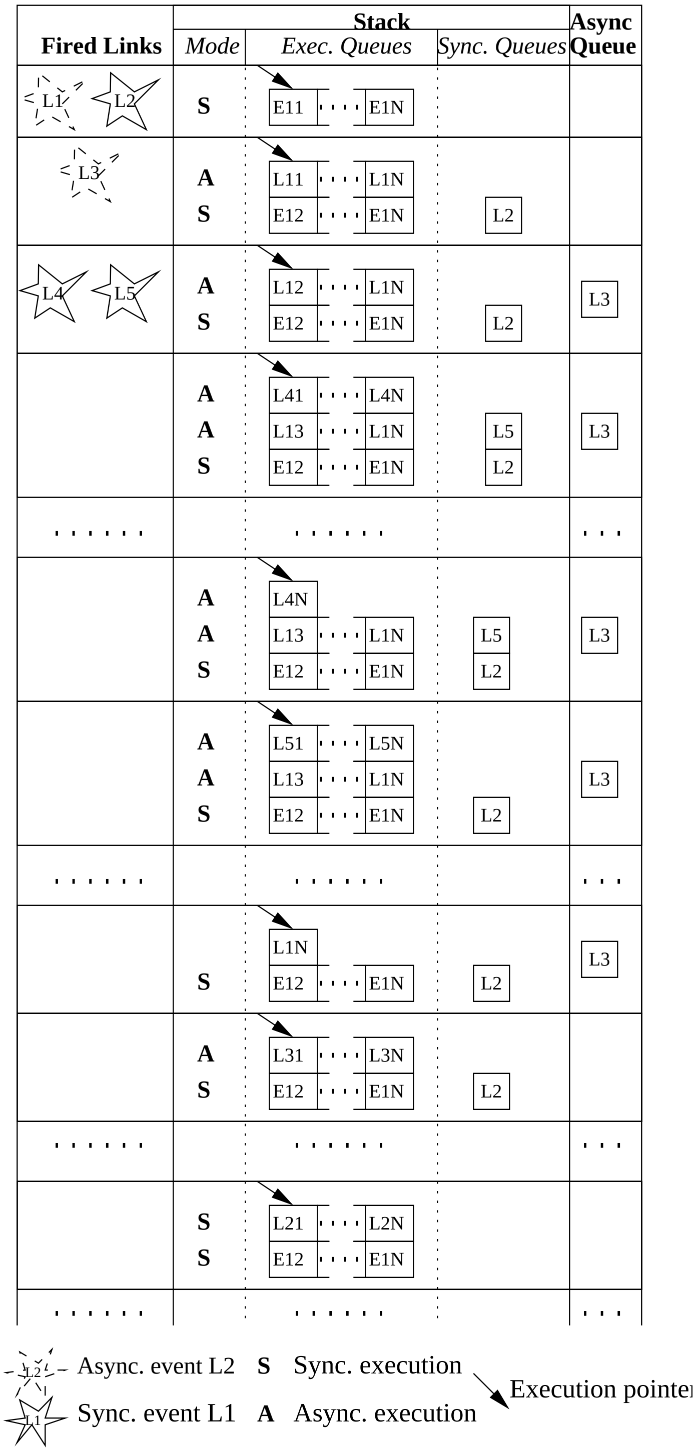


Figure 5.2: Stack event model example

Parsing

MHEG-8 representation parsing is handled by the XML parser and the engine which has to process the resulting DOM tree. Since, MHEG-8 objects representation corresponds to the MHEG class hierarchy (e.g. subclasses simply add more elements to the ones of the respective superclass), object parsing should have a hierarchical form as well. This can be achieved by a parsing class hierarchy similar to the MHEG one. However, there is no need to adopt this approach since a more "compact" solution might be used, which represents each class by a parsing function. Each function that corresponds to a subclass will always call the superclass function first. MHEG objects will only be created by functions that correspond to concrete MHEG classes.

Error checking does not have to be performed during the initial object construction (because of the error-ignoring behaviour of MHEG, the engine is not expected to produce error messages). However, the sooner the checks are performed, the better debugging information and internal object model consistency we have.

5.2.2 Implementation

The implementation is a straightforward application of the concepts in the previous section. Since, there is no user interaction (no presentables or interactibles are implemented) the only browser specific code will relate to the MHEG MLFC handling and the parser and DOM-tree access. In this section, we will describe the implementation of the first MHEG extension to X-Smiles.

The engine functionality is divided into 5 parts as illustrated in Table 5.3; the rest of the implementation section will describe each of those parts.

The *Object model* contains a class for each supported MHEG class (as defined in Section 5.2.1), where internal behaviours, internal and exchanged

Component	Description
Object model	Class hierarchy for the representation of the internal object model.
Elementary actions	Implementation of the concrete subclasses for elementary actions.
Execution handling	Event and execution handling.
Referencing	Representation of MHEG object references and indirect reference types.
Core	Core functionality of the engine (includes parser, engine manager, error handler etc).

Table 5.3: MHEG engine components

attributes and the elementary actions interface are implemented. Class hierarchy is similar to the MHEG standard hierarchy and and a partial illustration (excluding the variable classes) is shown in Figure 5.3. The implementation follows the guidelines of Section 5.2.1.

The elementary actions part contains the subclasses of the abstract *MHEGElementaryAction* class. They contain all the parameter information specified in the source file (e.g. target object, parameters etc) and provide a *run()* function which is responsible for resolving all the references and calling the appropriate member function of the target MHEG object. The source file information is kept using variables of the *referencing* classes which represent object references, indirect references, generic objects et cetera. Resolving is performed in cooperation with the active application at the time the action is run.

Execution handling consists of the event handling and the execution of actions specified as class attributes. The latter functionality is provided by the *run* function of the elementary actions. The former is controlled by the *MHEGProcessor* class which implements the model described and illustrated in Section 5.2.1 (and in Figure 5.2, page 51).

The stack and queues handling is internal to the *MHEGProcessor*. The latter provides public functions for generating events and "running" the

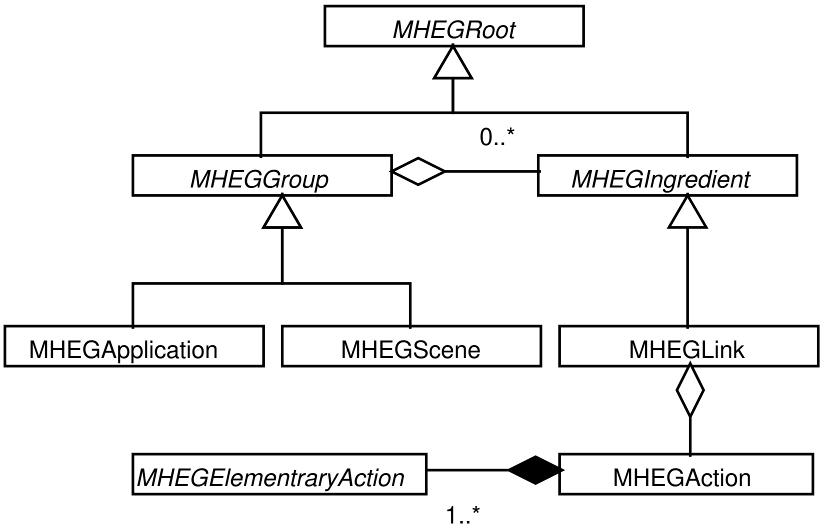


Figure 5.3: Engine MHEG class hierarchy

application. When the *MHEGProcessor* is "run", it terminates only when the presentation is over. We have to point out that since asynchronous events handling requires a multi-threaded architecture, *MHEGProcessor* must be carefully designed in order to be thread safe. The overall model is shown in Figure 5.4.

The object referencing part of the implementation contains structures for representing every kind of object reference or generic object type (e.g. a generic integer which can either be a hard coded integer or a reference to an integer variable). A partial class diagram is shown in Figure 5.5. *RefInterface* provides the functionality of the MHEG-5 "generic object reference". The "generic" classes correspond to the respective MHEG-5 entities which are mainly used for storing elementary actions' parameter information. The *ObjRef* class is used in every MHEG object for identification, and hash-code functions are provided for efficient indexing of object references.

Finally, the core component contains the link processor, the parser, the error handler, and the engine manager. Engine manager (class *MHEGMan-*

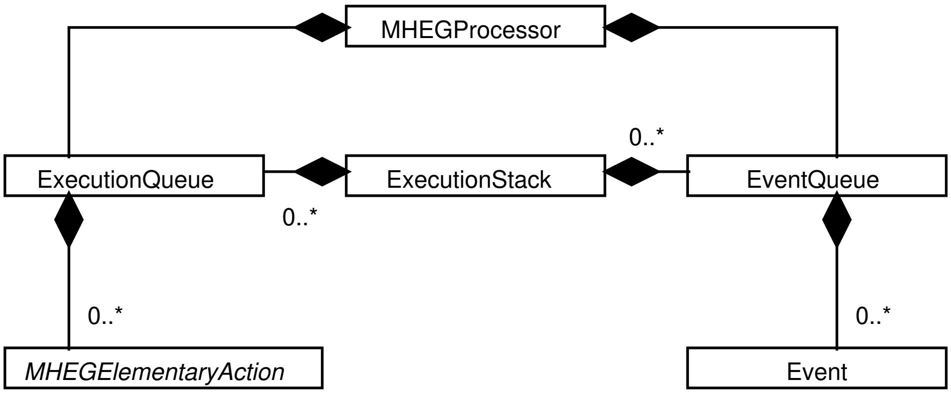


Figure 5.4: Engine execution hierarchy

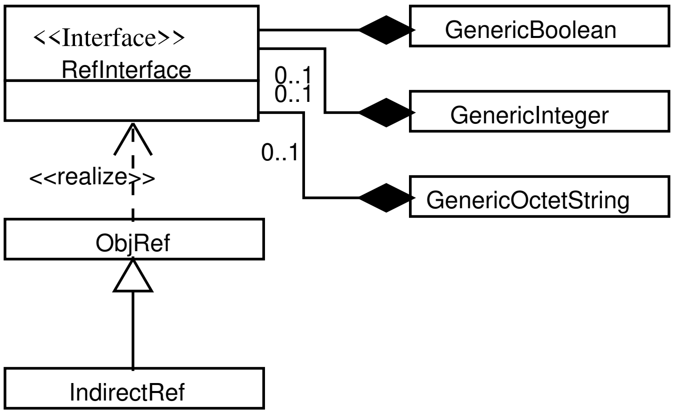


Figure 5.5: Engine reference class hierarchy

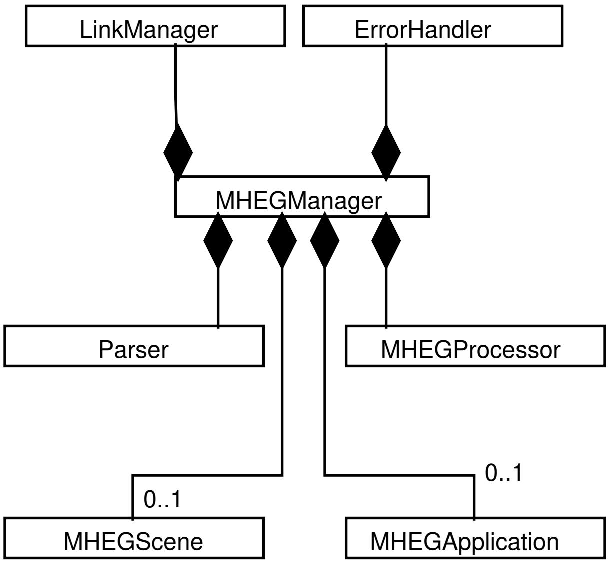


Figure 5.6: Engine manager class

ager) is responsible for linking and managing all the other engine parts and for initialization and destruction of the engine. Inter-part communication occurs through the engine manager and all engine components contain a reference to the manager. Figure 5.6 illustrates its relationship to the other components.

The link processor keeps track of the active links and handles fired links. It provides functions for event emission which notify the *MHEGProcessor* for any fired links. The parser is used for processing the DOM tree of the source file and build the internal object model. It works with a hierarchy of functions as described in Section 5.2.1. Finally, the error handler is mainly used for debugging purposes (since an MHEG engine simply ignores any errors occurred) and is responsible for the message output.

5.2.3 Evaluation

In this section we will evaluate the first step of the integration by studying a simple execution example and commenting on the implementation.

Since there are no presentables, the engine's function can only be monitored using the output debug messages. The implementation offers two interfaces: one that outputs all debug information to the console and one that uses an X-Smiles window to produce a more structured presentation of the execution trace. We will use the first variant which is more appropriate for our description.

A simple example consists of an application and a scene XML file. We will construct an MHEG application which, when activated, makes a transition to the specified scene. At this point, the scene will set the value of an integer variable to 10000, wait for this long (in milliseconds) and then quit. This simple example allows illustration of the basic MHEG objects, synchronous and asynchronous events and several forms of elementary action execution. The XML sources of the two examples are included in Appendix E.

The processing of these files by our implementation works as expected. Firstly, the application definition and the internal object model is constructed. The latter, precisely represents the specified application in addition to the default values specified by the MHEG-5 standard. Then, the application starts and the application object is activated (as well as its ingredients). The *isRunning* event is generated, and the corresponding link "fires". The link's action contains the *transitionTo* elementary actions which is successfully executed by parsing the scene file, and activating the scene. The first scene's link fires, the MHEG variable is set to "10000", and a timer is set for that amount of time. The timer fires after exactly 10 seconds, which means that the variable was set successfully, and executes a *Quit* elementary action which leads to the destruction of all the MHEG objects and to application termination. The detailed output of the MHEG engine that illustrates all these steps is included in Appendix E.

Even if our example shows that the engine works as expected, there are some outstanding issues. Firstly, the application object is not designed to load referenced objects which are not already loaded. As consequence the *TransitionTo* elementary action cannot be executed as specified by the standard, since, we would have to call a method of an unloaded object. As a consequence, "transitional" actions, are provided by the active application. However, the external interface to the engine remains the same, but there is no strict conformance.

Moreover, the parser design didn't prove to be the best solution. For each new parsed entity the same kind of checks and translation must be repeated. Probably, if we could find an association among the XML representation and the process of object creation we could provide a generic, better designed parser functionality.

Similar problems were introduced by the elementary actions design decision. Even if representing each elementary action as a separate class seems like a good design paradigm, it proves problematic during the implementation because of the number of the elementary actions. Probably a simpler approach (similar to the parser) should have been followed.

Finally, the code for this simple core engine proved to be lengthier than expected. Therefore, in the next section we will investigate how we can integrate MHEG and DOM event models in order to provide an easier and more compact solution for the event handling component of the engine.

5.3 Second step: Event model integration

After implementing the core MHEG engine, we can experiment on how to use additional existing browser features to support the MHEG functionality. The second part of the integration is concerned with the DOM and MHEG event models integration. Specifically, we will try to map MHEG events functionality to the DOM event model in order to achieve a more compact implementation.

5.3.1 Event models comparison

Firstly, it is important to compare the two models in order identify which MHEG event features are directly supported (by the DOM event model) and which are not. Below, we will describe how similar concepts are defined and handled for each model.

An MHEG event is generated by an MHEG object (either because of an internal behaviour or an elementary action) and has a source object, a type and an optional associated value. The source object is the MHEG object that emits the event; the type is one of the predefined MHEG event types, and the associated value might be an integer, a boolean or a string (depending on the type of the event). A DOM event is generated by the implementation and has a target, a type and might include additional information. The target is a DOM node, the type is a string and the additional information can be stored in attributes of the event object.

Event flow is different between the two models. Specifically, in MHEG there is no event flow. When an event is generated by the source object it simply causes the corresponding active links to fire. Further processing is specified by the actions of the links. On the other hand, DOM has a well defined event flow, where an event "flows" from the document root to the specified event target and might (if it is a bubbling event) flow upwards to the root again. Therefore, event handlers might be activated by events targeted to nodes lower in the DOM tree hierarchy.

MHEG events are handled by active link objects while DOM events are handled by registered event listeners. An MHEG link is associated to an event in terms of the event source, the event type and the optional event information. When an event is generated and there is an "associated" active link, it "fires" and the specified action is executed. The DOM event listeners are associated to events in terms of the event target and the event type. When an event's flow meets a corresponding event listener (listeners are attached to DOM tree nodes), the listener action is executed. A listener has to be specifically registered to an event target, and in order to unregister it we have to keep track of the listener object reference and its parameters.

An event may cause several handlers to execute in both models. For both the MHEG links and DOM listeners of the same node, simultaneous "firing" leads to the execution of all the handlers in sequence (however, the order is not specified). A significant difference is that in case of transitional MHEG elementary actions, further elementary actions and fired links, that are out of the new context, should not be handled. However, DOM event listeners are always executed.

MHEG events can either be synchronous or asynchronous, however, both are handled synchronously (i.e. there is no parallel execution). Generally, fired links must be handled as fast as possible (before the execution of the next elementary action). Since asynchronous events are not allowed to pre-empt other asynchronous events, they must be queued for later synchronous execution. A DOM event generated by an event handler is handled synchronously (similarly to MHEG). However, the DOM does not make a distinction between synchronous and asynchronous events, and there is no event pre-emption. If an event is generated while another one is executing, they will be handled in parallel.

Table 5.4 summarizes the comparison of the different event model properties. Based on this information, in the next section we will investigate how to map the MHEG to the DOM event model. We will also propose a design for an implementation that implements this mapping.

5.3.2 Design

Firstly, we will study how MHEG event model concepts map to DOM event model functionality. This will allow an implementation refinement featuring a more compact MHEG event model implementation that takes advantage

Property	MHEG	DOM
Event generation	MHEG objects	DOM implementation
Event parameters	**Source:** MHEG object **Type:** MHEG event type **Optional value:** Integer, boolean or string	**Target:** DOM tree node **Type:** String **Additional information:** Event class members
Event flow	N/A	Event capturing and bubbling
Event handling	Link objects	Event listeners
Handler activation/deactivation	Activation/deactivation of MHEG link	Register/unregister listener
Multiple handlers	Synchronous execution in unspecified order	Synchronous execution in unspecified order
Event recursion	Synchronous execution	Synchronous execution
Parallel handling	All events are handled in sequence	Parallel execution for different execution threads
Transitional behaviour	Specific transitional actions cause removal of some fired links	N/A

Table 5.4: Event models comparison

of the DOM events functionality.

An MHEG event is generated by an MHEG object, because of either an internal behaviour or an elementary action (which is implemented as a member of the MHEG classes). A DOM event can be generated by the `dispatchEvent()` method of an *EventTarget*. Since, both internal behaviours and elementary actions are implemented using Java code, a DOM event generation is a matter of calling the `dispatchEvent()` method.

An MHEG event is associated to an *event source*, a *type*, and an optional variable. The event source can be mapped to the DOM node that corresponds to the source MHEG object. For instance, an event emitted by an application object, could be dispatched to the DOM node that corresponds to the "`<application>`" tag. We have to point out that even if MHEG has an event *source* while DOM has event *target* we could use them interchangeably. The DOM event handlers correspond to events *targeted* to a specific *EventTarget* while the MHEG links handle events from a specified *source*. It is simply a different use of the terminology.

The event type and information mapping is trivial. A DOM event is identified according to its name which can be any string,[2] and an MHEG event type can be mapped to a corresponding string (e.g. map the "IsRunning" event to an "MHEG:IsRunning" DOM event string). An MHEG event can have an associated boolean, integer or string value. This can be included as an attribute of the DOM event class (we have to subclass `org.w3c.dom.Event` in order to provide an event implementation). Summarizing, event types are mapped to DOM event type strings, and associated event data can be included in the event class.

The "Event flow" difference will not introduce any problems, since the DOM event model functionality is more generic than that of MHEG. Specifically, if we disable event capturing and event bubbling, events will only trigger handlers registered for the specified event targets. "Disabling" implies that there will be no "capturing" listeners and the events will not bubble.

The DOM Event handling is performed by *event listeners* which are registered to *event targets*. Therefore, it would be useful to map MHEG links functionality to DOM event listeners. A DOM event listener is registered to a DOM tree node, and thereafter is activated each time a corresponding event flows to that node. As we have already mentioned, the event listener should be registered to the node that is associated to the event's source MHEG object, and the event type will be a string that corresponds to the event name. The principal purpose of the event handler would be to execute the associated elementary actions in sequence. This could be

[2] Except reserved strings which start with "`DOM`".

achieved by holding a reference to the associated MHEG link object, where the elementary actions are stored.

An MHEG link, can fire only if it is active, and can be activated or deactivated by either elementary actions or internal behaviours. This behaviour can be implemented using DOM handlers by either registering/unregistering them or by checking the link status in the handler code. The latter approach offers a simpler implementation (a simple check for each execution) and there is no need to keep track of the object handler references (they are needed in order to remove an event listener). However, the former solution offers better resource usage, since, "inactive" handlers will not be attached to event targets and will not have to be checked for each event. Therefore, we will follow the first approach.

"Multiple handlers" and "Event recursion" properties of the two models are handled identically. If there are more than one listeners activated by the same event they will be handled in sequence, and events generated by event handlers will be processed synchronously. However, if a DOM event is generated while another one is being handled in its own execution thread, they will be handled in parallel. Nevertheless, assuming that the basic engine functionality runs on a single thread, and that only asynchronous events occur in parallel, we simply have to control the asynchronous event handling.

When an asynchronous event handler is activated, it must not preempt handling of another asynchronous event and must be queued. A simple way to implement this is by launching a different thread that waits for other asynchronous events to finish. Consequently, the asynchronous event queue will be implemented as a queue of waiting threads of execution.

Finally, when an MHEG transitional elementary action is successfully executed, pending fired links that are out of the new context should be removed. Moreover, any queued elementary actions must not be handled as well. This behaviour can be a part of the event handler functionality, where there could be checks if the handled link is in "scope" and where elementary actions after transitional ones will be ignored.

Summarizing, we have described a mapping of the MHEG to DOM event model concepts which is illustrated in Table 5.5. An implementation of this mapping would allow MHEG events (and in general, the execution model) handling using DOM event functionality and is described in the next section.

MHEG	$\longmapsto$	**DOM**
Event generation	$\longmapsto$	DOM event dispatching
Event source	$\longmapsto$	DOM event target node that corresponds to the source MHEG object
Event type	$\longmapsto$	A string describing the event
Event data	$\longmapsto$	Event class attributes
Link objects	$\longmapsto$	DOM event handler registered at the respective node. Attribute values will be checked during handler execution.
Activation/Deactivation	$\longmapsto$	register/unregister event handlers
Multiple links	$\longmapsto$	Multiple handlers
Event recursion	$\longmapsto$	Event recursion
Parallel handling	$\longmapsto$	Queue of threads for asynchronous event
Transitional actions	$\longmapsto$	Checks of scope in event handler functionality

Table 5.5: Event models mapping

5.3.3 Implementation

In order to implement the mapping described previously, we have to provide a DOM event class which will represent MHEG events and a DOM event listener which will handle them. Moreover, we have to connect the new event model to the engine.

Firstly, since it is useful to be able to use both the old and this event model we will convert the old *LinkProcessor* class to an interface. There will be two implementations, *LinkProcessorSimple* and *LinkProcessorDOM* for the old and the new event model respectively. The interface will provide the shared functionality for event emission, and *Link* registration and removal. The new link processor hierarchy is illustrated in Figure 5.7.

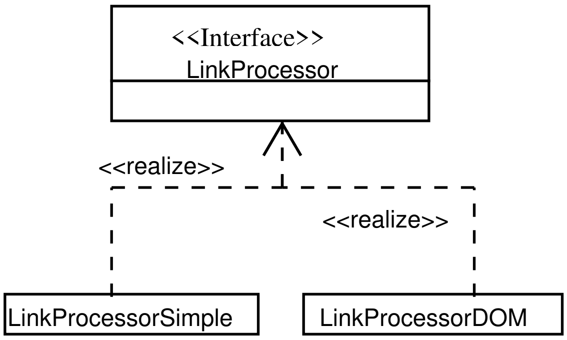


Figure 5.7: New link processor hierarchy

The *LinkProcessorSimple* encapsulates the old event model functionality, while the *LinkProcessorDOM* is based on the DOM event model. Specifically, the `addLink()` member function "activates" a link by adding the appropriate DOM event listener to the event target that corresponds to the event's source object. The listener is created by the *MHEGLink* object, as we will describe below. However, in order to find the "appropriate" target for the listener, there has to be an association between internal MHEG objects and the DOM tree elements. Therefore, we have to add an attribute to *MHEGRoot* that refers to the corresponding DOM node. In order to implement the `removeLink()` behaviour we have to keep a reference to the event handler for each link. Since, *MHEGLink* creates the event handler, it will also be responsible for keeping that reference. Finally, there are four variations of `throwEvent()` that corresponds to the four types of associated data. The `throwEvent()` function creates an event object and dispatches it to the event source's DOM tree node. Moreover, *LinkProcessorDOM* manages execution state information (synchronous or asynchronous) and contains the synchronization variable which is used for suspending and waking up asynchronous event handling threads.

We also have to provide a DOM event class that encapsulates the MHEG events functionality. It has to inherit `org.w3c.dom.events.Event` in order to be a DOM event and should contain information about the type and the associated data of the event. The latter is stored in a general *Object* and the type is stored using an integer value (similarly to the previous model). However, there is a static function which converts and integer type to a DOM event name in order to achieve consistent type translation. The MHEG DOM event class inherits a parser specific event implementation class that provides the basic event functionality. An illustration of the hierarchy for the defined event class, *MHEGEventDOM*, is shown in Figure 5.8.

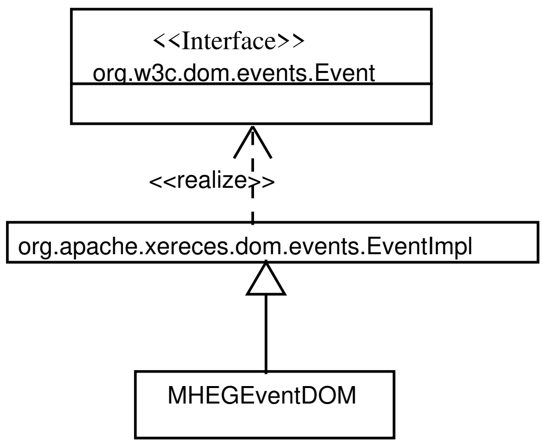


Figure 5.8: DOM event hierarchy

The *MHEGEventListenerDOM* class associates to event handling information by holding a reference to the respective link object. Its core functionality is the event handling and it depends on the type of the associated event. If the latter is synchronous, the listener simply executes the elementary actions in sequence. If it is asynchronous, it first checks if another asynchronous event is handled. In that case, a new thread is launched which waits until notified, otherwise the asynchronous event actions are immediately executed. When, the execution of an asynchronous event ends, one of the waiting threads (if any) is notified. This allows timely execution of asynchronous events, and gives them higher priority than the asynchronous ones (since all waiting threads will be executed before the execution returns to the pending elementary actions). An illustration of the event listener hierarchy is shown in Figure 5.9.

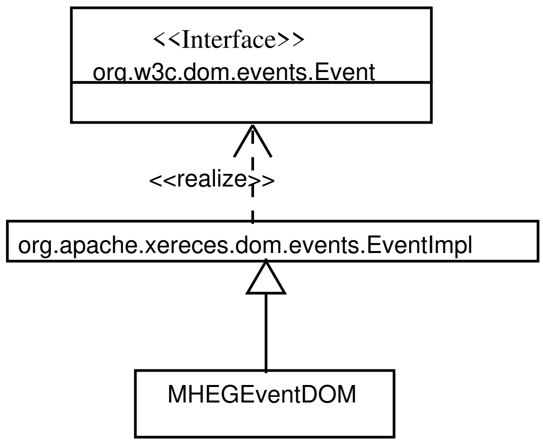


Figure 5.9: Event listener hierarchy

Finally, as we have already mentioned, we have to modify the *MHEGLink* class in order to keep a reference to and construct the associated *MHEGEventListenerDOM* object. All objects that might emit events need to be associated to the corresponding DOM-tree nodes. Since only subclasses of *MHEGRoot* may dispatch events, we simply have to add a reference to *MHEGRoot* that points to the corresponding *Node* of the DOM tree. The new event model design is summarized in Figure 5.10.

5.3.4 Evaluation

In order to test the functionality of the engine with the new event model, we will use the application example of Section 5.2.3 and compare the result to that of the previous engine implementation.

The new implementation behaves exactly as the previous one. The application transition is successful and the asynchronous timer event is handled

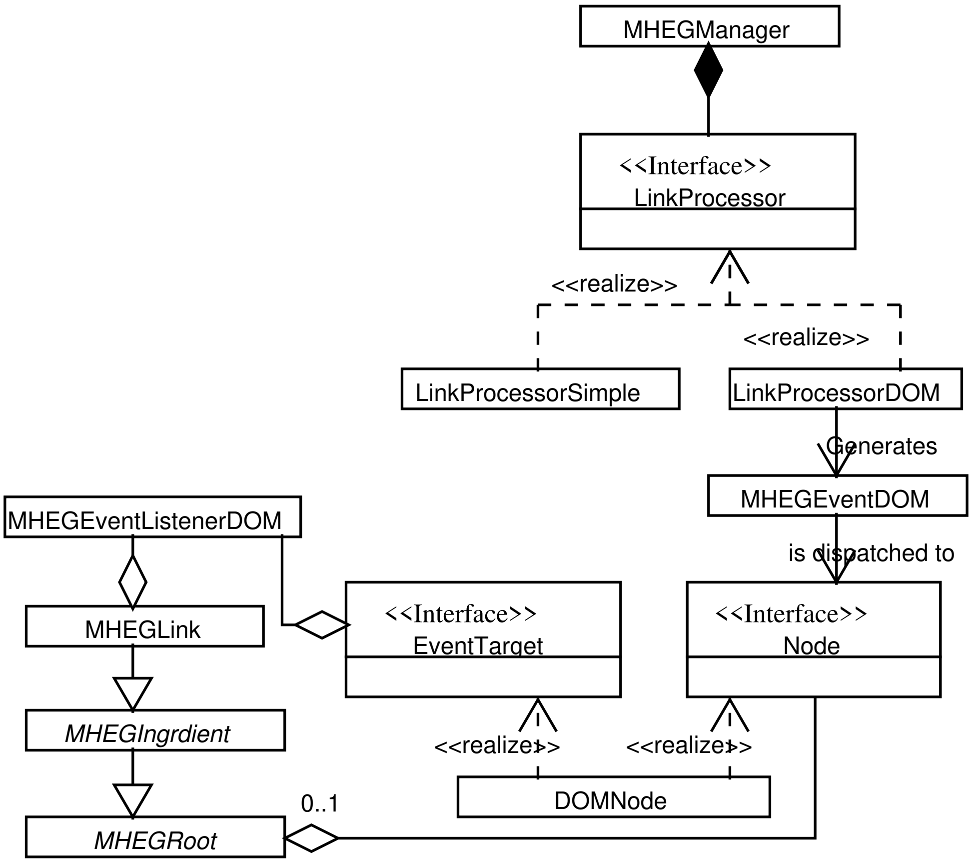


Figure 5.10: New event model design

properly. All the event handling debug output has been replaced by dispatching of events and activation and deactivation of listeners messages. In general the functional part of the new implementation seems correct. The detailed output of engine is included in Appendix E.

Our initial goal was to handle MHEG events using the DOM event model. However, we achieved more than that, since the whole execution model has been replaced by the new implementation. The latter, in addition to the old *LinkProcessor*, does not use the *MHEGProcessor*, the execution queues and stacks, the *Event* class and the event queues. Moreover there is an overall implementation code reduction since the old event handling and execution mechanism was about 700 lines long while the new one is only 350 lines. Even if this is not an objective measure of the overall complexity reduction, we could safely say that the DOM implementation is simpler than the previous one because of the MHEG – DOM event models similarities.

However there are still some problems to be resolved. The handler code might introduce parallel handling of events (which should normally be executed sequentially). This might happen when an asynchronous event link fires while another one is being handled. Nevertheless, this could be solved by additional checking and synchronization code in the handler.

Moreover, transitional elementary actions special behaviour is not correctly handled since out of scope fired links (which are not still processed) are not removed, and the execution continues normally. The behaviour could be corrected by including additional checks into the event handler implementation. Both this and the previous problem can be easily addressed, however, we will not consider them due to lack of available time.

5.4 Conclusion

In this chapter we studied how MHEG functionality can be integrated to a browser in general and to X-Smiles in particular. The integration process was divided in two steps. The first one was the implementation of a core MHEG engine that allowed an in depth study of the MHEG engine implementation, design and its integration with a browser. The second part was mostly focused on the use of a standard browser feature (the DOM event model) in order to provide an easier and more compact engine implementation. Since, we achieved an easier implementation of the MHEG event model using DOM event functionality we could say that we have met our goal.

Of course, there are many more steps the might be taken in order to achieve further integration. However, we were not able to investigate more approaches because the available time for the project was limited.

In the next chapter we will provide a more “high level” evaluation of our work and proposals for further research.

Chapter 6

Evaluation – further research

After describing our work on the integration of web browsers and interactive TV, we will evaluate our achievements, comment on the standards and tools used and provide thoughts for further research on the subject. Moreover, we will compare our work to the initial project plan and justify any identified deviation.

6.1 Integration evaluation

Our principal intention was to study the ITV and web browser domains and investigate the convergence of the two technologies. For the ITV domain we focused on the MHEG-5 standard, while for the browsers domain we focused on the X-Smiles browser and the DOM standard.

We have managed to implement a functional MHEG extension for X-Smiles which uses the MHEG-8 notation. Moreover, we studied how the MHEG event model can be implemented using DOM events functionality. These two steps allowed an in depth study of the MHEG and DOM standards and the X-Smiles architecture.

However, we cannot claim that our study is complete, since there still are some outstanding issues which could have been studied if there was more available time. These will be described below.

6.1.1 Minimal conforming engine

The first part of the integration (Section 5.2) consists of the design and implementation of a minimal conforming MHEG extension to X-Smiles. The implementation is functional, however, it can be argued "how much of an integration" it is. X-Smiles features are used only for connecting the MHEG engine to the browser and the DOM is only used for parsing the XML definitions. Nevertheless, it is a necessary step that provides the foundation for further investigation.

As far as the implementation is concerned, the first part provides all the core aspects of an MHEG engine. However, there are neither *presentable* nor *interactible* objects and consequently there is no explicit multimedia presentation or user interaction. The original intention was to implement these in the second part of the implementation if there was available time. However, because of the project time constraints, we chose to proceed to the events model integration which is of more interest as far as the integration is concerned.

Moreover, the implementation has some conformance problems relating to the transitional elementary actions. Firstly, *TranstionTo* is implemented as part of the application class because there is no automatic loading of referenced objects. Secondly, after the completion of a transitional action, further queued actions and fired links that are out of context are not removed. Again, the reason for these was the lack of time to implement the additional support.

Our basic decisions for the first part were the browser requirements, the minimal application domain definition, the internal object and referencing model, the queue–stack event model implementation, the separate handling of attribute actions and events, and the overall parser design.

The requirements for browser selection still seem reasonable and the selection of X-Smiles was adequate, since the integration of MHEG functionality was relatively easy. However, we probably should stress more the importance of Internet standards support, since, using international standards for the integration is more important than using browser specific functionality.

The minimal application domain definition proved optimal for the project timetable. Even for this very restrictive core, the implementation was quite time consuming and if we had included presentables and interactibles we would have run out of time.

Concerning the internal object model, the mapping of MHEG classes to implementation classes was satisfactory. It resulted to a consistent (with the standard), easy to handle and understand model. The elementary actions class hierarchy provided a clear internal model, however, the required repetition of similar code was quite time consuming. Nevertheless, the adopted referencing model was quite useful and allowed a higher level implementation of the elementary actions and the whole engine in general. Finally, dereferencing handling outside the elementary actions implementations of the MHEG classes successfully separated the concepts of dereferencing and action handling, and consequently contributed to a better overall design.

As we have already mentioned the event model of the first implementation step (using stacks of queues) reflects the intentions of the standard and it functions as expected. However, since a simpler implementation (similar to the description in Section 2.5) would also result to a conforming engine, we could have avoided such a complex approach. The decision to handle the attribute actions independently resulted to a simpler execution model implementation but might introduce parallelism in some occasions, which is probably not the intention of the standard; it might be more appropriate to let the execution queue handle this as well.

Finally, parser design successfully follows the hierarchical form the MHEG-8 syntax. However, it might be useful to allow a further investigation of a simpler implementation (probably relying on the properties of the input syntax, and the validation of the XML parser).

6.1.2 Event models integration

The second part of the integration (Section 5.3) focuses on an implementation of the MHEG event model using the DOM events functionality. We have managed to provide a mapping of the event model concepts and to substitute the first part's event model with a new one that uses the DOM events.

The principal achievement of this part is that the final implementation proved easier and more compact than the old one. Therefore, at least this part of the engine, can be implemented more efficiently using existing browser features. Moreover, the integration is based on an international standard rather on a browser specific model. Consequently, this implementation can be used for any browser that supports DOM events and is not X-Smiles specific.

Concerning the implementation, the core functionality works as expected. Synchronous and asynchronous events are handled similarly to the initial implementation. However, there are some outstanding issues concerning event preemption and transitional elementary actions. Specifically, if a synchronous event is being executed and an asynchronous event is generated, instead of pre-empting the execution or waiting in a queue, the asynchronous event will be handled in parallel. However, when there is a collision of two asynchronous events the queuing is handled properly. Moreover, similarly to the old model, transitional elementary actions are not handled as specified by the standard because pending elementary actions and fired links might still be executed.

Finally, we have to point out that a full study of the event models integration was not possible because of the lack of the engine support for presentables and interactibles. This prohibited the investigation a mapping from DOM *UIEvents* to MHEG user interface events. There would also be an association with the *Views*[29] part of the DOM standard. However, even if there was adequate support from the core engine, there would be additional problems since current X-Smiles version does not support neither *UIEvents* nor *Views*.

6.1.3 General comments

Generally, we could say that the main obstacle for the integration process was the project time constraints, and the need to implement the core MHEG engine in order to gain the required understanding of the standards and to support further study. The implementation of the core engine proved quite time consuming and therefore we managed to investigate only one case of further integration (the event models). Based on this we can argue that the incremental design and implementation approach was a good decision since otherwise we might have ended up with a non working implementation or incomplete design.

The decision to use MHEG-8 for MHEG representation proved useful since the integration process was made easier. XML validation in addition to the DOM functionality provided by the browser reduced the complexity of the parser and made the event models integration feasible. Therefore we could safely argue that the MHEG-8 standard contributes towards the convergence of ITV and web technologies.

DOM events support allowed event models integration mainly because DOM model is more generic than the MHEG one. However, since there is no distinction between synchronous and asynchronous events, we had to incorporate the respective MHEG functionality using Java features (threads, synchronization et cetera).

The X-Smiles specific features were not extensively used for the integration because X-Smiles is not restrictive and gives much freedom to the

MLFC implementation. Therefore, most of the engine can be implemented almost independently of X-Smiles. However, if we had included presentables and interactibles we might have had more X-Smiles – MHEG interaction (at least for the user interface support).

Finally, we have to say that our study was focused on the MHEG-5 integration into a web browser. We have not studied how the MHEG content will reach the browser (e.g. through a web server or an STB). Several problems were solved with this approach since otherwise we would have to additionally consider different transfer protocols, caching schemes, content handling et cetera.

6.2 Comments on the standards used and X-Smiles

After implementing the MHEG engine we have concluded that MHEG-5 is quite powerful for representing interactive multimedia. It allows development of versatile applications and the event model is designed in such a way that can be handled by low resource target platforms. Moreover, the concept of the application domain is very useful for adapting the model into a wide variety of domains. However, the standard has some relatively vague parts which need further specification in order to assist the engine and application developer. For instance, the event model could benefit by a more detailed explanation.

Since XML is supported by most of the current browsers, MHEG XML notation is of great use because it allows use of standard XML features which make the process of input processing much easier. Moreover MHEG-8 syntax allows hierarchical processing of input in a way that corresponds to the MHEG class hierarchy. However, the resulting DOM tree represents MHEG objects containment and not the object hierarchy or the presentation spatial containment. If that was the case, we could make use of event bubbling and capturing event handling in order to map MHEG classes behaviour to DOM events concepts. Moreover, the MHEG-8 standardization might enforce general purpose use of MHEG for other areas in addition to ITV. Nevertheless, even if XML languages are easy to write by hand (since XML is a textual notation), in order to develop a useful MHEG application, an authoring tool has to be used. This is a consequence of the amount of XML code that has to be written and of the object referencing scheme (which uses numerical indexing instead of naming).

6.3 Comparison to the original project plan

The original project plan (included in Appendix B) was to integrate MHEG functionality to the Mozilla web browser. Our main goal was to identify which parts of the MHEG engine can be implemented using existing browser features and to modify Mozilla in order to make it MHEG aware. Moreover, our implicit intention was to straightforwardly implement the MHEG functionality using browser features without first implementing a restrictive core which uses only the absolutely necessary browser components.

Firstly, we have used X-Smiles instead of Mozilla. The latter was initially chosen because it is a full featured and more mainstream browser. However, the complex design and the lack of adequate documentation lead us to reconsider the target platform for our research. Consequently, we have introduced the browser assessment chapter where our main goal was to identify the appropriate platform for our study. X-Smiles proved to be the best solution among the alternatives we have set.

The initial plan was to implement MHEG functionality using browser features and the W3C's XML, DOM and XSL standards. We finally used the XML (for document processing) and DOM (for document processing and event models integration). However, we have not extensively used browser features, as explained above, and we did not investigate how other W3C standards, like XSL, might be of use because of the time constraints.

Finally, we originally intended to proceed to the integration in a single step and to provide an MHEG engine with at least some basic support for presentables. However, the inherent complexity of the whole process lead us to separate the process into a number of steps of which only the first two were investigated.

6.4 Further research

After evaluating our study we will examine possible further research in different levels of abstraction.

6.4.1 Implementation corrections – extensions

At first, the MHEG execution model for both parts of the integration requires some further consideration. Probably, the attribute actions (e.g. *onActivation*) execution should be integrated with the events execution. For the first part's engine we simply have to modify the processor in order to handle them. For the second part the solution might be to emulate them using DOM events. Moreover, transitional actions behaviour should be corrected. Pending fired links that are out of the new context should be removed and further elementary actions should not be executed.

The object loading mechanism should be integrated into the general referencing functionality of the application object. This way there will be no need to explicitly load objects when needed (e.g. at a *transitionTo* action). When an object that is visible by the application namespace is referenced, it should be loaded automatically. This will also allow a standard conforming implementation of the transitional elementary actions which are currently implemented as functions of the currently active application.

Moreover, the execution of asynchronous and synchronous events for the second part of the integration should be synchronized in a way that there is no parallel execution. This would require suspending synchronous events execution when an asynchronous one is handled. However, synchronous events which are launched as part of the asynchronous event handler should be handled normally.

It would also be interesting to investigate how event models integration could be achieved without the use of threads. The MHEG-5 standard is designed for light weight platforms and using threads to emulate the event queue might be quite resource consuming.

Finally, we should include handling of presentable and interactible MHEG objects in order to have a full-featured engine. This approach should focus on using the least possible X-Smiles specific features so that the engine could be ported to another browser platform in order to check the above concepts in different architectures.

6.4.2 Further integration

A basic step for further integration would be to study how MHEG user events can be mapped to DOM Events concepts. That would require the implementation of presentables and interactibles and a DOM implementation that supports *UIEvents* and DOM Views[29]. For instance, an association between the DOM objects and their screen representation could allow generation of DOM *UIEvents*. This could then be mapped to MHEG user events and used for supporting user interaction.

The integration of other MHEG components (except the event model) could also be studied. For instance, elementary actions internal representation might be substituted by the respective DOM one. This would require an elementary action execution design which will use DOM information in order to process elementary actions. Moreover, the whole internal object model could be substituted by a DOM object model. In that case internal behaviour execution could be emulated using event bubbling and capturing. This would also require a DOM tree transformation in order to create a placeholder for internal class attributes and for creating a more adequate tree structure. This transformation could be achieved by the use of the XSL-T[26] functionality.

It would also be interesting to use other supported standards for the integration process. For instance some MHEG interactibles (like a push button or a text area) could be implemented using the corresponding HTML elements.

6.4.3 Related research ideas

As we have already mentioned, we have studied a restricted case of the ITV domain were a personal computer was used for browser execution and simple HTTP transmission of MHEG information and content was assumed. The whole concept could be studied in a wide variety of configurations. For instance, the browser might be running on an STB. In that case the transmission protocols might be different and there will be additional caching issues (for instance, if the transmission is a based on an object carousel). Moreover, the use of MHEG-8 should be re-examined if significantly different transmission protocols are to be used.

In the case were the browser is run on an STB, we could have both ITV and web support by an MHEG aware browser. However, in such a resource scarce environment several additional constraints must be taken into account (e.g. memory and processor usage, available bandwidth etc)

As far as the transmission of content is concerned, depending on the target platform, different approaches can be investigated. For instance, the "XML protocol"[31] is a W3C working draft that could be used for MHEG objects information. Also the transport protocol for real time applications (RTP)[10] could be used for content transmission. However, the protocols to be used should always be studied in relation to the target platform. For instance, HTTP transmission might be a good solution for a web browser on a desktop computer, however it might not be adequate for a STB or a mobile phone.

An interesting investigation would be the comparison of the MHEG-5 and SMIL[25] standards for web based multimedia applications. MHEG-5 was not originally intended as a web application, but since the publication of the MHEG-8 standard, it has become an attractive way for supporting interactive multimedia for the web. Moreover, SMIL is considered one of the dominant current web multimedia standards. A comparison of the two standards would be useful for testing the applicability of both for modern multimedia applications and for identifying the advantages and disadvantages of each.

Finally, our study could be extended in order to investigate a generic way to support XML content. XML is able to represent virtually everything because it is only concerned with the structure of the information. However, the semantics information is lost as well as the information on how to handle the content. In order to avoid incompatible browser extensions for the support of specific document types, a generic semantics language may be developed. The latter will provide additional semantics information in a way similar to XSL. However, it will be concerned with the "semantics" of the tags and on how they should be processed and presented. This effort could benefit from the shared required functionality for different content types (e.g. parsing, internal object model, rendering machine etc). A superset of this functionality could be provided by the browser core and the additional semantics information will simply "customize" the existing browser components in order to handle a specific content type.

Chapter 7

Concluding remarks

Our main concern throughout this project was to investigate the integration of the Interactive TV (ITV) and web browsers. As we have already mentioned, the study of the integration is important because both areas will benefit since it will help towards enhanced services for both of the domains. Web browsing will include interactive multimedia services while interactive TV would benefit from the ability to access the vast amount of existing interactible web information.

Our approach was based on the modification of the X-Smiles browser in order to introduce MHEG functionality. X-Smiles was selected as the target web browser platform after a browser assessment process in which we studied six browser alternatives for the most adequate one for our research. The MHEG-5 standard was selected as representative for the ITV because it is accepted as part of the DAVIC ITV specification and as the U.K. terrestrial ITV platform. Moreover, the recent MHEG-8 standard, which defines an XML representation for MHEG-5 content, was an additional reason for the use of MHEG, since, most of the current browsers support XML and the related Internet standards.

We divided the integration process in two steps. The first one was the implementation of an MHEG extension for X-Smiles which provided the basic MHEG core functionality. This was achieved by defining a minimal application domain to which the implementation conforms. The core engine implementation used only the absolutely required X-Smiles features which included the XML parser and the DOM model. The second step was concerned with the MHEG and DOM event models. Our goal was to implement the MHEG event model using existing DOM event functionality. Our final achievement was to provide an event model implementation which was easier to write and more compact than the initial approach.

Our study can be extended to further integrate the two areas and to use as many existing standard browser features as possible. Within the constraints of the project time limits, we were able to demonstrate the feasibility of integrating these two models. Further integration should be relatively straightforward on top of the foundation laid within this work.

Appendix A

Abbreviations

API: Application Programming Interface.

ASN.1: The MHEG notation defined in the first part of the standards.

CSS: Cascading Style Sheets standard.

DAVIC: The Digital Audio and Video Council.

DOM: The Document Object Model standard.

DOM Events: The DOM-2 Events standard which specifies the DOM-2 event model.

DTD: Document Type Declaration.

HTML: HyperText Markup Language standard

HTTP: HyperText Transfer Protocol

IDL: Interface Definition Language

ITV: Interactive television

MHEG: Multimedia and Hypermedia information coding Experts Group. Also, the family of the 8 MHEG standards.

MHEG-5: MHEG part 5, "Support for base-level interactive applications" standard.

MHEG-8: MHEG part 8, "XML Notation for ISO/IEC 13522-5 (MHEG XML)" standard.

MLFC: Markup Language Functionality Component.

OO: Object Oriented.

STB: Set-top box.

W3C: The World Wide Web Consortium

WWW: World Wide Web

Web client: The application used to present web content.

X-Smiles: The browser platform used for the integration process.

XSL: The eXtensible Stylesheet Language standard.

XSL-T: The XSL Transformations standard.

Appendix B

Initial project description

This appendix contains a copy of the original project description with slight modifications in order to fit the new document layout.

B.1 Introduction

The project is concerned with the integration of two technologies that used to be distinct and evolve almost independently: web and Interactive TV.

Most of the web browsers were only capable of displaying simple HTML hyperlinked text, transferred using HTTP protocol. However, due to the evolution of networks and computer architectures, different requirements came into play. One of them is interactive multimedia, which is inherent in the field of interactive TV. Therefore, the integration of the two technologies seems beneficial and it will allow users to seamlessly move between web and interactive multimedia content.

What we are going to investigate, is the integration of MHEG functionality into web browsers. We have chosen MHEG because it is the accepted standard for providing interactive multimedia content for UK terrestrial digital TV. MHEG-8 is an XML representation of MHEG objects, which makes it even more applicable for processing by an XML-aware web browser.

As far as the browser is concerned, we are going to use Mozilla, an open source browser that incorporates most of the latest web technologies.

We expect to extend the above investigation to the integration of MHEG functionality into Mozilla.

B.2 The problem area

The project will involve two basic challenges. Firstly, to manage to find the commonalties among different web technologies and the MHEG standard. Secondly, the modification of Mozilla, which, due to lack of documentation, can be considered a research effort by itself. Below, we will give a brief overview of the different technologies that are expected to be involved, some example relationships among them, and a brief overview of Mozilla.

B.2.1 The standards

MHEG

MHEG provides a standard way of representing and transferring interactive multimedia objects. The objects and the relationships between them describe the structure of an interactive multimedia application. Actually, the top-level object is the "Application" object. It may contain scene objects, which in turn may contain, among others, media objects. Each object has an interface, which is the set of functions that can be performed on it. The MHEG event model is able to represent object and user interactions and reactions to special internal events. When an event is fired (e.g by the expiration of a timer, a user interaction or the end of a video clip), the actions to be taken are described by links. These actions are in form of sequential elementary actions, which are performed on objects (similarly to member calls in object oriented programming languages).

When we say "transferring interactive multimedia objects", we mean that a standard way to encode and transfer object, structure and event handling information. This is based on ASN.1 (abstract syntax notation 1) which can take textual or binary form.

MHEG-8 extends MHEG by providing an XML representation for describing and transferring objects. Since, web and Internet users are more used to these kind of mark-up languages, it is most probably that the adoption of MHEG-8 will speed up the integration of Interactive TV and web.

MHEG content is processed and presented by an MHEG engine. What we are interested in, is to incorporate the functionality of an MHEG engine into web browsers (specifically Mozilla). We have to investigate if we can reuse existing components of Mozilla in order to construct the MHEG engine. For instance, use the XML parser to parse the MHEG specification, or use DOM-2 functionality to internally represent MHEG objects and events.

XML

XML will certainly be of involved because it is used for MHEG representation. As far as core XML is concerned, we are only interested in the parsing of the MHEG information. This is expected to be the easiest of the relationships to be implemented because it doesn't require any modifications.

DOM

DOM-1 is a W3C recommendation that emerged as a way to retrieve documents and to describe document structure. Most of the XML parsers use a DOM tree to represent the parsed information. DOM-2 is an extension of DOM-1, which allows many different kinds of information to be represented. We are mostly interested in the event model of DOM-2. We will investigate if it is possible to use it for the representation of MHEG events.

XSL and CSS

XSL and CSS are W3C recommendations as well. We are not interested in a direct relation between them and MHEG. However, the functionality that a browser must offer in order to support the display of documents using extended display information (mostly for CSS) can be useful in representing

MHEG content. For instance, CSS allows dynamic modification of mark-up properties. There might be a way to use this functionality in order to allow elementary actions to alter the attributes of MHEG objects.

Other Standards

The set of standards that will be involved is not clear at the moment, because further studying and investigation of the core technologies (MHEG and Mozilla) is required.

B.2.2 Mozilla

We have chosen Mozilla as the browser paradigm for our analysis and implementation. Mozilla is an open source project, co-ordinated by "mozila.org".

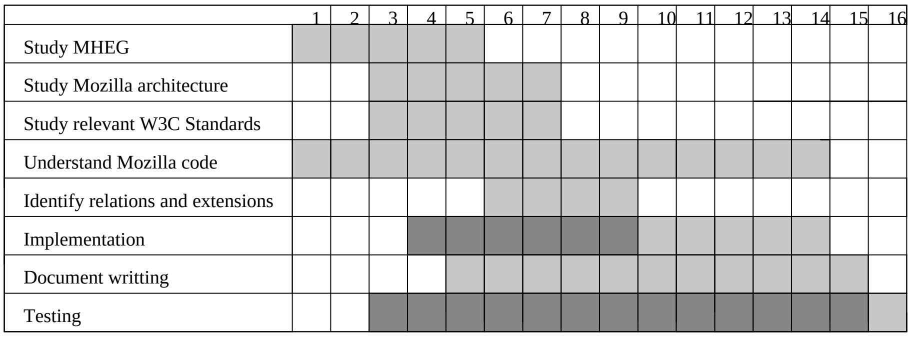
Mozilla is cross platform and easily extendable. Most of the code is in a subset of C++, which is defined by specific cross-platform constraints. The architecture is based on the core XPCOM functionality and modules plugged into it. XPCOM is a cross-platform equivalent to Microsoft's COM. The modules communicate through interfaces, which are defined in XPIDL (the Mozilla alternative to IDL). That means that it is "easy" to extend Mozilla, even by using a language other than C++. It currently supports, among others, XML, XSL, DOM, DOM-2 (partly), CSS and Java. Most of these are implemented as independent modules communicating through XPCOM.

We are mostly interested in identifying the modules which are related to the theoretical analysis and extend them to incorporate MHEG functionality. We will most probably develop some new modules for MHEG parts that cannot be integrated by extending existing code. The involved modules and the required extensions are still quite unclear since their identification is one of the core parts of our research.

The main disadvantage of Mozilla is the lack of complete up to date documentation (for development). Therefore, extending Mozilla involves investigation of the code (more than 1 million of lines - 250 Mbytes source) in order to identify how everything fits together. However, there is a sophisticated source-cross-referencing mechanism, which eases navigation through the code and makes the "quest for comments" easier.

B.2.3 Project schedule

The figure below is an overview of the project schedule divided into 16 weeks.



Firstly, we will start by studying MHEG standards. At the same time, we begin the exploration of Mozilla code, which will almost last until the end of the project. Initially the focus would be on the core aspects, like XPCOM interface. Gradually, we will proceed to more specific parts of the code, according to our research. Understanding Mozilla code will be an on-demand based process.

On week 3, after gaining a basic understanding of MHEG, we will start investigating W3C standards and parts of Mozilla architecture that seem more relevant. At the same time, we will focus on understanding the specifics of Mozilla implementation related to these standards (for the "Understanding Mozilla code" part).

After finishing the study of MHEG and having a brief idea about relevant W3C standards and Mozilla architecture, we can begin investigating the relations that can be achieved among them. At that point (week 6), a report describing MHEG should be ready.

During this period, basic extensions and integration of code in Mozilla should have already started. This contains probably the core of MHEG engine, handling of XML files and the incorporation of the new file format in Mozilla.

After finishing and reporting the investigation of the relations, the main part of the implementation begins, which aims in realising the specified relations and extensions. The implementation phase must be finished by the end of week 14. At this point, basic tests and debugging on individual parts of the implementation should have been finished as well (testing phase begins one week after the start of the implementation). In week 15, our main concern will be debugging the final compilation of modules, testing the implementation on different platforms and reporting about this process.

Throughout the project, we will produce different reports ("Document writing"), that will constitute parts of the final dissertation. The compilation of these reports and the production of additional material must take place before week 16. In that week we will overview, correct and finalise the dissertation.

We must point out, that this schedule is based on the current knowledge and intuition about the whole process. Probably, after studying the relevant material, our schedule may change significantly. Additionally, there is the highly unpredictable part called "Mozilla"! Since, understanding Mozilla involves considerable searching through the code (for comments and interfaces), and since there is no complete up to date overview of its architecture, it is difficult to plan our time.

B.3 Conclusion

Our main objective for this project is to hand in a detailed overview of the standards, ways to combine them, and have a working version of an MHEG-aware version of Mozilla. We do not intend to implement the whole MHEG standard. We will only consider the parts that demonstrate out theoretical analysis.

Appendix C

Minimal application domain definition

- **Exchanged representation**: The representation defined by the MHEG part 8 standard ([15]).
- **Group Identifier encoding**: Relative or absolute URIs as specified by IETF's RFC 2396 ([17]). They should point to the file that describe the corresponding group object.
- **Set of classes**: The minimal set of classes defined by the standard are *Application*, *Scene*, *Link* and *Action*. We have to include the super-classes of these, the *Variable* and its sub-classes which are needed for support of the necessary elementary actions. The resulting set of classes is illustrated in Table C.1.
- **Set of features**: Features are defined as optional or mandatory (in order for an engine to conform to the application domain). In our case, all optional features will *not* be implemented. Since, none of them is needed for the illustration of the core engine functionality we specify all of them as optional (Table C.2).
- **Content data encoding**: Since we do not support any presentables, there is no content to describe.

Supported MHEG classes
Root
Group
Ingredient
Application
Scene
Link
Action
Variable
BooleanVariable
IntegerVariable
OctetStringVariable
ObjetRefVariable
ContentRefVariable

Table C.1: Minimal application domain classes

Feature	**Requirement**
Ancillary connections	Optional
Caching	Optional
Cloning	Optional
Free moving cursor	Optional
Scaling	Optional
Stacking of applications	Optional
Trick modes	Optional

Table C.2: Minimal application domain features

- **UserInput registers**: There will be no *UserInput* events. However, since the specification of a *UserInput register* is mandatory for the scene encoding, we have to define a value for a null register. We will use the integer value "1".

- **Semantic constraints**: Since there are not presentables, semantic constraints are covered by the "set of features". However, we will provide the required table (Table C.3).

- **Engine Events**: There will be no engine events.

Feature	**Constraint**
FreeMovingCursor	Optional
ApplicationStacking	Optional
Scaling	N/A
SceneCoordinateSystem(X,Y)	No combination is supported
SceneAspectRatio(W,H)	No combination is supported
AncillaryConnections	Optional
TrickModes	Optional
MultipleRTGraphicsStreams(0)	Zero
MultipleAudioStreams(0)	Zero
MultipleVideoStreams(0)	Zero
OverlappingVisibles	Not supported
Cloning	Optional

Table C.3: Minimal application domain constraints

- **GetEngineSupport**: No additional GetEngineSupport strings.

Appendix D

Browser alternatives

This appendix is devoted to the 6 different browser platforms that we have tested in order to identify the most appropriate for the integration. There is a brief description for each browser, which justifies the results shown in Table 4.2 (page 37).

D.1 Mozilla browser

Mozilla is an open source web browser "designed for standards compliance, performance and portability"[20]. Mozilla is derived from Netscape Communicator, which was released as open source. The original version had many problems because it was an early release and some of the proprietary Communicator components were removed. Mozilla organization leads the development of Mozilla browser, and the main goal is to make it a fully functional, standards compliant browser.

Mozilla is implemented mainly in C++, and its architecture is based on an XPCOM core. XPCOM is an open source alternative to Microsoft's COM. It allows different components (possibly implemented in different languages) to interoperate in a language independent manner. Component interfaces are defined in XPIDL which is an alternative to IDL.

There are different support libraries for many aspects of the browser functionality. For instance, there is a layout engine, a network library, a user interface library and many others. Most of them are highly customizable, since the development of Mozilla components produces general purpose software that can be used even outside the scope of a browser.

As far as the supported media types are concerned, only basic image formats are supported. This lack of media support is probably a result of Mozilla's platform neutral design and of the fact that is difficult to find open source, portable libraries for the more "advanced" media types. However, since basic graphics and user interaction are supported, it is possible to build the basic MHEG functionality on top of them.

Mozilla event model seems[1] to be hierarchical. When an event is generated or caught by a Mozilla component, it is propagated to all other components that might be interested (not in a broadcast, but in a recursive manner). We have to point out that in parallel with the browser's internal event model, DOM events are also propagated and handled.

There is support for most of the Internet standards like HTML, XHTML, CSS (1,2,3), XML, DOM and others. However, the support for most of them is not yet complete, and there are still many bugs to be corrected (the development team is trying hard to produce the stable 1.0 release).

In general we can say that Mozilla is a rather complete browser, with a well designed modular architecture. However, the lack of good documentation and the very primitive media support might prove problematic for our research. Moreover, its complexity may not be adequate for the time constraints of the project.

[1]The problem with Mozilla is that it is not easy to find up-to-date documentation – except for the comments in the source code – that describes in detail the architecture and how everything fits together. Therefore, at this stage we can only make assumptions for the internal architecture.

D.2 X-Smiles browser

X-Smiles is an open source, Java based, XML browser, which is "a non-profit project started by the Telecommunications Software and Multimedia Laboratory at Helsinki University of Technology"[32]. The basic difference from the other alternatives is that it doesn't support HTML. As a part of a research project, it is not (yet) aimed to provide a wide-range of services to the end-user. However, it supports XML, so all XML languages might as well be supported (actually, there might be support for XHTML in the future). The latest version (0.32) supports XML, XSL-T, XSL-FO, XForms, ECMAScript, SMIL 1.0 and DOM-1. DOM-2 is also partially supported. As far as media types are concerned there is support for *GIF* and *JPEG* image formats, for *MPEG* and *AVI* video formats and for *WAV* audio data.

One of the main goals in X-Smiles development is to provide support for multimedia services for either desktop or embedded devices. This is in line with our goals since MHEG describes interactive *multimedia* content. Additionally, the ability to run on scarce resource embedded devices, is advantageous for support of MHEG in set-top boxes.

Moreover, X-Smiles has very good documentation for both the user and the designer. Its internal architecture is quite simple. Basically, it uses an "event broker" which dispatches events among browser components. This architecture is easily extended since other components can be added by simply registering them to the event broker (allowing them to receive and handle all the appropriate events).

As a conclusion, we can say that X-Smiles seems quite promising for our project. However, there are drawbacks. It is still in its very first releases and inevitably there are problems in the implementation. The Java platform offers portability and support libraries, but introduces high delays which make a complex program like a browser to run relatively slow.[2]

[2]However, this will not be an issue for a set-top box that implements the Java virtual

D.3 Amaya browser

Amaya is a W3C's open source browser – authoring tool[24]. The principal goal of Amaya development is to provide a tool for testing new web technologies. In order to accomplish this, it is designed in a well structured extensible manner with quite good documentation (for both the end user and the developer). The basic drawback is that it is implemented in C, because it is based in a document editing-presentation library called "Thot" which is also implemented in C.

In addition to XHTML it also supports CSS, MathML, XML and XLink. The later two are partially implemented, since only the required features for supporting XHTML are included. The documents are represented internally as a tree (similar to DOM tree) which corresponds to the document structure. However, some important features like JavaScript, animated images and frames are not supported. This might be a reason to avoid selecting Amaya since the lack of this support might imply problematic extensibility to other technologies. Actually, it seems like Amaya is quite extensible as far as HTML specific extensions are considered, but when totally new data formats are concerned (like XML or MHEG) it doesn't seem promising enough. If we try to make Amaya MHEG aware there will be the danger of having to re-engineer the whole browser and our research goal might fail.

D.4 HotJava browser

HotJava (TM) browser[22] is a development of *Sun microsystems corporation*, and is implemented in Java. Its main aim is to provide a light-weight browser that can be used for devices like set-top boxes. Since MHEG support for interactive TV is mainly focused on this resource scarce environments, HotJava initially looked like a good solution. However, since the latest release, the source code is no longer available. It seems that it will

machine in hardware.

be published soon but for now we cannot use this platform, so we do not further investigate it.

D.5 Arena and Mosaic browsers

Even though these two browsers are different, we study them in the same section since they can only be considered as of historic interest. Arena[1] was derived from a text-mode browser. As long as it was developed it managed to keep up to date with most of the Internet standards. However, the development seems to have stopped since Netscape's code was made available to public, and the Mozilla project started.

Mosaic browser[21] was an NCSA (The National Center for Super computing Applications at the University of Illinois at Urbana-Champaign) project, and was quite famous before some years. However, it was not an open source project, therefore the architecture and code documentation is difficult to find. The development seems to have stopped and it is not supported any more.

These browsers are now considered obsolete. However we mention them since they were quite famous as alternatives to the dominant commercial browsers that are widely used now.

Appendix E

Execution examples

This appendix contains the source XML files and the engine output for the execution examples of Sections 5.2.3 and 5.3.4.

E.1 XML sources

E.1.1 The application file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mheg5 SYSTEM "mheg5.dtd" >
<mheg5>
  <application groupid="reportExAp.xml"> <!-- the application-->
    <items>
      <link objnum="1">
        <linkcondition>
          <eventsource objnum="0"/>
          <eventtype type="isrunning"/> <!-- fire at activation-->
        </linkcondition>
        <linkeffect>
          <action>
            <transitionto> <!--activate the scene-->
              <objref objnum="0" groupid="reportExSc.xml"/>
            </transitionto>
          </action>
        </linkeffect>
      </link>
    </items>
  </application>
</mheg5>
```

Listing E.1: The example application

E.1.2 The scene file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mheg5 SYSTEM "mheg5.dtd" >
<mheg5>
  <scene groupid="reportExSc.xml"> <!-- the scene object -->
    <items>
      <integervar objnum="1"> <!--The integer value -->
        <integervalue origvalue="2000"/> <!--initial value-->
      </integervar>

      <link objnum="2"> <!-- Fires at scene activation -->
        <linkcondition>
          <eventsource objnum="0"/>
          <eventtype type="isrunning"/>
        </linkcondition>
        <linkeffect>
          <action>
            <setvariable> <!-- Set integer var. to 10000-->
              <objref objnum="1"/> <!-- Target variable -->
              <newgenericinteger>
                <integer value="10000"/> <!-- The new value-->
              </newgenericinteger>
            </setvariable>

            <settimer> <!--Activate the timer-->
              <objref objnum="0"/> <!-- Target scene -->
              <integer value="1"/> <!-- Timer ID -->
              <indirectref objnum="1"/> <!-- Timer value -->
             </settimer>
          </action>
        </linkeffect>
      </link>

      <link objnum="3"> <!-- Fires when timer expires -->
        <linkcondition>
          <eventsource objnum="0"/>
          <eventtype type="timerfired"/>
          <eventdata>
             <integer value="1"/> <!-- Timer id=1 -->
          </eventdata>
```

```
        </linkcondition>
        <linkeffect>
          <action>
             <quit> <!-- Quit the application -->
                   <objref objnum="0" groupid="reportExAp.xml"/>
             </quit>
          </action>
        </linkeffect>
      </link>
    </items>
    <inputeventreg num="1"/> <!-- the dummy input register -->
    <scenecs xscene="800" yscene="600"/>
  </scene>
</mheg5>
```

Listing E.2: The example scene

E.2 Engine output for Section 5.2.3 test

```
Time: 03:03:18 [DEBUG]: Parsing document: reportExAp.xml
Time: 03:03:18 [DEBUG]: Parsing document: Using custom parser
Time: 03:03:21 [DEBUG]: Document parsed:reportExAp.xml
APPLICATION TREE
***MHEGApplication***
Ref:(reportExAp.xml:0)
RunStat:false
AvailStatus:false
Info:No info
OnStartup: null
OnCloeDown: null
Items:
        ***MHEGLink***
        Ref:(reportExAp.xml:1)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Type: 4
        Source: (reportExAp.xml:0)
        Effect:
                ***MHEGAction***
                Elementary actions:
                        ***TransitionTo***
                        Target: (reportExSc.xml:0)
                        TransEffect: null
```

```
                          ConnTag: null
OnSpawnCloseDown: null
OnRestart: null
FINISHED: false
Starting MHEG processor
Time: 03:03:21 [DEBUG]Execution: EXEC_QUEUE: Waiting
Time: 03:03:21 [DEBUG]: APPLICATION: Running
Time: 03:03:21 [DEBUG]Internal behaviour: Group:
    activation at (reportExAp.xml:0):starting
Time: 03:03:21 [DEBUG]Internal behaviour: Root:
    activation at (reportExAp.xml:0):starting
Time: 03:03:21 [DEBUG]Internal behaviour: Group:
    Preparation at (reportExAp.xml:0):starting
Time: 03:03:21 [DEBUG]Internal behaviour: Root:
    Preparation at (reportExAp.xml:1):starting
Time: 03:03:21 [DEBUG]Internal behaviour: Ingredient:
    content preparation at (reportExAp.xml:1):starting
Time: 03:03:21 [DEBUG]Internal behaviour: Ingredient:
    content preparation at (reportExAp.xml:1):finished
Time: 03:03:21 [DEBUG]Internal behaviour: Root:
    Preparation at (reportExAp.xml:1):finished
Time: 03:03:21 [DEBUG]Internal behaviour: Root:
    Preparation at (reportExAp.xml:0):starting
Time: 03:03:21 [DEBUG]Internal behaviour: Root:Content
    Preparation at (reportExAp.xml:0):called
Time: 03:03:21 [DEBUG]Internal behaviour: Root:
    Preparation at (reportExAp.xml:0):finished
Time: 03:03:21 [DEBUG]Internal behaviour: Group:
    Preparation at (reportExAp.xml:0):finished
Time: 03:03:21 [DEBUG]Internal behaviour: Root:
    activation at (reportExAp.xml:0):finished
Time: 03:03:21 [DEBUG]Internal behaviour: Link:
    activation at (reportExAp.xml:1):starting
Time: 03:03:21 [DEBUG]Internal behaviour: Root:
    activation at (reportExAp.xml:1):starting
Time: 03:03:21 [DEBUG]Internal behaviour: Root:
    activation at (reportExAp.xml:1):finished
Time: 03:03:21 [DEBUG]Internal behaviour: Link:
    activation at (reportExAp.xml:1):finished
Time: 03:03:21 [DEBUG]Event: Event fired: isrunning
Time: 03:03:21 [DEBUG]Event: Event isrunning is in
    queue
Time: 03:03:21 [DEBUG]Execution: EXEC_QUEUE: Restarting
Time: 03:03:21 [DEBUG]Execution: EXEC_QUEUE: Preparing
    to execute next action
Time: 03:03:21 [DEBUG]Execution: EXEC_QUEUE: Found
    action fi.hut.tml.xsmiles.mlfc.mheg5.objectmodel.
    elact.TransitionTo
```

```
Time: 03:03:21 [DEBUG]Elementary Action: Application:
    transitionTo at (reportExAp.xml:0):started
Time: 03:03:21 [DEBUG]Elementary Action: Application:
    transitionTo at (reportExAp.xml:0):deactivating non
    shared application objects
Time: 03:03:21 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExAp.xml:1):starting
Time: 03:03:21 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExAp.xml:1):starting
Time: 03:03:21 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExAp.xml:1):finished
Time: 03:03:21 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExAp.xml:1):finished
Time: 03:03:21 [DEBUG]Elementary Action: Application:
    transitionTo at (reportExAp.xml:0):deactivating
    current scene
Time: 03:03:21 [DEBUG]Elementary Action: Application:
    transitionTo at (reportExAp.xml:0):parsing new scene
Time: 03:03:21 [DEBUG]: Parsing document: reportExSc.
    xml
Time: 03:03:21 [DEBUG]: Parsing document: Using custom
    parser
Time: 03:03:21 [DEBUG]Internal behaviour: Group:
    activation at (reportExAp.xml:0):finished
Time: 03:03:21 [DEBUG]: APPLICATION: Waiting until
    finished
Time: 03:03:22 [DEBUG]: Document parsed:reportExSc.xml
Time: 03:03:22 [DEBUG]Elementary Action: Application:
    transitionTo at (reportExAp.xml:0):activating new
    scene
*******Active Scene Changed:***********
***MHEGScene***
Ref:(reportExSc.xml:0)
RunStat:false
AvailStatus:false
Info:No info
OnStartup: null
OnCloeDown: null
Items:
        ***MHEGLink***
        Ref:(reportExSc.xml:3)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Type: 8
        Source: (reportExSc.xml:0)
        Effect:
                ***MHEGAction***
```

```
                Elementary actions:
                        ***Quit***
                        Target: (reportExAp.xml:0)
        ***MHEGLink***
        Ref:(reportExSc.xml:2)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Type: 4
        Source: (reportExSc.xml:0)
        Effect:
                ***MHEGAction***
                Elementary actions:
                        ***SetValue***
                        Target: (reportExSc.xml:1)
                        New value: GenericInteger : 10000
                        ***Set Timer***
                        Target: (reportExSc.xml:0)
                        Tag: GenericInteger: 1
                        value: GenericInteger: Indirect Ref: (reportExSc.xml:1)
                        Absolute time: null
        ***MHEGIntegerVariable***
        Ref:(reportExSc.xml:1)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Value: 2000
******************************
Time: 03:03:22 [DEBUG]Internal behaviour: Group: Preparation at (reportExSc.xml:0):starting
Time: 03:03:22 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:3):starting
Time: 03:03:22 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:3):starting
Time: 03:03:22 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:3):finished
Time: 03:03:22 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:3):finished
Time: 03:03:22 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:2):starting
Time: 03:03:22 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:2):starting
Time: 03:03:22 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:2):finished
```

```
Time: 03:03:22 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:2):finished
Time: 03:03:22 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:1):starting
Time: 03:03:22 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:1):starting
Time: 03:03:22 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:1):finished
Time: 03:03:22 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:1):finished
Time: 03:03:22 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:0):starting
Time: 03:03:22 [DEBUG]Internal behaviour: Root:Content Preparation at (reportExSc.xml:0):called
Time: 03:03:22 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:0):finished
Time: 03:03:22 [DEBUG]Internal behaviour: Group: Preparation at (reportExSc.xml:0):finished
Time: 03:03:22 [DEBUG]Internal behaviour: Group: activation at (reportExSc.xml:0):starting
Time: 03:03:22 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:0):starting
Time: 03:03:22 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:0):finished
Time: 03:03:23 [DEBUG]Internal behaviour: Link: activation at (reportExSc.xml:3):starting
Time: 03:03:23 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:3):starting
Time: 03:03:23 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:3):finished
Time: 03:03:23 [DEBUG]Internal behaviour: Link: activation at (reportExSc.xml:3):finished
Time: 03:03:23 [DEBUG]Internal behaviour: Link: activation at (reportExSc.xml:2):starting
Time: 03:03:23 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:2):starting
Time: 03:03:23 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:2):finished
Time: 03:03:23 [DEBUG]Internal behaviour: Link: activation at (reportExSc.xml:2):finished
Time: 03:03:23 [DEBUG]Internal behaviour: Variable: activation at (reportExSc.xml:1):starting
Time: 03:03:23 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:1):starting
Time: 03:03:23 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:1):finished
Time: 03:03:23 [DEBUG]Internal behaviour: Variable: activation at (reportExSc.xml:1):finished
Time: 03:03:23 [DEBUG]Event: Event fired: isrunning
```

```
Time: 03:03:23 [DEBUG]Event: Event isrunning is in
    queue
Time: 03:03:23 [DEBUG]Internal behaviour: Group:
    activation at (reportExSc.xml:0):finished
Time: 03:03:23 [DEBUG]Elementary Action: Application:
    transitionTo at (reportExAp.xml:0):finished
Time: 03:03:23 [DEBUG]Execution: EXEC_QUEUE: Preparing
    to execute next action
Time: 03:03:23 [DEBUG]Execution: EXEC_QUEUE: Found
    action fi.hut.tml.xsmiles.mlfc.mheg5.objectmodel.
    elact.SetVariable
Time: 03:03:23 [DEBUG]Elementary Action:
    IntegerVariable:SetValue at (reportExSc.xml:1):
    executing
Time: 03:03:23 [DEBUG]Elementary Action: Variable:
    SetValue at (reportExSc.xml:1):executed
Time: 03:03:23 [DEBUG]Execution: EXEC_QUEUE: Preparing
    to execute next action
Time: 03:03:23 [DEBUG]Execution: EXEC_QUEUE: Found
    action fi.hut.tml.xsmiles.mlfc.mheg5.objectmodel.
    elact.SetTimer
Time: 03:03:23 [DEBUG]Elementary Action: Scene:SetTimer
     at (reportExSc.xml:0):started with tag:1 val
    : 10000 absTime:false
Time: 03:03:23 [DEBUG]Elementary Action: Scene:SetTimer
     at (reportExSc.xml:0):Scheduled
Time: 03:03:23 [DEBUG]Execution: EXEC_QUEUE: Waiting
Time: 03:03:33 [DEBUG]Internal behaviour: Scene.Timer1
    at (reportExSc.xml:0):Fired
Time: 03:03:33 [DEBUG]Event: Async event fired:
    timerfired
Time: 03:03:33 [DEBUG]Execution: EXEC_QUEUE: Restarting
Time: 03:03:33 [DEBUG]Execution: EXEC_QUEUE: Preparing
    to execute next action
Time: 03:03:33 [DEBUG]Execution: EXEC_QUEUE: Found
    action fi.hut.tml.xsmiles.mlfc.mheg5.objectmodel.
    elact.Quit
Time: 03:03:33 [DEBUG]Elementary Action: Application:
    quit at (reportExAp.xml:0):finishing application
Time: 03:03:33 [DEBUG]Elementary Action: Application:
    quit at (reportExAp.xml:0):destroying active scene
Time: 03:03:33 [DEBUG]Internal behaviour: Group:
    destruction at (reportExSc.xml:0):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:1):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:1):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:1):finished
```

```
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:1):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:2):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:2):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:2):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:2):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:2):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:2):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:3):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:3):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:3):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:3):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:3):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:3):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:0):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Group:
    deactivation at (reportExSc.xml:0):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:1):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:2):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:2):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:2):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:3):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:3):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:3):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:0):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:0):finished
```

```
Time: 03:03:33 [DEBUG]Internal behaviour: Group: deactivation at (reportExSc.xml:0):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Root: destruction at (reportExSc.xml:0):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Group: destruction at (reportExSc.xml:0):finished
Time: 03:03:33 [DEBUG]Elementary Action: Application: quit at (reportExAp.xml:0):destroying active application
Time: 03:03:33 [DEBUG]Internal behaviour: Group: destruction at (reportExAp.xml:0):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root: destruction at (reportExAp.xml:1):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Link: deactivation at (reportExAp.xml:1):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root: deactivation at (reportExAp.xml:1):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Link: deactivation at (reportExAp.xml:1):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Root: destruction at (reportExAp.xml:1):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Root: destruction at (reportExAp.xml:0):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Application: deactivation at (reportExAp.xml:0):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Group: deactivation at (reportExAp.xml:0):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Link: deactivation at (reportExAp.xml:1):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root: deactivation at (reportExAp.xml:1):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Link: deactivation at (reportExAp.xml:1):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Root: deactivation at (reportExAp.xml:0):starting
Time: 03:03:33 [DEBUG]Internal behaviour: Root: deactivation at (reportExAp.xml:0):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Group: deactivation at (reportExAp.xml:0):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Application: deactivation at (reportExAp.xml:0):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Root: destruction at (reportExAp.xml:0):finished
Time: 03:03:33 [DEBUG]Internal behaviour: Group: destruction at (reportExAp.xml:0):finished
Time: 03:03:33 [DEBUG]Elementary Action: Application: quit at (reportExAp.xml:0):waking up main thread
Time: 03:03:33 [DEBUG]: APPLICATION: Finished
Application Finished
```

```
APPLICATION TREE
***MHEGApplication***
Ref:(reportExAp.xml:0)
RunStat:false
AvailStatus:false
Info:No info
OnStartup: null
OnCloeDown: null
Items:
        ***MHEGLink***
        Ref:(reportExAp.xml:1)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Type: 4
        Source: (reportExAp.xml:0)
        Effect:
                ***MHEGAction***
                Elementary actions:
                        ***TransitionTo***
                        Target: (reportExSc.xml:0)
                        TransEffect: null
                        ConnTag: null
OnSpawnCloseDown: null
OnRestart: null
FINISHED: true
SCENE TREE
***MHEGScene***
Ref:(reportExSc.xml:0)
RunStat:false
AvailStatus:false
Info:No info
OnStartup: null
OnCloeDown: null
Items:
        ***MHEGLink***
        Ref:(reportExSc.xml:3)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Type: 8
        Source: (reportExSc.xml:0)
        Effect:
                ***MHEGAction***
                Elementary actions:
                        ***Quit***
                        Target: (reportExAp.xml:0)
```

```
        ***MHEGLink***
        Ref:(reportExSc.xml:2)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Type: 4
        Source: (reportExSc.xml:0)
        Effect:
                ***MHEGAction***
                Elementary actions:
                        ***SetValue***
                        Target: (reportExSc.xml:1)
                        New value: GenericInteger
                            : 10000
                        ***Set Timer***
                        Target: (reportExSc.xml:0)
                        Tag: GenericInteger: 1
                        value: GenericInteger: Indirect
                            Ref: (reportExSc.xml:1)
                        Absolute time: null
        ***MHEGIntegerVariable***
        Ref:(reportExSc.xml:1)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Value: 10000
```

E.3 Engine output for Section 5.3.4 test

```
Time: 02:34:16 [DEBUG]: Parsing document: reportExAp.
    xml
Time: 02:34:16 [DEBUG]: Parsing document: Using custom
    parser
Time: 02:34:18 [DEBUG]: Document parsed:reportExAp.xml
APPLICATION TREE
***MHEGApplication***
Ref:(reportExAp.xml:0)
RunStat:false
AvailStatus:false
Info:No info
OnStartup: null
OnCloeDown: null
Items:
        ***MHEGLink***
        Ref:(reportExAp.xml:1)
        RunStat:false
        AvailStatus:false
        InitActive: true
```

```
        Shared: false
        Type: 4
        Source: (reportExAp.xml:0)
        Effect:
                ***MHEGAction***
                Elementary actions:
                        ***TransitionTo***
                        Target: (reportExSc.xml:0)
                        TransEffect: null
                        ConnTag: null
OnSpawnCloseDown: null
OnRestart: null
FINISHED: false
Time: 02:34:19 [DEBUG]: APPLICATION: Running
Time: 02:34:19 [DEBUG]Internal behaviour: Group:
   activation at (reportExAp.xml:0):starting
Time: 02:34:19 [DEBUG]Internal behaviour: Root:
   activation at (reportExAp.xml:0):starting
Time: 02:34:19 [DEBUG]Internal behaviour: Group:
   Preparation at (reportExAp.xml:0):starting
Time: 02:34:19 [DEBUG]Internal behaviour: Root:
   Preparation at (reportExAp.xml:1):starting
Time: 02:34:19 [DEBUG]Internal behaviour: Ingredient:
   content preparation at (reportExAp.xml:1):starting
Time: 02:34:19 [DEBUG]Internal behaviour: Ingredient:
   content preparation at (reportExAp.xml:1):finished
Time: 02:34:19 [DEBUG]Internal behaviour: Root:
   Preparation at (reportExAp.xml:1):finished
Time: 02:34:19 [DEBUG]Internal behaviour: Root:
   Preparation at (reportExAp.xml:0):starting
Time: 02:34:19 [DEBUG]Internal behaviour: Root:Content
   Preparation at (reportExAp.xml:0):called
Time: 02:34:19 [DEBUG]Internal behaviour: Root:
   Preparation at (reportExAp.xml:0):finished
Time: 02:34:19 [DEBUG]Internal behaviour: Group:
   Preparation at (reportExAp.xml:0):finished
Time: 02:34:19 [DEBUG]Internal behaviour: Root:
   activation at (reportExAp.xml:0):finished
Time: 02:34:19 [DEBUG]Internal behaviour: Link:
   activation at (reportExAp.xml:1):starting
Time: 02:34:19 [DEBUG]Internal behaviour: Root:
   activation at (reportExAp.xml:1):starting
Time: 02:34:19 [DEBUG]Internal behaviour: Root:
   activation at (reportExAp.xml:1):finished
Time: 02:34:19 [DEBUG]Event: Added DOM listener to (
   reportExAp.xml:0) for event: 4
Time: 02:34:19 [DEBUG]Internal behaviour: Link:
   activation at (reportExAp.xml:1):finished
```

```
Time: 02:34:19 [DEBUG]Event: Link for: MHEG Event: 4(src: (reportExAp.xml:0) type: 4 data: null) Fired.
Time: 02:34:19 [DEBUG]Event: Link for: MHEG Event: 4(src: (reportExAp.xml:0) type: 4 data: null) Executing.
Time: 02:34:19 [DEBUG]Elementary Action: Application: transitionTo at (reportExAp.xml:0):started
Time: 02:34:19 [DEBUG]Elementary Action: Application: transitionTo at (reportExAp.xml:0):deactivating non shared application objects
Time: 02:34:19 [DEBUG]Internal behaviour: Link: deactivation at (reportExAp.xml:1):starting
Time: 02:34:19 [DEBUG]Event: Removing DOM listener to (reportExAp.xml:0) for event: 4
Time: 02:34:19 [DEBUG]Internal behaviour: Root: deactivation at (reportExAp.xml:1):starting
Time: 02:34:19 [DEBUG]Internal behaviour: Root: deactivation at (reportExAp.xml:1):finished
Time: 02:34:19 [DEBUG]Internal behaviour: Link: deactivation at (reportExAp.xml:1):finished
Time: 02:34:19 [DEBUG]Elementary Action: Application: transitionTo at (reportExAp.xml:0):deactivating current scene
Time: 02:34:19 [DEBUG]Elementary Action: Application: transitionTo at (reportExAp.xml:0):parsing new scene
Time: 02:34:19 [DEBUG]: Parsing document: reportExSc.xml
Time: 02:34:19 [DEBUG]: Parsing document: Using custom parser
Time: 02:34:20 [DEBUG]: Document parsed:reportExSc.xml
Time: 02:34:20 [DEBUG]Elementary Action: Application: transitionTo at (reportExAp.xml:0):activating new scene
*******Active Scene Changed:***********
***MHEGScene***
Ref:(reportExSc.xml:0)
RunStat:false
AvailStatus:false
Info:No info
OnStartup: null
OnCloeDown: null
Items:
        ***MHEGLink***
        Ref:(reportExSc.xml:3)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Type: 8
```

```
        Source: (reportExSc.xml:0)
        Effect:
                ***MHEGAction***
                Elementary actions:
                        ***Quit***
                        Target: (reportExAp.xml:0)
        ***MHEGLink***
        Ref:(reportExSc.xml:2)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Type: 4
        Source: (reportExSc.xml:0)
        Effect:
                ***MHEGAction***
                Elementary actions:
                        ***SetValue***
                        Target: (reportExSc.xml:1)
                        New value: GenericInteger : 10000
                        ***Set Timer***
                        Target: (reportExSc.xml:0)
                        Tag: GenericInteger: 1
                        value: GenericInteger: Indirect Ref: (reportExSc.xml:1)
                        Absolute time: null
        ***MHEGIntegerVariable***
        Ref:(reportExSc.xml:1)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Value: 2000
******************************
Time: 02:34:20 [DEBUG]Internal behaviour: Group: Preparation at (reportExSc.xml:0):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:3):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:3):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:3):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:3):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:2):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:2):starting
```

```
Time: 02:34:20 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:2):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:2):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:1):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:1):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Ingredient: content preparation at (reportExSc.xml:1):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:1):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:0):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Root:Content Preparation at (reportExSc.xml:0):called
Time: 02:34:20 [DEBUG]Internal behaviour: Root: Preparation at (reportExSc.xml:0):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Group: Preparation at (reportExSc.xml:0):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Group: activation at (reportExSc.xml:0):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:0):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:0):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Link: activation at (reportExSc.xml:3):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:3):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:3):finished
Time: 02:34:20 [DEBUG]Event: Added DOM listener to (reportExSc.xml:0) for event: 8
Time: 02:34:20 [DEBUG]Internal behaviour: Link: activation at (reportExSc.xml:3):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Link: activation at (reportExSc.xml:2):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:2):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:2):finished
Time: 02:34:20 [DEBUG]Event: Added DOM listener to (reportExSc.xml:0) for event: 4
Time: 02:34:20 [DEBUG]Internal behaviour: Link: activation at (reportExSc.xml:2):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Variable: activation at (reportExSc.xml:1):starting
```

```
Time: 02:34:20 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:1):starting
Time: 02:34:20 [DEBUG]Internal behaviour: Root: activation at (reportExSc.xml:1):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Variable: activation at (reportExSc.xml:1):finished
Time: 02:34:20 [DEBUG]Event: Link for: MHEG Event: 4(src: (reportExSc.xml:0) type: 4 data: null) Fired.
Time: 02:34:20 [DEBUG]Event: Link for: MHEG Event: 4(src: (reportExSc.xml:0) type: 4 data: null) Executing.
Time: 02:34:20 [DEBUG]Elementary Action: IntegerVariable:SetValue at (reportExSc.xml:1): executing
Time: 02:34:20 [DEBUG]Elementary Action: Variable: SetValue at (reportExSc.xml:1):executed
Time: 02:34:20 [DEBUG]Elementary Action: Scene:SetTimer at (reportExSc.xml:0):started with tag:1 val: 10000 absTime:false
Time: 02:34:20 [DEBUG]Elementary Action: Scene:SetTimer at (reportExSc.xml:0):Scheduled
Time: 02:34:20 [DEBUG]Internal behaviour: Group: activation at (reportExSc.xml:0):finished
Time: 02:34:20 [DEBUG]Elementary Action: Application: transitionTo at (reportExAp.xml:0):finished
Time: 02:34:20 [DEBUG]Internal behaviour: Group: activation at (reportExAp.xml:0):finished
Time: 02:34:20 [DEBUG]: APPLICATION: Waiting until finished
Time: 02:34:30 [DEBUG]Internal behaviour: Scene.Timer1 at (reportExSc.xml:0):Fired
Time: 02:34:30 [DEBUG]Event: Link for: MHEG Event: 8(src: (reportExSc.xml:0) type: 8 data: 1) Fired.
Time: 02:34:30 [DEBUG]Event: Link for: MHEG Event: 8(src: (reportExSc.xml:0) type: 8 data: 1) Executing.
Time: 02:34:30 [DEBUG]Elementary Action: Application: quit at (reportExAp.xml:0):finishing application
Time: 02:34:30 [DEBUG]Elementary Action: Application: quit at (reportExAp.xml:0):destroying active scene
Time: 02:34:30 [DEBUG]Internal behaviour: Group: destruction at (reportExSc.xml:0):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Root: destruction at (reportExSc.xml:1):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Root: deactivation at (reportExSc.xml:1):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Root: deactivation at (reportExSc.xml:1):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Root: destruction at (reportExSc.xml:1):finished
```

```
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:2):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:2):starting
Time: 02:34:30 [DEBUG]Event: Removing DOM listener to (
    reportExSc.xml:0) for event: 4
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:2):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:2):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:2):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:2):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:3):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:3):starting
Time: 02:34:30 [DEBUG]Event: Removing DOM listener to (
    reportExSc.xml:0) for event: 8
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:3):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:3):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:3):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:3):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    destruction at (reportExSc.xml:0):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Group:
    deactivation at (reportExSc.xml:0):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:1):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:2):starting
Time: 02:34:30 [DEBUG]Event: Removing DOM listener to (
    reportExSc.xml:0) for event: 4
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:2):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:2):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Link:
    deactivation at (reportExSc.xml:3):starting
Time: 02:34:30 [DEBUG]Event: Removing DOM listener to (
    reportExSc.xml:0) for event: 8
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    deactivation at (reportExSc.xml:3):starting
```

```
Time: 02:34:30 [DEBUG]Internal behaviour: Link: deactivation at (reportExSc.xml:3):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Root: deactivation at (reportExSc.xml:0):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Root: deactivation at (reportExSc.xml:0):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Group: deactivation at (reportExSc.xml:0):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Root: destruction at (reportExSc.xml:0):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Group: destruction at (reportExSc.xml:0):finished
Time: 02:34:30 [DEBUG]Elementary Action: Application: quit at (reportExAp.xml:0):destroying active application
Time: 02:34:30 [DEBUG]Internal behaviour: Group: destruction at (reportExAp.xml:0):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Root: destruction at (reportExAp.xml:1):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Link: deactivation at (reportExAp.xml:1):starting
Time: 02:34:30 [DEBUG]Event: Removing DOM listener to (reportExAp.xml:0) for event: 4
Time: 02:34:30 [DEBUG]Internal behaviour: Root: deactivation at (reportExAp.xml:1):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Link: deactivation at (reportExAp.xml:1):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Root: destruction at (reportExAp.xml:1):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Root: destruction at (reportExAp.xml:0):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Application: deactivation at (reportExAp.xml:0):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Group: deactivation at (reportExAp.xml:0):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Link: deactivation at (reportExAp.xml:1):starting
Time: 02:34:30 [DEBUG]Event: Removing DOM listener to (reportExAp.xml:0) for event: 4
Time: 02:34:30 [DEBUG]Internal behaviour: Root: deactivation at (reportExAp.xml:1):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Link: deactivation at (reportExAp.xml:1):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Root: deactivation at (reportExAp.xml:0):starting
Time: 02:34:30 [DEBUG]Internal behaviour: Root: deactivation at (reportExAp.xml:0):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Group: deactivation at (reportExAp.xml:0):finished
```

```
Time: 02:34:30 [DEBUG]Internal behaviour: Application:
    deactivation at (reportExAp.xml:0):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Root:
    destruction at (reportExAp.xml:0):finished
Time: 02:34:30 [DEBUG]Internal behaviour: Group:
    destruction at (reportExAp.xml:0):finished
Time: 02:34:30 [DEBUG]Elementary Action: Application:
    quit at (reportExAp.xml:0):waking up main thread
Time: 02:34:30 [DEBUG]: APPLICATION: Finished
Application Finished
APPLICATION TREE
***MHEGApplication***
Ref:(reportExAp.xml:0)
RunStat:false
AvailStatus:false
Info:No info
OnStartup: null
OnCloeDown: null
Items:
        ***MHEGLink***
        Ref:(reportExAp.xml:1)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Type: 4
        Source: (reportExAp.xml:0)
        Effect:
                ***MHEGAction***
                Elementary actions:
                        ***TransitionTo***
                        Target: (reportExSc.xml:0)
                        TransEffect: null
                        ConnTag: null
OnSpawnCloseDown: null
OnRestart: null
FINISHED: true
SCENE TREE
***MHEGScene***
Ref:(reportExSc.xml:0)
RunStat:false
AvailStatus:false
Info:No info
OnStartup: null
OnCloeDown: null
Items:
        ***MHEGLink***
        Ref:(reportExSc.xml:3)
        RunStat:false
```

```
        AvailStatus:false
        InitActive: true
        Shared: false
        Type: 8
        Source: (reportExSc.xml:0)
        Effect:
                ***MHEGAction***
                Elementary actions:
                        ***Quit***
                        Target: (reportExAp.xml:0)
        ***MHEGLink***
        Ref:(reportExSc.xml:2)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Type: 4
        Source: (reportExSc.xml:0)
        Effect:
                ***MHEGAction***
                Elementary actions:
                        ***SetValue***
                        Target: (reportExSc.xml:1)
                        New value: GenericInteger
                           : 10000
                        ***Set Timer***
                        Target: (reportExSc.xml:0)
                        Tag: GenericInteger: 1
                        value: GenericInteger: Indirect
                            Ref: (reportExSc.xml:1)
                        Absolute time: null
        ***MHEGIntegerVariable***
        Ref:(reportExSc.xml:1)
        RunStat:false
        AvailStatus:false
        InitActive: true
        Shared: false
        Value: 10000
Time: 02:34:30 [DEBUG]Event: Link for: MHEG Event: 8(
   src: (reportExSc.xml:0) type: 8 data: 1) Finished
   and notifing next thread
```

Bibliography

[1] *The Arena Web Browser*. Accessed at May 2001.
Available via web: http://www.yggdrasil.com/Products/Arena/.

[2] Michael Baentsch and Peter Rösch. Weaving interactive media into the web: The www-glass gateway. In *Third international World-Wide Web Conference, Workshop D: Interactive and Distributed Multi-Media System on Highspeed Networks*, April 1995.

[3] R. A. Bissell and A. Eales. The set-top box for interactive purposes. *BT Technol J*, 13(4):66–77, October 1995.

[4] Dr. Thomas J. Casey. *MHEG-5 technical guide*. BSI, for final review prior to publication edition, January 1998.

[5] C. Dobbyn, D. Shrimpton, and T. Casey. Models of convergence between the world wide web and interactive television using mheg-5, March 1999.

[6] Marica Ehiffre, Claudio Marchisio, Pietro Marchisio, Paolo Panicciari, and Silvia Del Rossi. Mheg-5 — aims, concepts, and implementation issues. *IEEE Multimedia*, pages 84–91, January–March 1998.

[7] Marica Ehiffre and Pietro Marchisio. A multimedia presentation system for mheg-5 applications. *Computer Standards & Interfaces*, 20(4–5):375–287, Februrary 1999.

[8] L. Geyer, M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. Mheg in java - integrating a multimedia standard into the web. In *Poster Proceedings 6th International World-Wide Web Conference*, April 1997.

[9] Mark Grand. *Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML*, volume first. Wiley, 1998.

[10] Audio-Video Transport Working Group. *RFC 1889: RTP: A transport protocol for real-time applications.* IETF organisation, January 1996. Available via web: http://www.ietf.org/rfc/rfc1889.txt.

[11] Akemi Hatayama, Katsuya Shinohara, Takao Omachi, and Sachiko Kitawaki. A converter between mheg-5 and html 4.0. *IEEE Transactions on Consumer Electronics*, 45(3):732–744, August 1999.

[12] H.M.Deitel, P.J.Deitel, T.R.Nieto, T.M.Lin, and P.Sadhu. *XML How to Program.* Prentice Hall Inc., 2001.

[13] Mikko Honkala. *X-Smiles 0.32: Tehnical Specification, version 1.7.* X-Smiles.org. Last document update: April 2001. Available via web: http://www.xsmiles.org/TechSpec/TechSpecStyled.htm.

[14] International Organisation for Standardization and International Electrotechnical Commision (ISO/IEC). *ISO/IEC IS 13522-5: Coding of multimedia and hypermedia information — Part 5: Support for base-level interactive applications*, first edition, April 1997.

[15] International Organisation for Standardization and International Electrotechnical Commision (ISO/IEC). *ISO/IEC CD 13522-8: Coding of multimedia and hypermedia information — Part 8: XML Notation of ISO/IEC 13522-5 (MHEG XML)*, working draft edition, December 1999.

[16] Eckhart Kppen and Gustaf Neumann. A practical approach towards active hyperlinked documents. *Computer Networks and ISDN Systems*, 30(1–7):251–258, April 1998.

[17] T. Berners Lee, R. Fielding, U.C. Irvine, and L. Masinter. *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*. IETF organisation, August 1998. Available via web:
http://www.ietf.org/rfc/rfc2396.txt.

[18] Mark McManus. Dynamic navigation with dom. taking advantage of the document object model. *Web techniques magazine*, 3(3), March 1998.

[19] *The mnemonic project*. Accessed at May 2001.
Available via web: http://www.mnemonic.org/.

[20] The mozilla organization. *Mozilla browser*. Accessed at May 2001.
Available via web: http://www.mozilla.org/.

[21] NCSA. *NCSA Mosaic*. Accessed at: May 2001. Accessed at May 2001.
Available via web: http://archive.ncsa.uiuc.edu/SDG/Software/Mosaic.

[22] SUN microsystems. *HotJava TM Browser Product Family*. Accessed at May 2001.
Available via web: http://java.sun.com/products/hotjava/index.html.

[23] Vedhagiri Valliappan, David Shrimpton, Chris Dobbyn, and Tom Casey. Transforming web pages for interactive tv using xsl. In *ICME 2001, IEEE International conference on Multimedia and Expo, Tokyo*, August 2001.

[24] W3C (MIT, INRIA, Keio). *Amaya W3C's editor/browser*. Accessed at May 2001.
Available via web: http://www.w3.org/Amaya/.

[25] W3C (MIT, INRIA, Keio). *REC-smil-19980615: Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*, first edition, June 1998. Available via web: http://www.w3.org/TR/REC-smil/.

[26] W3C (MIT, INRIA, Keio). *XSL Transformations (XSLT) Version 1.0 Recommendation*, November 1999. Available via web: http://www.w3.org/TR/xslt/.

[27] W3C (MIT, INRIA, Keio). *Document Object Model (DOM) Level 2 Core Specification*, November 2000. Available via web: http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/.

[28] W3C (MIT, INRIA, Keio). *Document Object Model (DOM) Level 2 Events Specification*, first edition, November 2000. http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/.

[29] W3C (MIT, INRIA, Keio). *Document Object Model (DOM) Level 2 Views Specification*, first edition, November 2000. http://www.w3.org/TR/2000/REC-DOM-Level-2-Views-20001113/.

[30] W3C (MIT, INRIA, Keio). *Extensible Markup Language (XML) 1.0 recommendation*, second edition, October 2000. Available via web: http://www.w3.org/TR/2000/REC-xml-20001006.

[31] W3C (MIT, INRIA, Keio). *XML Protocol Abstract Model working draft*, July 2001. Available via web: http://www.w3.org/TR/2001/WD-xmlp-am-20010709/.

[32] X-Smiles.org. *X-Smiles, an open XML browser for exotic devices.* Accessed at May 2001.
Available via web: http://www.xsmiles.org/.