System Architecture of WebSmart - A Web-Oriented Synchronized Multimedia Authoring System

by
Yue Xu
B.S., FuDan University, 1991

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
Master of Science
in
THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

we accept this thesis as conforming
to the required standard

The University of British Columbia
April 1998
© Yue Xu, 1998
In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Computer Science Department

The University of British Columbia

2366 Main Mall

Vancouver, BC

Canada V6T 1Z4

Date: \[\text{Apr. } xx, 1998\]
Abstract

Embedding multimedia contents such as audio and video into a Web page promotes wide applications on the Internet. However, to present web-oriented interactive multimedia raises extra requirements on network bandwidth, synchronization mechanisms, presentation engines, and authoring environments. Moreover, the heterogeneity of the Internet environment calls for open applications to fulfill these requirements seamlessly on various platforms.

This paper proposes an approach to create Web-oriented synchronized multimedia documents by integrating the latest development in Java, Java Media Framework, JavaScript and HTML. As the result a WebSmart system is produced. It includes a presentation engine as well as an authoring environment, and it also realizes two levels of inter-object synchronization. Designed using an Object-Oriented methodology and implemented in Java, it aims to be a platform-neutral application.

This paper introduces the system architecture and the technology framework of WebSmart system. Performance analysis and future work are also presented at the end of the paper.
## Contents

Abstract ........................................................................................................ ii

Contents ....................................................................................................... iii

List of Tables .............................................................................................. vi

List of Figures .............................................................................................. vii

Acknowledgments ....................................................................................... viii

1 Introduction .............................................................................................. 1

1.1 Problem domain ................................................................................... 1

1.2 Contributions of WebSmart ................................................................. 5

1.3 Thesis outline ....................................................................................... 7

2 WebSmart Architecture ............................................................................ 9

2.1 Functional model .................................................................................. 10

2.2 Scenario of creating a presentation ....................................................... 13

2.3 WebSmart synchronization statements ................................................. 26

2.4 Object model ........................................................................................ 29

2.4.1 Entity objects and interface objects ................................................ 29

2.4.2 Class diagram .................................................................................. 31

3 Technology Framework ............................................................................ 42

3.1 Java ...................................................................................................... 42

3.2 Java Media Framework (JMF) ............................................................... 45

3.3 HTML .................................................................................................. 49

3.4 JavaScript ............................................................................................ 50
List of Tables

TABLE 1. JMF implementations ................................................................. 45
# List of Figures

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Functional Model</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>wsEditor</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>Section window</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Object Properties window</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>Timing window</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>VideoFrame window</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>The wsEditor main window after an object Notes is added</td>
<td>21</td>
</tr>
<tr>
<td>8</td>
<td>PageStyle window</td>
<td>22</td>
</tr>
<tr>
<td>9</td>
<td>Objects Layout window</td>
<td>23</td>
</tr>
<tr>
<td>10</td>
<td>CaptionMaker window</td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>QuizDesigner window</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>Question window</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>Notation of entity objects and interface objects</td>
<td>30</td>
</tr>
<tr>
<td>14</td>
<td>An example of interface objects and entity objects</td>
<td>31</td>
</tr>
<tr>
<td>15</td>
<td>Object Model Notation</td>
<td>32</td>
</tr>
<tr>
<td>16</td>
<td>Object Model</td>
<td>41</td>
</tr>
<tr>
<td>17</td>
<td>The output HTML files of WebSmart</td>
<td>50</td>
</tr>
<tr>
<td>18</td>
<td>Control flow in WebSmart</td>
<td>52</td>
</tr>
</tbody>
</table>
Acknowledgments

I owe a great deal of gratitude to my thesis supervisor, Dr. Gerald Neufeld. It was he who led me into this interesting area, multimedia on the Internet, the area which I might take as my future professional career. He has provided with generous support as well as useful suggestions on the WebSmart project. I'm also grateful for his patience in reviewing my thesis.

My gratitude also goes to Dr. Norm Hutchinson, who kindly took the time to be the second reader of this thesis, and gave many helpful comments.

Thanks Lan Li, my partner in this project, for the effective cooperation between us. This one year’s busy but enjoyable work is a precious experience to me, and I learned a lot from her diligent researching attitude.

Finally, I wish to thank my parents and my husband for the understanding and love they are always ready for me. Every piece of my progress belongs to them.

Yue Xu

The University of British Columbia

April 1998
Chapter 1

Introduction

The advent of the Internet has greatly changed the way in which people get information and knowledge. Today more and more people are retrieving and sharing information from the World Wide Web. With the development in digital audio/video media, a new type of web page which presents multimedia content such as audio and video is emerging. Such multimedia presentations will extend or even revolutionize the educational system as education is no longer limited to the “classroom” format in which all the students and instructors are required to collect in one location for the specific period. A remote student can attend a class at his own convenience and pace. Moreover, the class will remain effective as the number of students increases. In addition to this distant learning, multimedia web presentations also have other applications such as on-line publication/advertisement, corporate training and product demonstration.

1.1 Problem domain

Multimedia, as defined in [1], has three characteristics:

- *Multiple media* (for example, audio, video, data, text, and graphics), which center on including such types as audio or video for which time is crucial;

- *Interactivity*, rather than simple one-way broadcasting;
• **Networked offerings**, as contrasted with those that stand alone.

In order for multimedia presentations to be effective, the following challenges need to be addressed:

• **Bandwidth**

The current Internet is marked by its limited bandwidth and unstable transmission. The situation will worsen when it transports voluminous video files. Most present public accesses to the Internet are via either 10-Mbps LAN/Ethernet, or typically 28.8Kbps home-use modem. On the other hand, video formats requires a high bandwidth. For example, International Standards Organizations (ISO)'s Moving Pictures Expert Group (MPEG) [2] requires 1.2 to 40 Mbps, Intel’s Digital Video Interactive (DVI) requires 1.2 to 1.8 Mbps, and International Telecommunications Union (ITU)'s H.261 [3] requires 0.064 to 2 Mbps. Even in 10-Mbps LANs which allow for several parallel compressed video streams, the performance is nondeterministic due to some mechanisms such as the *Carrier Sense Multiple Access with Collision Detection (CSMA-CD)*. This will be briefly discussed in Section 4.5 (Performance analysis).

To solve this bandwidth problem, Internet-oriented compression algorithms are being designed by companies and international standard organizations. Such algorithms include RealNetworks’ *RealSystem codec*, ISO’s *MPEG4*, and ITU’s *H.323/H.324* standards.

• **Synchronization**
Unlike a document which contains only text and image, a multimedia document which contains audio/video defines a new temporal axis along with the traditional spatial axis. Audio and video are both time-sensitive media formats. As time goes on, various components within a single document might drift in time away from each other, therefore creating an inconsistent effect to the document viewer.

Blakowski [6] defined two levels of multimedia synchronization, *intra-object* and *inter-object* synchronization. The former is concerned with the time relations within one media object, and the latter is concerned with time relations between two or more media objects.

Before the temporal relationships can be specified within a multimedia application, a model has to be set up as the formal notation to describe the synchronization activities. For this purpose several models have been created such as a timeline model in [7], a petri-net model in [8], and a Media Relation Graph in [9].

- **Presentation**

Another issue for a multimedia web presentation is the mechanism to playback the video and audio clips within a browser such as the Netscape’s Navigator and the Microsoft’s Internet Explorer. The display of text and image is static, and these elements are embedded in HTML files by special tags. However, for continuous media such as audio/video, a VCR simulated interface providing controls such as *Start/Pause, Stop, FastForward, and Rewind* is most preferable to ordinary people.
Plug-in is a widely used approach to present continuous media content within a web browser. A plug-in [11] is a separate code module that behaves as though it is part of a browser. Being a code library whose source conforms to standard C/C++ syntax, a plug-in’s file type depends on the platform, for example, a .DLL (Dynamic Link library) on Windows, or a .SO or .DSO on Unix. Plug-ins can be embedded in HTML files. When the web page viewer goes to a page that contains embedded data of a media type that invokes a plug-in, the browser usually loads the plug-in code into memory, creates and initializes a new instance of the plug-in. When the viewer leaves the page or closes the window, the instance is discarded. When the last instance of a plug-in is deleted, the plug-in code is unloaded from memory. Different vendors provide plug-ins according to different APIs, some widely-used multimedia plug-ins includes RealNetwork’s RealAudio, RealVideo [22], and Apple’s QuickTime [12].

In this thesis, however, we’ll introduce another multimedia presentation approach which leverages the recent progress in Java APIs and which provides a standard interface to presentation engine designers.

- **Authoring environment**

The Hypertext Markup Language (HTML) [10] defines the layout and spatial relationship of components within a text-based web document. When it comes to the multimedia documents where a temporal relationship exists as well as the spatial relationship, more functionality is needed in assembling and synchronizing the multi-components of a page. In addition, as WWW is used by more non-professionally people, the authoring platform is required to be easy-to-use as well as versatile.
Authoring Systems, as defined in [13], are “software tools that enable users to create interactive instruction without programming, thereby allowing those who lack either access to programmers, time to program, or interest in learning programming to engage in courseware development.” Unlike programming or authoring languages such as C or HTML which requires coding, authoring systems provide on-screen tools to help users enter content of the document and organize the layout.

Besides the basic authoring functions described above, a multimedia authoring environment should also provide a mechanism to set up the synchronization events and define interactions between a web page and its viewers. Currently two complementary methods are being used, scripting language and declarative language. Examples for scripting language are JavaScript [17], VisualBasic [18] and Macromedia’s Lingo [19], while examples for declarative language are HTML, QuickTime and SMIL [24].

From the above discussion, we conclude that supporting integrated multimedia documentation on the web requires more than a suitable network with sufficient bandwidth. Synchronization models, presentation engines, and authoring environments, are all among the new techniques to be developed.

1.2 Contributions of WebSmart

WebSmart, as its name implies, is a Web-oriented synchronized multimedia authoring system. It was designed to provide the following functions:

- **A presentation engine** to playback the video content within a web browser without extra plug-ins. WebSmart uses a light-weight Java Applet to achieve the following functions:
• Play back the continuous media from network.

• Control the presentation of the documents.

• Synchronize the components.

• Tune the performance according to the available bandwidth.

• **An authoring environment** to create the multimedia web documents. It provides the following functions:
  
  • The document created is composed of several components that are accessible via a URL, e.g. files stored on a Web server.
  
  • The components have different media types, such as audio, video, image or text.
  
  • The synchronization among components can be generated in a visual way in the authoring environment.
  
  • Interactions between the document and its viewer can be pre-set.
  
  • The layout of the document page is easy to handle. Users can set the position and size of each component.

The major contributions of WebSmart are:

• It creates a simple document model for web-oriented multimedia presentations. This model uses a distant learning scenario, and embeds components such as *Table of Content, Media* and *accompanying HTML pages* which are distinct in meaning and easy to be applied to other applications.
• It generates a series of authoring tools (wsEditor, CaptionMaker, and QuizDesigner), and multimedia presentations can be created in a WYSIWYG manner using these tools.

• Besides the authoring tools, it also offers alternatives for experienced web authors. A set of synchronization statements are designed for creating more flexible presentations, and these statements use existing technologies (such as JavaScript) which are well understood and do not require extra learning time.

• WebSmart provides a platform-neutral approach to create and present multimedia Web documents by integrating Java [14], Java Media Framework [16], JavaScript and HTML. The authoring system runs seamlessly on various platforms including Unix and Windows95/NT, and presentations it created are accessible from ordinary browsers without installing special plug-ins.

• With its synchronization model and algorithms, the system realized two levels of inter-object synchronization: flipping and lip sync. Moreover, the synchronization events can be created easily within the WebSmart authoring environment without extra coding in either system programming languages or scripting languages.

• The system can be easily extended due to its object-oriented design and 100% Java implementation.

1.3 Thesis outline

This thesis focuses on the WebSmart architecture and its authoring environment, with a brief description on the synchronization mechanism. For detailed synchronization
models and algorithms, see [20].

The rest of the thesis is organized as follows: Chapter 2 gives a detailed description of the system architecture, Chapter 3 introduces the technology framework of this project, Chapter 4 analyzes the performance of the system, Chapter 5 reviews the related work, finally Chapter 6 draws the conclusions and proposes for future work.
Chapter 2

WebSmart Architecture

To manage the difficulty in the development process and extend the flexibility of the system, WebSmart adopted an object-oriented design methodology. Object-Oriented design divides the complexity of the system into smaller modules with interfaces defined clearly among them. This helps different members in a development team to coordinate and communicate with each other. Furthermore, the multimedia authoring is a new area and developing quickly, this requires its architecture to support easy extension. Object-Oriented designing models software systems as collections of cooperating objects which communicate with each other by passing messages. Compared with other design methods such as Top-down structured design and Data-driven design, the Object-Oriented design adapts more distinctly to extensions by simply adding in new collections of objects.

This chapter discusses the architecture of the system in a top-down sequence. Section 2.1 analyzes the high level functionality of the system with regard to the requirements for multimedia authoring systems (refer to Chapter 1), Section 2.2 complements Section 2.1 with a scenario of creating a WebSmart multimedia presentation. Section 2.3 introduces WebSmart-defined synchronization event API which provides an alternative to the scenario described in Section 2.2. Section 2.4 addresses the low level implementation of the system using an object model, and refines each functional
module defined in Section 2.1.

2.1 Functional model

WebSmart was primarily designed for distant learning applications. However, it can also be applied to other applications such as corporate training and on-line publication/advertisement. Each presentation generated by WebSmart consists of three types of components:

- **Table of Contents (TOC)** outlines the major topics of a presentation. Just as a speech or a lecture is composed of several parts, a WebSmart presentation is composed of several *sections*, where each section is composed of one or more video clips. After creating the outline of a presentation, authors can select hiding or showing the TOC on the generated web page.

- **Media (Audio/Video)** presents the audio/video content. It can not be hidden in the web page created, because synchronization events taking place in other components of the page are all driven by this Media object. It is feasible to include more than one Media object within a WebSmart presentation and present audio/video streams in these Media objects in a synchronized way. In the current distant learning application we built (refer to Section 4.3), there is only one speaker (the instructor) in the class, therefore the web page includes only one Media object.

- **Accompanying HTML pages** include all the other objects which simulate the overhead and notes in a real classroom course. Located in neighboring frames in a web page along with the TOC and Media, these HTML pages can be flipped as the presentation video is going on, therefore creating a virtual classroom’s atmosphere.
The whole system (Figure 1) is made up of an authoring environment and a presentation engine. The authoring environment creates a set of HTML files for a presentation. The HTML files are transmitted to the Web browser at the client site. The media content embedded within the HTML files is presented by a Java Applet (Display Applet) which is also migrated to the client site.

The authoring environment consists of four parts: synchronization mechanism, layout module, QuizDesigner and wsEditor.

WebSmart supports two levels of inter-object synchronization, namely flipping pages and lip synchronization. The former one is a form of interaction described as “15 seconds into video-clip 1, show slide_1 in the Notes Frame and title_1 in the title Frame; ... 37 seconds into video-clip 2, show slide_5...”. The latter one realizes a finer-grained form of synchronization. The Media frame in a WebSmart page includes an optional text field which displays the closed caption of the audio/video content. As the media clip is being presented, the closed caption field changes its text in a pace corresponding to the movement of the narrator’s lips. This closed caption feature enables deaf and hard of hearing people to watch a multimedia-embedded web page, improves people’s reading scores and helps them understand/learn foreign languages. Closed captions are signals that arrive with video/audio. They require a special decoder to make them visible, and can be turned on or off by the user.

The layout module provides a visual tool for authors to arrange the layout of exported web pages. Authors can add/delete multimedia components on the page, change sizes and positions of these components, hide/restore components from the page. All
these editing actions are done visually in the \textit{wsEditor}. When authors export the multi-components setting, a multi-framed web page will be created in a set of HTML files.

The \textit{QuizDesigner} creates a kind of HTML files with more intensive interactions than ordinary pages. Besides the \textit{flipping pages} scenario described above, the QuizDesigner realizes the scenario like "20 second into video-clip 3, pause the video till the viewer answers the question in slide_10;...", and like "if the viewer succeeded in greater than 5 of the 10 questions that contained in slide 7 to 16, display video clip 5 in the \textit{Media Frame}; otherwise display video clip 6".

The \textit{wsEditor} is the central part of the authoring environment. It collects synchronization events generated by the \textit{Sync module} and layout information obtained from the \textit{Layout module}. The \textit{QuizDesigner} creates one HTML file for each of the questions in a quiz, these HTML file names will be quoted in the \textit{wsEditor} when a \textit{quiz} section is created. After a WebSmart document is exported, the \textit{wsEditor} creates a set of HTML files to be transmitted to the web browser.

Besides the exported HTML files mentioned above, the authoring environment also creates two other formats of output. The \textit{QuizDesigner} creates a \textit {.quiz} file for each quiz. This \textit {.quiz} file is kept for future modifications of the quiz. The \textit{wsEditor} also creates a \textit {.ws} file for all the sections, events and layout information within a WebSmart presentation.
2.2 Scenario of creating a presentation

WebSmart is designed to be an authoring environment which adapts to different levels of web page designers. An author without any programming experience can create his
presentation exclusively within the system. The complete process includes entering sections and their video clips, inserting accompanying HTML frames and their synchronization events, designing the page layout, entering caption text and setting caption pace, finally designing the questions, if there are any. An author with HTML experience can build his own multi-framed web page for more dynamic layout. An author with JavaScript experience can create his own synchronization mechanisms using a simple form of event API defined by WebSmart. The HTML pages displayed in accompanying frames can be created in most HTML editors such as Netscape's Communicator.

This section illustrates the steps of creating a multimedia presentation. The next section will describe the event API provided by WebSmart.

1. Start the Editor

The wsEditor opens a window as depicted in Figure 2. The two default frames in the initial window are Table of Contents (left) and Media (right).
2. Add in Sections

To add video/audio clips into the document, authors must set up section(s) in the "Table of Contents" to include these clips. However, the "Table of Contents" Panel can be hidden from Object Layout in the future. The information of a section is added in the Section window as depicted in Figure 3.
If you prefer using audio-only files rather than video files for the presentation, you can put the audio-only files in the "Video Files" field, and leave the "Audio Files" field empty.

Audio-only and image files can act as alternatives to the video files when the network bandwidth is insufficient (refer to Chapter 4 Performance and Future Work).

Caption files can be created by the CaptionMaker (refer to step 8).
• If you have more than one video/audio clip for a section, use ";" to separate the clips. For example, you can have the following entries:

Video Files: movies/sec1.1.mpg; movies/sec1.2.mpg; movies/sec1.3.mpg

Audio Files: movies/sec1.1.mpa; movies/sec1.3.mpa

Image Files: movies/sec1.gif; movies/sec1.3.gif

In this case, three video clips (sec1.1.mpg, sec1.2.mpg, and sec1.3.mpg) will be sequentially played in this section. Notice, the second video file does not have a corresponding audio-only file.

3. Add in Objects

Objects are accompanying components that go along with a Media component. When exported, these objects become frames in a web page. Under a distant learning context, these objects can act as course notes and images which will be flipped as the video is presented in the Media frame; therefore the whole page simulates a live classroom environment.

By selecting the “Add Object” item from the Object menu in the wsEditor (Figure 2), authors can enter object information in the “Object Properties” window (Figure 4). There are three types of objects that can be added, namely, Video/Audio (for creating Media frame), HTML (for creating accompanying components for the video/audio), and Table of Contents. The Layout item allows authors to choose the exported web page style, whether the multi-components of a web page are put into frames or layers. The Frame is the layout style used widely in most current web
pages, and *Layer* is a new technique defined in Dynamic HTML (DHTML) which has not been supported in most commercial browsers. The DHTML will be discussed in detail at Chapter 4 (Performance and Future Work).

![Object Properties window](image)

**FIGURE 4. Object Properties window**

The "Timing" window (Figure 5) is popped out after the "Timing" button in the "Object Properties" window is clicked. The Timing window requires information describing the synchronization events taking place in the current object. The three entries in the first row of the "Timing" window in Figure 5 set up a *flipping page* event for the *Notes* object defined in Figure 4. The event is to display *slide1.html* in the *Notes* frame when the *movies/sec1.mpg* is presented to the 5th second in the video stream.
4. Synchronization between components

The synchronization information can be entered in the *Timing* window. It can also be generated automatically in the “VideoFrame” window. Click the “Play Clip” button in the *Video/Audio Panel* (Figure 2), or play video files for a specific section by selecting a section in the *Table of Contents* list and clicking the “Play Section” button, a *VideoFrame* window (Figure 6) is opened.

First, click the “Play” button to start the video playback, or move the slider to the desired position in the clip. When the video stream is at the point where a synchronization event is desired, click on a source URL in the corresponding object list within the *wsEditor* window (the source URL is entered in the *Timing* window), then click “Timing” button below this object. The captured synchronization event time is then updated in the object list.
Figure 7 displays the look of the wsEditor main window after an object *Notes* is added and the synchronization information is generated from the *VideoFrame* window.
5. Set the HTML page style

Users can choose their favorite HTML page styles. The objects (including Table of Contents, Video/Audio Panel, and accompanying HTML objects) are arranged by framesets in the exported HTML file. Users can select whether the framesets are separated by rows or columns, and they can also decide the size of each frameset, as in Figure 8. The layout set in this figure is to separate the two framesets of the web page by rows, and each frameset takes half height of the entire page.
6. Set the object layout

Besides page style, object layout can also be set by authors as in Figure 9. Users can swap the positions of several objects by exchanging their position IDs in the Objects Layout window, and can also set the size of each object. If authors want to hide a specific object (including Table of Contents), they only need to set its height or width to zero. The hidden objects can be recovered by resetting their sizes.
7. Save and Load a .ws file

WebSmart defines a specific file format (.ws file) to save the intermediate results of the user’s input. Authors can save the current document and resume the editing in the future. The final web-accessible HTML files are created by exporting the .ws file from the wsEditor’s main menu (Figure 2).

8. CaptionMaker

The CaptionMaker is a visual tool (Figure 10) that makes closed captioning an easy task. It synchronizes caption texts with video/audio, and stores the encoded texts into .cap files. The .cap files are attached to their video/audio files in WebSmart Editor for adding closed captions into the multimedia presentations.
itself is built upon JMF technology. We are expecting to give you a real feeling of what JMF can do for you. If you have any comments or suggestions on improving the designing or anything related with the course, feel free to let us know. This course is based on JavaSoft's JMF specification and Intel's JMF tutorial. The course includes two parts: an instruction part, and a short quiz. The instruction part is composed of three chapters: JMF introduction, the Basic Media Player, and the extending part of the Basic Media Player. If you want to start from a specific section, click it in the syllabus frame. Now let's start and we hope you enjoy the course.

FIGURE 10. CaptionMaker window

Where:

Caption_field, the single line text field under the picture frame, is for displaying and editing a caption line for the current clip segment.

File_view, the text area besides the picture frame, displays the entire caption file with the current caption line highlighted.

Audio/video clips are played back in either step or continuous mode. In step mode, the playback pauses at the pace set by the user, waiting for him to enter the text corresponding to this short segment of the media content, before continuing to the next segment of the media. The continuous manner, on the other hand, allows the user to review the caption texts entered and make modifications.

9. QuizDesigner
The QuizDesigner maintains a list of questions (Figure 11), and each question is edited in a Question window (Figure 12). In the “Question” window, users enter the name of the HTML file which saves this question, the question body, the question type (an input question or a multiple-choice question), expected answer, and the time point at which a video clip should be paused and the name of the video clip to be paused. In the case of multiple-choice questions, users also have to enter the content of choices. The quiz and its questions can be saved in a .quiz file, the HTML files of the questions are generated by exporting the .quiz file.

FIGURE 11. QuizDesigner window
2.3 WebSmart synchronization statements

WebSmart adapts to different levels of users by different levels of interface. Section 2.2 illustrated how to create a web presentation exclusively in the system's visual tools. However, authors with programming experience can customize their documents for
both page layout and synchronization setup. To make more complicated page layouts, authors could refer to HTML manuals. To customize synchronization setup, WebSmart provides an event API for maintaining event tables and keeping statistics on page viewers’ interactions.

Instead of defining a complicated scripting language as most commercial products do, WebSmart’s synchronization event API consists of only three basic statements, namely, clearAllEvents(), addEvent(), endEvent(), and five extended statements for the QuizDesigner, namely, quizReset(), one_more_quiz(), one_more_correct_answer(), getQuestionsTaken() and getCorrectAnswers(). These statements are embedded in JavaScript functions within HTML files.

The syntax of these statements is as follows:

- `clearAllEvents();`
- `addEvent(JS_function, URL_jump, VideoClip, EventTime);`
- `endEvent();`
- `quizReset();`
- `one_more_quiz();`
- `one_more_correct_answer();`
- `getQuestionsTaken();`
- `getCorrectAnswers();`

The semantics are as follows:
• `clearAllEvents()` informs the system to build new synchronization event tables for the current section of the presentation. The event table will be discussed briefly in the `WS` class in Section 2.4.2 (Class Diagram). For more information about the implementation details on synchronization event table, please refer to [20].

• `addEvent()` adds a synchronization event into the event table. The scenario of the event is described as "asking the browser to execute the JS function when the VideoClip is being displayed to the EventTime". Usually the `JS_function` is to ask the browser to flip the content of an HTML frame within a Web page, and `URL_flip` indicates the URL address of the new content of the frame.

• `endEvent()` informs the system to close the event adding process and trigger the initialization processes.

• `quizReset()` starts a new quiz which consists of a set of questions.

• `one_more_question()` informs system that one more question has been taken. This statement, as well as the `one_more_correct_answer()` described next, is used to report quiz information and create statistics for the quiz summary page.

• `one_more_correct_answer()` informs system that one more correct answer is given by the page viewer.

• `getQuestionsTaken()` returns number of questions taken. This statement, as well as the `getCorrectAnswers()` described next, is used in creating the quiz summary page.
- `getCorrectAnswers()` returns the number of correct answers the page viewer has given.

A typical HTML file generated by the WebSmart authoring environment is the `section1.html` in Appendix A. Authors can also create or modify this kind of HTML files in a plain text editor. Notice in the sample file, "changetitle()" and "changeNotes()" are two JavaScript functions, "title" and "Notes" are names taken from two WebSmart objects in the presentation page. Chapter 3 discusses how WebSmart integrates Java, JavaScript, Java Media Framework and HTML in detail.

### 2.4 Object model

As the functional model (Section 2.1) defines the functionality of the system, the object model describes the static data structure of objects, classes, and their relationships to one another. It is a refinement towards the implementation phase in the project lifestyle. In this section we’ll first distinguish the roles of entity objects and interface objects, then give the class diagram of the system, focusing on entity objects.

#### 2.4.1 Entity objects and interface objects

According to Jacobson [25], objects within a software system can be divided into three classes based on their functionality. These classes are `control objects`, `entity objects` and `interface objects`. Interface objects describe the bidirectional communication between the system and its users. Entity objects are used to model the information that the system will handle over a longer period of time and also the behavior that naturally belongs to this information. In some complex cases, there often remains behavior that is not naturally placed in either interface objects or entity objects. Such behavior is placed in con-
trol objects. This thesis adopts the following notations in describing the relationship between these objects (Figure 13).

![Entity Object](image1.png) ![Interface Object](image2.png)

**FIGURE 13. Notation of entity objects and interface objects**

In WebSmart, we use only *Entity Objects* and *Interface Objects*. For each information structure, we define an entity object (named *unit* where *unit* is the name of the information structure such as *Section* and *wsObj*, etc.) to maintain its information and an interface object (named *unit_GUI*) to handle the communication between the system and its users. The entity object and its corresponding interface object exchange information via their public methods.

Figure 14 depicts an entity object (*WSObj*) and its corresponding interface object (*WSObj_GUI*), as well as their relationship. The WSObj_GUI pops up a window for users’ access, and collects users’ input such as text and checkbox selection. The WSObj’s property information collected in the WSObj_GUI is then reported to the WSObj. The WSObj retrieves the property information, does operations on it, and communicates further with other entity objects in the system. WSObj also feeds back information to the WSObj_GUI. When a WSObj_GUI opens an existing WSObj, it gets information from that WSObj for object name, height, and width.

The purpose of encapsulating the user interaction separately from the entity information is to protect the entity data from unintentional modification from outside of
the system, and also to establish a clearer interface between different developers within a team.

-0=0
-0=0

FIGURE 14. An example of interface objects and entity objects

2.4.2 Class diagram

Since the only role interface objects play is to pass users' input to entity objects, and major functions of the system are realized in entity objects, we'll focus on entity objects in the object model. We adopt Rumbaugh's Object Model Notation [26] in our object modelling. Figure 15 depicts the basic notations we use.

An **object class** describes a group of objects with similar properties (attributes), common behavior (operations), common relationships to other objects, and common semantics. An **association** describes a group of links with common structure and common semantics, it often appears as verb in a problem statement such as "class A uses methods defined in class B". An **aggregation** is the "part-whole" or "a-part-of" relationship in which objects representing the components of something are associated with an object representing the entire assembly. The **generalization** and the **inheritance** are powerful abstractions for sharing similarities among classes while preserving their dif-
ferences. Generalization is the relationship between a class and one or more refined versions of it. The class being refined is called the superclass and each refined version is called a subclass. Each subclass is said to inherit the features of its superclass. The solid balls are multiplicity symbols (zero or more). The sign “1+” means “one or more” multiplicity.

**FIGURE 15. Object Model Notation**

As introduced in the functional model, WebSmart system is made up of a presentation engine and an authoring environment. The authoring environment in turn is
implemented in five blocks: *wsEditor*, *Synchronization mechanism, GUI and Layout*, *QuizDesigner* and *utilities*. The former four blocks correspond to modules in the functional model defined in Section 2.1. The *utilities* block provides services such as printing debug information to the other blocks. There are also other associations among blocks. Figure 16 depicts the object model.

- **Presentation Engine**

  The presentation engine is an independent block in the object model. It is realized by three classes: *Display, EventGroup* and *EventThread*.

  The *Display* is the applet embedded in HTML files. Its major functions lie in three categories:

  - **Interfacing with the HTML file.** The applet reads the parameters listed in the HTML file and supplies these values to the media player. These parameters include video file name, audio-only file name, and caption file name.

  - **Controlling the media player.** The media player which presents the media content is created by the Display applet. This player realizes all the controlling functions provided to the end user. These functions include *start, stop, suspend, resume, fastforward, and rewind*. Besides, an “audio-only” option (refer to Chapter 4 Performance) is also supported in the Display applet.
• **Interfacing with JavaScript.** The interaction between the web page and its viewers is realized by the communication between Java and JavaScript (details will be given in Chapter 3: Technology Framework). The JavaScript to Java communication is realized in the public functions provided in this Display applet.

The **EventGroup** is used by the **Display** to control the multiple events taking place during a presentation. As there are multiple frames within a web page, and each frame can define synchronization events associated with the media presentation, the EventGroup provides a consistent management over these events. Its major function is to pass the controlling operations to each thread member within the group.

An **EventThread** object takes care of synchronization events taking place within a single frame. It can be a member of the EventGroup. The major function of an EventThread is to wait for the time set by an event and execute the event by calling JavaScript functions.

• **wsEditor**

Block **wsEditor** assembles a presentation's major parts, including, **WS, WS_Panel, ContentPanel, Section, VideoPanel**, and **WSObj**.

A **WS** object represents a WebSmart presentation, and it is usually made up of multiple **WS_Panels**. The major role of the WS object is to keep track of all the panels and objects on the page, and export them to HTML files at last. The WS object also maintains a hashtable for synchronization events. This event table is indexed
by video clip names, and saves a vector of WSObj objects which have associated synchronization events for each video clip. At the time to write a section HTML file, a Section examines its video clips, extracts synchronization events for each WSObj associated with each video clip, and inserts events into the HTML file using the synchronization statements defined in Section 2.3.

A WS_Panel object is physically displayed as a layout frame, and functionally embeds multimedia content. According to the type of multimedia content it embeds, a WS_Panel can be realized as a ContentPanel, a VideoPanel, or a WSObj. WS_Panels are defined recursively. A WS_Panel can contain other WS_Panels, therefore a hierarchical structure is formed. The WS_Panels at the top level of the hierarchy are exported to the framesets in the final presentation page, and WS_Panels at the lower levels are exported to frames.

A ContentPanel corresponds to the Table of Contents under our distant learning context. It contains at least one Section. Its major function is to maintain all the sections for a presentation. Besides the addition and deletion of a section, the ContentPanel also maintains the hierarchy of the sections within a presentation. The numbering of a section (e.g. “1.” or “1.1” or “1.1.1”) and its indentation are created automatically by the system.

A Section simulates a part of a speech under our distant learning context. It acts as a means to bring audio/video content into a web page. The set of sections within a ContentPanel makes up the complete scenario of a presentation. According to the interactivity involved, sections are divided into two types: "instruction" and "quiz". Interactions in an instruction section are only between the Media compo-
nent of the web page and the page viewers. Interactions in a quiz section also extend to those among the accompanying frames of the web page and the page viewers.

The VideoPanel corresponds to the Media component. It plays back audio/video content in an exported WebSmart presentation. In the authoring environment, the VideoPanel uses the PlayerFrame to playback the video clips for users to capture timing points of synchronization events.

A WSObj instance is a WS_Panel object which is neither a ContentPanel nor a VideoPanel. It corresponds to an accompanying frame within the exported web page. Content of an accompanying frame can be flipped at pre-defined media points as the audio/video content is being presented in the Media component. This kind of flipping activity is called an event. The WSObj class provides functions in three categories:

- communication with the WSObj_GUI object on object’s property information.
- communication with the WS object on the synchronization event hashtable.
- communication with the PlayerFrame object to create synchronization events.

• Synchronization mechanism

The synchronization mechanism block is composed of VideoPanel, WSObj, PlayerFrame and its subclass CaptionMaker, and CaptionPace. Two levels of synchronization defined in Section 2.1 (Functional Model), namely, flipping page and lip synchronization, are set up in this block.
A *flipping page* event is generated by the communication among the VideoPanel, the WSObj and the PlayerFrame. The VideoPanel responds to users’ requests to play video clips by opening the PlayerFrame. The main functions of the PlayerFrame are to play the video clips sequentially and to report media time at WSObj’s request. The WSObj responds to users’ requests to insert an event, gets the current media time from the PlayerFrame, then updates the event timing information of the WSObj. Later this event timing information will be used in the event hashtable of the WS object.

The *lip synchronization* information is created by the CaptionMaker and the CaptionPace. The CaptionMaker inherits the continuous media playback operation from the PlayerFrame, and extends another "STEP" playback operation. In the STEP mode, the media playback pauses automatically after a certain length of media stream has been played, waiting for the user to enter caption text. After the text for the whole video clip have been entered, the user can switch back to CONTINUOUS mode to review the generated caption file. The CaptionPace is used to set the length of time for each periodic play in the STEP mode.

**• GUI and Layout**

The GUI and Layout block is composed of GUI classes and Ratio_Layout. The GUI classes implement interface objects in a way described in Section 2.4.1 (Entity object and Interface object). Their basic function is to take users’ input and pass the information to the corresponding entity objects.
The **Ratio_Layout** is an extended layout manager developed by Parr [28]. A layout manager is an object that positions and resizes the components in an AWT container according to some algorithm. Traditional layout managers defined in the Java language (e.g. BorderLayout, FlowLayout, CardLayout, GridLayout, and GridBagLayout) divide the space of a webpage into grids in different levels of granularity. Each component on the page occupies certain blocks of grids. Although the granularity is becoming finer and finer, the page author still lacks the flexibility of setting a component at an arbitrary position and at an arbitrary size. The Ratio_Layout, however, meets this demands effectively. Every component is specified by a ratio vector \((x, y; \text{width}, \text{height})\), where \((x, y)\) is the position of the upper left corner of the component, and \((\text{width}, \text{height})\) decides the size of the component. Every parameter within the vector is a percentage ratio to the entire page size. For example, a component specified by \((0, 0; 0.45, 0.75)\) is a rectangle starting at the upper left corner of the page, and taking 45% of the whole page’s width and 75% of the whole page’s height.

The behavior of a layout manager is effected by the implementation of a **LayoutManager** interface behind the scene. By implementing the following methods defined in the LayoutManager, users can define their specialized layout manager.

- **addLayoutComponent(String r, Component c)** defines how a component is added into the associated **Container**.
- **removeLayoutComponent(Component comp)** defines how a component is removed from the **Container**.
• preferredLayoutSize(Container target) defines the preferred size of this container after laying out all the objects.

• minimumLayoutSize(Container target) defines the minimum size of this container after laying out all the objects.

• layoutContainer(Container target) sets the (x,y) coordinate and size (width, height) of each component contained in the target Container.

The Ratio.Layout implements the above methods of the LayoutManager interface with a special modifier to indicate the characteristics of each component in the container. The modifier is in the form of "xratio, yratio;widthRatio,heightRatio". By keeping track of modifiers for all the components, and calling reshape() to draw these components on the screen, the Ratio.Layout sets the components at arbitrary positions and arbitrary sizes within a page.

• QuizDesigner

The QuizDesigner block consists of Quiz and Question. It creates a special type of section - quiz. A Quiz object has multiple Question parts. The main functions of the Quiz class are maintaining the question list, updating a .quiz file, and generating a quiz summary page upon users' request. Each question is exported to an HTML file which realizes the interaction scenario described in Section 2.1 (Functional Model).

• Utilities
The *Utilities* block provides services to other blocks of the system. The **WS_Util** uses **InfoDialog** to create a message window which displays multiple lines of information (via the **MultiLineLabel** class). The **WS_Util** also generates dialog window to ask users for confirmation (via the **YesNoDialog** class), and generates dialog window to allow users to open or save a file (via the **FileViewer** class).
FIGURE 16. Object Model
Chapter 3

Technology Framework

Java, Java Media Framework (JMF), JavaScript and HTML constitute the technology framework of the WebSmart system. This chapter discusses the advantages and disadvantages of each component, and gives an example on how WebSmart is built by integrating these components.

3.1 Java

Designed by Sun as “A Simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language”, Java [14] is used as the system programming language of WebSmart. This section examines some of Java’s major features and discusses both Java’s advantages and disadvantages as it is compared with other languages.

- Object-Oriented

To function within increasingly complex, network-based environments such as the Internet, programming systems must adopt object-oriented concepts. The reason is that the encapsulated, message-passing paradigm of the object-oriented methodology coincides with the needs of distributed, client-server based system. Java was designed to be object-oriented, and it [15] supports primary object oriented characteristics such as Encapsulation, Polymorphism, Inheritance and Dynamic binding.
The mapping between the object model of WebSmart and its implementation is straightforward: all the classes, associations, inheritance and aggregation have direct correspondences in the language. One disadvantage of Object-Oriented languages is the overhead caused by intra-object communications, and this makes them not as suitable as process-oriented languages (such as C) for real-time applications.

- **Architecture neutral and portable**

  In contrast to C++, Java doesn't generate a binary format machine code which depends completely on the native hardware instructions - rather, it generates byte-codes: a high-level, machine-independent code for a hypothetical machine. The bytecodes generated by a Java compiler is realized in specific hardware and software platforms by the Java interpreter and run-time system. This kind of architecture-neutral and portable language platform of Java is known as the **Java Virtual Machine**. The benefit of this virtual machine mechanism is obvious. In a heterogeneous network environment as the Internet, applications must be capable of executing on a variety of hardware and software architectures. The disadvantage of Java's interpreted nature is that the interpretation process of the bytecode slows down the execution speed.

- **Multi-threaded**

  Multimedia applications are featured by many concurrent activities such as playing an audio/video clip in one frame of the web page, displaying images in another frame, and at the same time waiting for users' input in yet another frame. Such an interactive scenario requires a multithreading support from the platform on which it
runs. Java, fortunately, has support for threads built in its language level. A Thread class is defined, with a list of methods including run(), start(), stop(), sleep(), setName(), getName(), isAlive(), etc. Users now can assign a specific task to a thread, and make multiple threads run asynchronously within an application.

Moreover, Java defines another class called ThreadGroup, which, as the name implies, is a class that handles a group of threads. By organizing a set of relevant threads into a group, programmers are allowed to manipulate many threads by calling a single method such as suspend(), resume(), stop(), or destroy(), etc.

Conventional languages such as C and C++ are single-threaded. Implemented in Java, WebSmart has deployed Threads and GroupThreads efficiently in its synchronization tasks.

• Extensible

Ever since its debut in late 1995, Java has expanded in various areas in computer science. Besides its core language features, Java has already extended APIs to such areas as database access (JDBC), remote execution (RMI), interactive media (JMF), sound (JavaSound), graphics and imaging (Java 2D and Java 3D), and even telephony (Java Telephony). One of the considerations in the early design phase of WebSmart was to take advantage of the rich set of APIs that Java vendors provide, and to build object oriented modules which can be easily expanded as the Java platform expands. The Java Media Framework discussed below is one of the recently published Java APIs on which WebSmart is built. Currently only JMF Player is
defined and implemented. Once the live capturing and video conferencing are real-
ized in JMF, these functions will be merged into WebSmart without much difficulty in design and implementation.

3.2 Java Media Framework (JMF)

As "an application programming interface (API) for incorporating media data types into Java applications and applets", JMF [16] is utilized by the WebSmart project in several critical modules such as the presentation engine, and the synchronization mechan­anism.

JMF is developed jointly by Sun Microsystems Inc., Silicon Graphics Inc., and Intel Corporation. With its first official version being released in September, 1997, it now has following implementations on different platforms:

<table>
<thead>
<tr>
<th>vendor</th>
<th>platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun</td>
<td>Solaris</td>
</tr>
<tr>
<td></td>
<td>Windows 95/NT</td>
</tr>
<tr>
<td>SGI</td>
<td>Irix 6.2/6.3</td>
</tr>
<tr>
<td></td>
<td>(JMF spec ver 0.96 only)</td>
</tr>
<tr>
<td>Intel</td>
<td>Windows 95/NT</td>
</tr>
<tr>
<td>Brian Griffith</td>
<td>Macintosh</td>
</tr>
<tr>
<td>Archipelago Productions</td>
<td>Windows95/NT</td>
</tr>
</tbody>
</table>

**TABLE 1. JMF implementations**

JMF is being released in three stages: Player, Capture, and Conference APIs. At the time of this writing, only Java Media Player has been released. It supports the synchronization, control, processing and presentation of compressed streaming and stored time-based media, including video, audio and MIDI across all Java enabled platform.
The central structure within the Java Media Player is the player. A player is defined in [16] as "an object that processes a stream of data as time passes, reading data from a DataSource and rendering it at a precise time". The operations on a player during its lifecycle can be described as follows:

1. Creating an instance of the player

A player can be created directly from a URL by calling `createPlayer()`. The Manager uses this media URL to create an appropriate type of player according to the content of the media resource. An identical interface exists to embed media for both a web-oriented document and a local file stored in CD-ROM. When created, a player also needs to register a `ControllerListener` so that the latter can observe media events posted by the player.

2. Controlling the player

Before any media content can be presented by a player, a `realize()` operation has to be done to allocate system resources (e.g. visual component and control component) exclusively to the player. As the resource of presentation is being allocated, an asynchronous operation `prefetch()` can be applied on a player to get its media content from either the network connection or local storage. Once the visual component and control component are ready, a player can start displaying media content as long as there is content prefetched into the buffer. Document viewers interact with the player from the control component (a set of buttons on the web page) to `start()` or `stop()` the presentation.

3. Responding to media events
As the player changes its status among unrealized, realizing, realized, prefetching, prefetched, started or error situations such as resource unavailable, connection error, etc., it posts event reporting messages to the ControllerListener registered, therefore the latter can react correspondingly.

4. Deallocating the player

When the player is no longer in use, it should be processed in the following three steps:

- first, stop() the player, so that it's not active.

- second, deallocate() the player to release any exclusively resources and minimize its use of non-exclusive resources so that other players can start as soon as possible.

- finally, close() the player to release all of the resources that the player was using and causes the player to cease all activity.

WebSmart adopts JMF as its major component in the multimedia presentation framework for the following reasons:

- Open architecture.

Differentiating from other proprietary players wrapped as plug-ins, Java Media Player provides the application developers with an unified interface for embedding audio and video in their applications and applets. Users do not need to prepare their media files into a special format, because the current Java Media Player supports most of the widely used audio and video formats/codecs and transport protocols.
including DVI, MIDI, MPEG-1, H.261, H.263, AVI, QuickTime, File, FTP, HTTP, RTP, etc. Besides, a piece of code written by the JMF specification runs smoothly in all JMF implementation platforms without modification.

- Streaming media support

Streaming is a buffering technique which begins playback of a video clip before the whole video file has been downloaded. By giving the video file a few seconds to load before starting the image, a reserve of video is available in the memory of the client's computer. The display of the buffered content and transmission of the next sections taking place concurrently, solved the delay problem in media presentation, and provides a real-time video access via the Internet. Streaming related standards include the Internet Engineering Task Force (IETF)'s Real Time Streaming Protocol (RTSP) [4] and Microsoft's Advanced Streaming Format (ASF) [5], etc.

The JMF specified the support to streaming video. When using an HTTP connection to download media files in certain 'streamable’’ formats (MPEG and Quicktime), the Player can begin playback before all of the media data is received. If the download rate is not enough to keep up with the rate at which the Player is using data, an under-run condition occurs. At this time the Player continues to download data, waiting until enough data has been received to begin playback again (Prefetching).

- Time-keeping and Synchronization support
As stated in Chapter 1 (Introduction), a time management mechanism is required to deal with time-based media such as audio and video in multimedia documentation. The JMF TimeBase and Clock interfaces define the mechanism for managing the timing and synchronization of media playback.

A TimeBase represents the flow of time. "A Java Media player uses its TimeBase to keep time in the same way that a quartz watch uses a crystal that vibrates at a known frequency to keep time" [16]. A Java Media Player can use its TimeBase to keep time but never transform or reset it. On the other hand, a "media time" is defined for a player to represent a point in time within the stream that the player is presenting. "The media time can be started, stopped, and reset much like a stopwatch" [16]. The media time of a stream can be mapped to a system TimeBase by a Clock object.

WebSmart controls the time-keeping within a single media stream by adjusting its MediaTime, and synchronizes among multiple media streams by associating their players with the same TimeBase.

3.3 HTML

The HyperText Markup Language (HTML) [10] is the lingua franca for publishing hypertext on the World Wide Web. As illustrated in Section 2.1 (Functional Model), the major output of the WebSmart authoring environment is a group of HTML files which are fed into the web browser. This group of HTML files form a logically hierarchical structure in presenting the content of a WebSmart presentation:
FIGURE 17. The output HTML files of WebSmart

In contrast to the above logical hierarchy, the actual HTML files are stored in the server in a flat structure (e.g. they are files under a same directory), with indistinctive relationship between each other.

The *application.html* file uses tags `FRAMESET` and `FRAME` to organize the layout of multi-components within a page, and *sectionX.html* files uses tag `PARAM` to define multimedia contents such as `VIDEO-FILE`, `AUDIO-FILE`, `IMAGE-FILE`, and `CAPTION-FILE`. Display of the page layout is static. Parsing and operating on the media contents are made by the presentation engine implemented in a Java applet.

Despite its wide applications on the Web, the original HTML has limitations in creating dynamic web pages. Section 6.2 (Future work) will discuss some of the new languages and interfaces being developed to produce dynamic web pages.

### 3.4 JavaScript

JavaScript is used by WebSmart as a scripting language to link various components (e.g. Java, Java Media Framework, and HTML) within the system. As Ousterhout ana-
lyzed in [35], system programming languages (C, C++, Java) are suited for building components where the complexity is in the data structures and algorithm, and scripting languages (Perl, Tcl, Visual Basic, JavaScript) are well suited for combining applications where the complexity is in the connections. Compared with system programming languages which are mostly strong typed and compiled, scripting languages are generally typeless and interpreted. A typeless language makes it much easier to link together components. Interpretation provides rapid development by eliminating the compile times. Moreover, being a much higher level of language than system programming languages allows more casual programmers (instead of professional programmers) to master a language in a short period of time and focus on their main jobs in building applications. So far scripting languages are widely applied in three categories of applications: Graphical User Interface (Visual Basic, HyperCard, and Tcl/Tk), Internet (Perl, JavaScript), and Component frameworks (Visual Basic).

JavaScript [17] is "Netscape's cross-platform, object-based scripting language for client and server applications". The language can be divided into a client side and a server side. Both sides share the Core JavaScript which consists features such as variables, functions, and LiveConnect. The client-side encompasses extras such as the pre-defined objects only relevant to running JavaScript in a browser, and server-side encompasses extras such as predefined objects and functions only relevant to running JavaScript on a server. So far, WebSmart only utilizes the client-side JavaScript, of which the LiveConnect is the most attractive feature to the project.

LiveConnect is used to enable communications between JavaScript and Java applets in a page and between JavaScript and plug-ins loaded on a page. WebSmart uses
it in realizing the mutual communication between a Java applet and JavaScript functions embedded in an HTML file. The control flows among different parts in the framework as follows:

![Control Flow Diagram](image)

**FIGURE 18. Control flow in WebSmart**

Users control the process of a presentation by the interactions with an HTML page such as pressing a button or entering a textfield. On receiving these actions, HTML calls on the JavaScript functions in the page and passes users' input to these functions. JavaScript functions in turn call the methods defined in the Java applet which is embedded in the same HTML file. Java applet functions execute tasks such as starting or stopping the presentation of a video stream. The control flow can also be reversed in such cases as flipping a frame within the HTML page. In this case, a method in the Java applet calls a function in the JavaScript scripts. Since the JavaScript has the access to components of a page such as windows and frames, it flips a frame by resetting the content of that frame. At the end, users notice the changes happened in the web page in a real time manner.

The communication between Java and JavaScript is realized as follows:

- **JavaScript to Java**
When the LiveConnect is enabled, JavaScript can make direct calls to Java methods. A Java method is referred to in the following format:

\[\text{[Packages.]}\text{packageName.className.methodName}\]

For example, with the following tag in the HTML file:

\[<\text{APPLET Name="clock" Code=Display.class Width=320 Height=340 MAYSCRIPT}>\]

where \text{Display.class} is the name of the compiled bytecode for the Java applet, the JavaScript function can call a method of the applet in

\[\text{applets["clock"].appletPlay()}\]

to start the media player.

- **Java to JavaScript**

To access JavaScript methods, properties, and data structures from within a Java applet, users should import the Netscape javascript package to the Java source code. The author of an HTML page must permit an applet to access JavaScript by specifying the MAYSCRIPT attribute of the \texttt{<APPLET>} tag. Access to a JavaScript function from a Java applet proceeds in two steps:

1. Get a handler for the JavaScript Window

   Before you can access JavaScript, you must get a handle for the Navigator window. The statement below shows how to get a window handle within a Java applet:

   \[\text{JSObject win = IOObject.getWindow(this);}\]

   where \text{JSObject} is a class name in the \texttt{netscape.javascript} package.
2. Calling JavaScript methods

Use the `call()` method of `JSObject`, which has the following syntax:

```javascript
JSObject.call(methodName, argArray())
```

where `argArray` is an `Array` of Java objects used to pass arguments.

For example, when a pre-defined timing point for a synchronization event is due, the Java applet calls JavaScript function `js_func` in

```javascript
win.call(js_func, arg);
```

where `win` is the window handle obtained in step 1, and `js_func` could be a flipping action such as defined in an HTML page as follows:

```javascript
function changeNotes(n) {
    top.frames["Notes"].location=n;
}
```

where `Notes` is the name of a frame in the page, and `n` is the URL address for the new content of the frame.

3.5 An example of integration

Now we've discussed the major components of the WebSmart technology framework, we'll see how these components create a WebSmart exported HTML file in detail. We use the following scenario in demonstrating the integration of Java, Java Media Framework, JavaScript and HTML in this section:

As the user clicks on the Quiz section in the TOC frame, a video clip which narrates questions will start presenting in the media frame. At the 2nd second of the narration, the title frame will flip to a new title page, showing the Quiz logo. At the 11th second of the narration
the Notes frame will display the content of question 1. At the 13rd second of the narration (it is the time point when the narration of the first question is over), the presentation of the video will be paused until the user enters the answer into the text field in the Notes frame and clicks the "Done" button. Upon receiving the "Done" button, the presentation of the video resumes and starts the narration of the next question.

The above scenario is accomplished by two HTML files (e.g. section3.html [Appendix B] and q1.html [Appendix C]), and three Java programs (e.g. Display.java, EventThread.java and EventGroup.java) in following steps.

1. Enabling LiveConnect

- Import JavaScript package into Java applet.

  The Display.java, EventThread.java and EventGroup.java should include the following statement at the beginning of the source code:

  ```java
  import netscape.javascript.*;
  ```

- Specify the MAYSCRIPT attribute in the <APPLET> tag to permit the Java applet to access JavaScript functions embedded in section3.html.

  ```html
  <applet name = "clock" code=Display.class width=320 height=350 MAYSCRIPT>
  ```

- Get the handle of the Navigator window in the Java applet. This window handle will be used when the applet calls a JavaScript function. The following function in the Display.java gets the window handle when the applet is invoked.
public void start() {
    
    window = JSObject.getWindow(this);

    ...
    
}

2. Embedding media streams in HTML files

The following declarations have to be included in section3.html for the Java applet Display.class to get the media streams of the presentation.

<PARAM name="VIDEO-FILE" value="movies/sec11_1.mpg">
<PARAM name="AUDIO-FILE" value="movies/sec11_1.mpa">
<PARAM name="IMAGE-FILE" value="movies/sec11_1.gif">
<PARAM name="CAPTION-FILE" value="movies/sec11_1.cap">

3. Embedding JavaScript functions in HTML files

JavaScript functions accomplish the following objectives:

- Adding synchronization events

The loadEvents() functions in section3.html and ql.html call the addEvent() function of the Java applet to add the events of flipping the title page in the title frame, flipping the question content into the Notes frame, and pausing the video presentation in the media frame. Following is the loadEvents() function in ql.html.

function loadEvents() {
    
    var APP=parent.frames["media_app"].document.applets["clock"]; 

    APP.addEvent("pauseApp", "," , "movies/sec11_1.mpg" , "13"); 
    
}
• Flipping actions

The following two functions implement the flipping of the title frame and the Notes frame within the browser window. JavaScript can refer to the whole window, or framesets and frames within the window, and update their properties. In the sample functions, the parent refers to the frameset window that contains the frame from which the JavaScript function is called, the location property is used in JavaScript to denote the contents of a frame, and the parameter n is the URL address to the new content of the frame.

```javascript
function changeTitle()
{
    parent.frames["title"].location = n;
}
changeNotes()
{
    parent.frames["Notes"].location = n;
}
```

• Controlling Java applet

JavaScript controls the process of the Java applet by calling Java applet's method (e.g. the appletPause() in the example). In the next step (Writing Java public methods) we'll see how this appletPause() is implemented.

```javascript
function pauseApp()
{
    var APP = document.applets["clock"]; 

    APP.appletPause();
}
```

4. Writing Java public methods
To be accessed by JavaScript functions, a Java method has to be defined as public. According to the functionality, the public Java functions for communication with JavaScript in WebSmart can be divided into three categories:

- **Handling event tables**

  As introduced briefly with the WS class in Section 2.4 (Object Model), an event hashtable is maintained for synchronization events. In the Display Java applet, functions are defined to associate these events with the corresponding threads. Details on this event hashtable can be found in [20].

  ```java
  public void addEvent(String jsfn, String layer, String mfile, 
  String t);

  public void endEvent();
  ```

- **Controlling the Java Media Player**

  This is where the JMF is used in WebSmart. The Display Java applet defines a Java Media Player which realizes the playback of the media streams in the media frame. By applying the `start()` and `stop()` operations on the Player, the applet controls the playback of the media stream.

  ```java
  public void appletPause() {
    
    player.stop();
  }

  public void appletContinue() {
    
    player.stop();
  }
  ```
• Calling JavaScript methods

When the time for a synchronization event (such as flipping content of a frame) is due, the Java applet will call corresponding JavaScript functions (as defined above in Embedding JavaScript functions in HTML files) to execute the synchronization event. The following function `executeEvent()` is defined in the EventThread.java.

```java
public void executeEvent(String layer_id) {
    String arg[] = {layer_id};
    win.call(thread_function, arg);
}
```

where `win` is the handle of the Navigator window which was obtained in Enabling LiveConnect step.

5. User Interface

At last we come to the GUI end where users interact with the system via the HTML page in the browser. In `ql.html` we use a "text" type of input for users to enter texts, and a "button" type of input for confirmation.

```html
<P><FORM NAME="quiz"><INPUT TYPE="text" NAME="answer" VALUE="" SIZE=50 ></FORM>
</P>
<P><INPUT TYPE="button" VALUE="Done" NAME="submit" OnClick=
check_answer()></P>
```
Chapter 4

Performance

The WebSmart is a multi-platform application which works on both UNIX and Windows platforms. To test the performance of accessing multimedia documents from the Internet, an HTTP server and a distant learning application have been built. This chapter analyzes the experimental results.

4.1 Multi-platform implementation

The WebSmart system [36] runs on multiple platforms. Currently the authoring environment works on Irix 6.2, Solaris, and Windows95/NT, with some small modifications in source code for the Irix 6.2 because the SGI's JMF implementation only supports the 0.96 version of the JMF specification. The presentation engine (the Display Applet) can be embedded in both Netscape’s Navigator (version 3.X or more) and Microsoft’s Internet Explorer (Version 3.X and more), and it provides applets compatible with both version 1.0 and version 0.96 of the specification. To run the system, a Java Development Kit (version 1.0 and up) and a Java Media Framework implementation have to be installed on local machines.

4.2 HTTP server

To test the performance of the WebSmart broadcasting on the Internet, a Netscape En-
terprise Server version 3.5.1 (later Apache HTTP server version 1.2.5) is installed on a
SGI workstation (rainforest.cs.ubc.ca) which runs Irix 6.2 at the Computer Science De­
partment of UBC. The URL address of the server is http://rainfor­
est.cs.ubc.ca/.

4.3 An application - a web-based JMF course

Besides the HTTP server, an application was set up on it to test the performance of ac­
accessing WebSmart documents via the Internet. The application is at the URL address
of http://rainforest.cs.ubc.ca/mmas/app1_10/appl.html for JMF
1.0 specification compatible, or http://rainforest.cs.ubc.ca/mmas/
app1_96/appl.html for JMF 0.96 specification compatible.

The application is a web-based course on Java Media Framework, and includes
two instruction sections as well as a quiz section. The whole web page of the course is
composed of four frames, namely, a Table of Content, a media playback frame, a title
frame and a Notes frame. The narration of the course is divided into short video clips,
each lasts for an average of 50 seconds in time. After compressing into MPEG-1 for­
mats, these clips are on average of 8.7MB.

The application is accessible via both LAN and 28.8Kbps modem. When ac­
cessed from a PC connected with the HTTP server via a LAN, the video presentation is
performed in a “stop-and-go” manner. When the bandwidth becomes insufficient tem­
porarily, a “data starved” message is displayed, and the download bar above the video
component within the media frame displays the buffering process. As soon as enough
media content has been fetched in the buffer, video presentation will resume by clicking
the play button.

When viewing the same JMF web course from the 28.8Kbps modem, the performance is unacceptable. For example, it takes about five minutes to prefetch enough content in buffer before video presentation starts.

4.4 An Audio-only option

To meliorate the unacceptable performance of the original presentation engine, an option was added to allow the document viewers to switch to an audio-only presentation mode when the bandwidth of the network connection becomes scarce. The option is provided with a button laid above the visual component of the media player. When viewers open the web document, a video clip is displayed by default, and the button shows the "Audio Only" text. Whenever viewers click the button, the video presentation will be stopped and an audio only clip which narrates the same presentation starts. At the same time, the button modifies its text display to be "Audio-Video" for viewers to switch back to video presentation mode.

The audio only clips divide the whole course narration into same lengths' segments as their corresponding video clips do. For the same length of 50 seconds narration, the audio clips take an average of 605KB, which is about 6.9% the volume of the video clips. The reduce in the media size transferred improved the performance of the presentation, but at the same time, lost the distinctive visual effect brought by the video clips.
4.5 Other issues

4.5.1 Compression algorithms

The performance enhancement from the audio-only option was obtained by sacrificing the distinctive visual effect of a video playback, therefore it can not be used as a long term approach. Besides, even in the audio-only mode, the 28.8Kbps modem access still jams frequently. The reason is that the MPEG-1 which is used in compression is not an ideal solution to Internet-based multimedia applications. For a compression algorithm to be suitable for the current Internet, an algorithm has to possess the following features: bandwidth scalability, resolution, frame-rate, frame-quality scalability, fast compress/decompression, ability to cope with network losses, and small encoding and decoding latency. MPEG-1 was designed for playback from CD-ROM, targeted at a 1-1.5Mbps bandwidth which is impractical for modem access to Internet.

Several new algorithms specifically targeted at Internet video are being developed. First is the RealSystem codec which compresses video content adapted to scalable bandwidth ranging from 10Kbps to 500Kbps. Recently the RealNetworks is working with Sun to integrate RealSystem video streams into JMF implementations. Second is the H.323/H.324 standards defined by the ITU. The core idea used here is that the video codec is a modular entity that can be plugged in and out. It is assumed that certain base video codecs (e.g. H.261/H.263, which are for the low bit-rate applications) are present in all implementations. By using the H.245 protocol, an H.323 application can negotiate the use of any other audio/video codec. This allows for new innovations to be easily incorporated into existing applications and adapts to applications which impose scal-
able bandwidth requirements. The last new standard introduced is the MPEG-4 [37] which is being developed by the MPEG group. Compared with its predecessors (i.e., MPEG-1 and MPEG-2), MPEG4 is designed for applications including Internet Multimedia, interactive Video Games, VideoConferencing, VideoPhone, Multimedia Mailing, and Wireless Multimedia, etc. MPEG-4 video is optimized for bandwidths ranging from low (<64Kbps), intermediate (64-384Kbps) to high bitrates (384Kbps - 4Mbps). Furthermore, MPEG-4 supports scene description (spatio-temporal synchronization and behavior) for audiovisual object presentation purpose, therefore is suitable to WebSmart presentations.

As these new Internet-oriented compression algorithms will provide new solutions to deal with scalable network bandwidth, the task of multimedia authoring systems is to build an flexible framework so that these new codec innovations can be accommodated easily once they become available. And this, was exactly one of the design strategies of the WebSmart system.

4.5.2 General purpose web server vs. multimedia specialized web server

Currently the WebSmart uses a general purpose web server to store multimedia data and transmit data to the client site. Such a server-less video streaming approach has advantages from an economic perspective, however, it also inherits many disadvantages from its simplicity. Once the client starts displaying the video content after a few seconds of buffering the file, the smooth presentation on the rest of the file is based on the hypothesis that the rest of the file must come over the network at a rate greater than or equal to the rate at which it is being played by the client (i.e., the bit-rate at which it was encoded). If ever this hypothesis is not realized, the video shown will become "stop-
and-go” in which the client waits till the buffer builds up again. This is obviously annoying and undesirable.

A better solution is to set up a multimedia specialized web server which is separate from the traditional HTTP-based web server and specialized in the video/multimedia streaming task. Its major difference is made by a function called “admission control”. Once a video is requested from a client, the server first decides whether to accept or deny the request based on the server’s CPU usage, network load, the identity of the client, or any other factors. If the request is accepted, a connection is established and the video is streamed to the client. The data packets are properly spaced in time on the network to match the fraction of link bandwidth being used by the client. The benefits of a server-based streaming video include better network throughput, better video quality to the end user, support for advanced features like admission control, cost-effective scalability to a large number of users, and protection of content copyright. As introduced in the Chapter 5 (Related Work), both RealSystem and NetShow use a specialized streaming video server.

4.5.3 Network connection

As we analyzed the performance over LAN, we also noticed some non-deterministic behavior. In low-load situations, the video presentation is smooth, but as soon as more traffic is injected into the subnet, the video display becomes jerky. The reason is rooted in the Carrier Sense Multiple Access with Collision Detection (CSMA-CD) which has been applied in Ethernet-based LANs. In high-load situations it exercises no control over access delay or available bandwidth per application. In addition, it does not provide any access-priority mechanisms and thus cannot give preferred treatment to real-
time traffic over conventional data.

Asynchronous Transfer Mode (ATM) seems to be a good solution to the above problem. The ATM is a cell-based multiplexing and switching technique. In the physical layer, it is targeted at 155Mbps optical transmission. In the ATM layer, the 53 octets ATM cells are multiplexed and switched in the network. In the ATM adaption layer, different adaptation layers (from AAL 1 to AAL 5) provide different classes of service to the upper application level. These services include constant bit rate and variable bit rate. Unlike the Ethernet, the ATM connection specifies quality of service (QoS) at the connection establish service, therefore avoids the non-deterministic behavior occurred in Ethernet. Although ATM seems to be a promising network in delivering real-time multimedia, the high cost to establish such kind of network prohibited its wide application. How the ordinary web users can benefit from it remains unseen.

Besides ATM, another kind of high-speed network is also in development. It is called high-speed LAN, and includes 100BASE-T Fast Ethernet and 1Gbps Gigabit Ethernet [38]. These high-speed networks inherit existing frame format from lower-speed Ethernet, therefore are easier to upgrade from current network devices with less cost in training, maintenance and troubleshooting than ATM is. Once these low-cost high-speed networks are combined with some QoS standards, for example, Resource Reservation Protocol (RSVP) [39], they can provide a practical and reliable solution to Internet multimedia presentations.
Chapter 5

Related Work

Multimedia Authoring System is a new area in computer science, yet it has grown rapidly within a year. Several commercial products have been released, and an international standard is in progress.

5.1 Authorware

The Authorware Interactive Studio [21] is a series of products developed by Macromedia for making multimedia presentations.

Authorware (currently version 4) provides an icon-based, drag-and-drop interface to allow users to create pages that include hyperlinks to text, digital movies, graphics and sounds. It also provides built-in editors for creating custom interactions such as graphic buttons, check boxes, and radio boxes.

The video embedded in an Authorware page is transmitted and played back by a plug-in called Shockwave which supports streaming video. The content of a Shockwave movie could be prepared by Director (currently version 6), another component of the Authorware Interactive Studio series. Director creates a format of media called Direct movies which can import 2D and 3D graphics, text, animation, sounds, and digital video. To add interactions into Director movies, page authors have to write scripts in the Lingo language. The Shockwave movies could be embedded in HTML codes,
and downloaded and played back in Internet browsers.

The major strength of Authorware is its rich functionality in generating and manipulating graphics and animation. It can be used by WebSmart users in creating HTML files for accompanying objects of the audio/video clips. One problem of the Authorware is that when creating Shockwave movie files, authors have no control over the bandwidth of the exported movies files. When the movies are transported in an unstable network such as the Internet, the performance of the video playback is unpredictable. Another problem is the compatibility of an HTML page which contains Director or Shockwave movies. A tag has been used for inserting a movies into an HTML file. However, Netscape’s Navigator and Microsoft’s Internet Explorer use different tags, and this brings extra work to page authors.

5.2 RealSystem

The RealSystem [22] is RealNetworks’s streaming media solution to the Internet and Corporate Intranets. It includes RealPlayer, RealServer, RealMedia Tools, and RealEncoder/RealPublisher, etc. Among them, RealPublisher creates RealAudio and RealVideo clips which are delivered by the RealServer over the Internet, and presented at the client site by the RealPlayer.

The most outstanding features of RealPublisher are its multi-template encoding, bandwidth negotiation, and synchronization mechanism.

When users encode audio and video clips, they are allowed to encode their files with different compression rates based on their bandwidth capability. Multiple templates generate multiple output files. The disadvantage is that the web site must have a
separate hypertext link and web page for each format or template. However, using bandwidth negotiation option will enable the RealServer to automatically stream the best file for the users' stated bandwidth regardless of the number of the alternative files. This bandwidth negotiation option can be turned on every time when users create a document with the RealPublisher.

To create a synchronized multimedia presentation for the RealSystem 5.0, users have to prepare a plain text file, called Input Events File. This Input Event File has to be merged with a non-synchronized media file to generate the final synchronized presentation file. This is done with the rmmerge.exe tool. In the Input Events File, each synchronization event should indicate its starttime, endtime and URL address.

The RealPublisher supports automatic HTML page creation by its HTML Generation Wizard. The wizard helps create the HTML files and allows users to choose if the file will be displayed in either a Pop-up RealPlayer or an Embedded RealPlayer.

The strength of the RealSystem lies in its capability to create high-quality content at a scalable bandwidth ranging from 10 Kbps to over 500 Kbps, and to deliver streaming content reliably under real-world conditions. As the RealNetworks is working with SUN to merge the RealSystem codec into JMF implementation, the performance of WebSmart presentations will be improved by compressing media streams using the RealSystem codec.

5.3 NetShow

The NetShow [23] is Microsoft's solution to an end-to-end system, server, client and tools that allows users to stream multimedia over intranets and the Internet. The key
idea here is a form of content called **Active Streaming Format** (.asf) files. These files sit on a NetShow server and are streamed to the clients that request the files. These files are created by Microsoft’s content creation tools and embedded within HTML files. At the clients’s side, these files can be viewed in a browser and presented by a **NetShow Player**.

**ASF Editor** is one of the .asf file creation tools. It assembles and synchronizes various types of content, and compresses the files to fit the source material into different bandwidth restrictions.

The NetShow installation includes several audio, image and video codecs such as **MPEG Layer-3, L&H, Voxware MetaSound, Voxware RT24/RT29** for audios, and **H.263, Microsoft MPEG4, VDOnet VDOWave**, **Duck trueMotiunRT**, and **Intel H.263** for videos. Users can select a codec as they create .asf files.

The ASF Editor combines and synchronizes the images, audio, video and scripts commands on a graphic timeline representation. Several options are provided for the synchronization:

- **Markers**: Markers are like bookmarks in the .asf file. NetShow Player uses markers to jump around in the .asf file. This is particularly useful when viewers are watching a big video file, markers help them to jump from section to section.

- **Script commands**: Visual Basic Scripts and JavaScript scripts are used to interleave events into the streaming .asf file. As the NetShow Player receives the .asf file, the script commands are passed to a client-side application via the ScriptCommand event that handles the commands and executes the actions. Script commands are sent in a table at the .asf file header.
• **URL flips:** An .asf file can be embedded into a framed HTML page. ASF Editor provides the interface to arrange URLs appearing in accompanying frames in a synchronized manner as the media frame is presenting the audio/video clip.

• **Captions:** caption is another form of additional information to be embedded in the .asf file. Users can specify a time for the display of caption, and enter the text of the caption in the ASF Editor.

In summary, NetShow shows strengths in organizing the synchronization information and fitting the multimedia material into the restricted bandwidths. The synchronization options are similar to those in WebSmart. However, the ASF Editor still needs more flexibility in page layout designing.

### 5.4 SMIL

The World Wide Web Consortium has been working on providing a common solution for integrating synchronized multimedia presentations into the Web. In November 1997, it released the first public working draft of the Synchronized Multimedia Integration Language (SMIL) [24]. Instead of using scripting languages as in most commercial products, it adopts a declarative format for expressing media synchronization on the Web.

SMIL is a markup language just as HTML, which can be edited by an ordinary text editor. Each SMIL document has a head and a body. The head part contains a layout section that determines the placement of each component on the web page, its position, size and scaling of a visual media object within a rendering window. In the body part, each component uses a schedule element to specify its temporal behavior. The synchro-
nization between elements can be in either parallel or sequential mode. In both modes, users define the following attributes of the schedule element: its media type (audio, video, text or image), source address (URL address), begin time, and end time of presentation etc. A rich set of options is available for defining the synchronization behavior among elements. These options include whether children of a parallel group keep their independent clock or share a common clock, and whether the display of an element ends when its duration has ended or the display of it freezes on the screen, etc. To improve the flexibility in performing at various transmission speeds, SMIL provides authors with a switch mechanism to specify a set of alternative elements from which to choose. The attributes which can be switched are bit rate, language, screen size, and screen depth etc.

SMIL is a comprehensive and flexible solution to the multimedia documentation. The problem is whether this SMIL specification will be implemented as a real standard, and how it will influence the current market where different multimedia formats exist. At the time of this writing, there is only one experimental implementation [27] which realized half of the SMIL features.

5.5 Summary

From the above discussion, we conclude that present multimedia authoring systems have provided with practical mechanisms to assemble and synchronize multimedia content within a web page, and also made progresses in playing back video streams with the Internet bandwidth limitation. Yet the incompatibility among the media formats hinders the distribution of documents within the heterogeneous Internet environ-
ment. SMIL is a good solution to a unified multimedia documentation, but whether and when it will be widely applied are still pending.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

The emergence of interactive multimedia presentations on the World Wide Web brought about both wide applications and big challenges. To enable video over the Internet, solutions need to be provided to deal with problems in bandwidth, synchronization, presentation and authoring environment. Many companies and organizations are working on new standards and products. Yet the heterogeneity of the Internet calls for open applications to work seamlessly on various platforms.

WebSmart is a research project targeted at creating web-oriented synchronized multimedia presentations. It consists of a presentation engine and an authoring environment. The presentation engine allows playback of video content within a web browser without additional plug-ins, and the authoring environment supports different levels of users to create multimedia-embedded documents with built-in synchronization mechanisms.

WebSmart uses a simple document model to create multimedia web presentations. Designed primarily for distant learning applications, it is able to generate three kinds of components for a presentation: Table of Content, Media, and Accompanying HTML pages. These components correspond instinctively to a real classroom course’s elements such as syllabus, instructor’s narration and overhead, therefore they are easy
to understand by presentation authors. Besides, these components can also be used in constructing other applications such as corporate training and on-line publication/advertisement.

WebSmart is designed to adapt to different levels of users. For inexperienced web page authors, presentations can be generated exclusively in WebSmart's authoring tools. These tools include a *wsEditor* for assembling all components in a presentation, a *CaptionMaker* for generating Closed Captions, and a *QuizDesigner* for creating quiz sections which embed more intensive interactions than ordinary instruction sections. These tools allow authors to design their documents in a WYSIWYG manner. For experienced authors, WebSmart defined a set of statements for creating customized synchronization mechanisms. These simple statements are embedded in JavaScript functions, therefore authors do not need to learn extra scripting languages.

To address the timing management problem in multimedia presentations, WebSmart realized two levels of inter-object synchronizations: *page flipping* and *lip sync*. Page flipping arranges URLs appearing in Accompanying HTML pages at pre-defined timing points as the *Media* frame is presenting the audio/video clip. The lip sync realizes a finer-grained form of synchronization. When the *Media* frame is presenting the audio/video clip, a text field below that frame displays the content of the narration in a pace corresponding to the movement of the narrator's lips. With WebSmart's original algorithm, these two levels of synchronization prevent the multiple components within a web page to drift in time away from each other, and present the time-sensitive media (audio/video) in a consistent manner. Both kinds of synchronization events can be generated visually in WebSmart authoring tools, and they are manipulated by the presen-
The WebSmart system is built in Object-Oriented methodology for easy maintenance and extension. By leveraging the latest progress in Java, Java Media Framework, HTML, and JavaScript, WebSmart tries to provide an open solution which is easy to incorporate future development. The authoring system runs seamlessly on both Unix and Windows95/NT platforms, and the presentations it creates are accessible from within ordinary web browsers. An HTTP server and a distant learning case have been built to test the performance of the system.

Although enhancement has been made to reduce the traffic put into the network and improve the performance of accessing multimedia documents via the Internet, more work is expected on scalable compression algorithms, specialized multimedia server and the network. Future work includes improving the functionality of WebSmart using Dynamic HTML, and accommodate live capturing and conferencing into the authoring environment.

6.2 Future Work

JMF is working on live capturing and conferencing specifications. Once these specifications are finalized, WebSmart can accommodate them into the current authoring environment.

Another potential extension to WebSmart is to improve the functionality of the system using the latest development in Dynamic HTML.

Despite of its wide applications on the web, current HTML is described by Bosak [33] as having following limitations:
• **Extensibility.** HTML does not allow users to specify their own tags or attributes in order to parameterize or otherwise semantically qualify their data.

• **Structure.** HTML does not support the specification of deep structures needed to represent database schemas or object-oriented hierarchies.

• **Validation.** HTML does not support the kind of language specification that allows consuming applications to check data for structural validity on importation.

The World Wide Web Consortium is working on creating new languages and interfaces to address the above problems. Each of these new elements works on a slightly different set of HTML problems: Extensible Markup Language (XML) on helping organizing and find data, Cascading Style Sheets (CSS) on Web page inheritance and presentation, and Document Object Model (DOM) on manipulating the content. CSS, combined with HTML 4.0, scripts, and DOM, is termed as **Dynamic HTML (DHTML)** by some vendors to create dynamic web pages.

6.2.1 Extensible Markup Language (XML)

As defined in W3C’s Recommendation 1.0 [31], XML “describes a class of data objects call XML documents and partially describes the behavior of computer programs which process them”. It is designed to make it easy and straightforward to use SGML on the Web.

**Standard Generalized Markup Language (SGML)** (ISO 8879) [34], is the international standard for defining descriptions of the structure and content of different types of electronic documents. SGML specifies grammars for document markup languages, and offers functions not available in HTML, such as the three listed above (Ex-
tensibility, Structure and Validation). HTML itself is based on SGML, with only a limited set of hard-wired tags implemented. The major problems of SMGL are its large, cumbersome specification and difficulty in learning and implementation of the language syntax.

XML is designed to bridge the gap between SGML’s richness and HTML’s ease of use in Web applications. The style and structure of an XML document look similar to an HTML document, except that the XML file can uses Document Type Definition (DTD) to specify the logical structure of a document. When the web server with XML content prepares data for transmission, it generates a context wrapper pointing to the associated DTD. Web clients, when unpacking the XML document, parse the content according to the DTD. One example of how a WebSmart presentation will look like when using XML coding is demonstrated below.

```xml
<WS_PRESENTATION>
  <NAME>A JMF Web Course</NAME>
  <SECTION>
    <NUMBER>1</NUMBER>
    <TYPE>Instruction</TYPE>
    <NAME>Introduction</NAME>
    <VIDEO_FILE>movies/sec11_l.mpg</VIDEO_FILE>
    <AUDIO_FILE>movies/sec11_l.mpa</AUDIO_FILE>
  </SECTION>
  <SECTION>
    <NUMBER>2</NUMBER>
    <TYPE>Quiz</TYPE>
    <NAME>A sample quiz</NAME>
</WS_PRESENTATION>
```
In this example, the presentation is named by "A JMF Web Course", and has two sections. Each section has a section number, type (Instruction or Quiz), a name, a video clip and an audio clip. A Quiz type section also includes one or more questions. Each question has a question body and an answer. Notice here that the tags in the XML file correspond instinctively with the logical structure of the presentation itself, instead of the FRAMESETs and FRAMES in original HTML files. What these tags will look like when you view it is up to the DTD, and the appearance of the page within browsers can be specified using CSS.

6.2.2 Cascading Style Sheet (CSS)

While XML helps organizing the data, style sheet describes how documents are presented in a Web browser, for instance, color, font, position, size and format. Some advantages of CSS [30] which can be applied to WebSmart are as follows:

- Positioning.
HTML uses FRAMESET and FRAME to layout components with sizes in percentage of the width and height of the entire page. WebSmart has to use a special layout manager (Ratio_Layout) to provide flexibility in adjusting the position and size of a component. CSS, however, places each component in an invisible rectangle whose width and height and distance from the top/left edges of the browser window can be specified in pixel's precision. Page authors are therefore allowed to dispose components in precise coordinates. Besides, as the term cascading style sheets implies, more than one style sheet can be used on the same document, with different levels of importance.

- Font.

HTML does not support font specification in a document, users must find workaround to change fonts for the presentation, as WebSmart does by defining special Java functions. CSS, however, provides explicit specification in page fonts, frees page authors from dealing with fonts outside of the HTML file.

6.2.3 Document Object Model (DOM)

The Document Object Model (DOM) [32] is a “platform- and language-neutral program interface that will allow programs and scripts to access and update the content, structure, and style of documents in a standard way”. Currently there are several vendors offering software to embed scripts in documents for content accessing and updating, but all in a non-standard interface which prohibits authors from using these documents in an interoperable manner. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of
how these objects can be combined, and a standard interface for accessing and manipulating them. Vendors can support the DOM as an interface to their proprietary data structures and APIs, and content authors can write to the standard DOM interfaces rather than product-specific APIs. As a consequence the interoperability is increased on the Web. With regard to WebSmart, once its XML presentation conforms to the standards specified by DOM, it can inter-operate with other web objects as long as the latter conforms to DOM as well.
Bibliography


83


[31] Extensible Markup Language (XML) 1.0, W3C Recommendation 10-February-1998 http://www.w3.org/TR/REC-xml


84


Appendix A

Appendix A lists source code for section1.html.

<HTML>
<HEAD>
<TITLE>Course Overview</TITLE>
</HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#990000" VLINK="#333399" ALINK="#333399" OnLoad="loadEvents()">
<H2>1 Course Overview</H2>
<applet name = "clock" code=Display.class width=320 height=350 MAYSRCRIPT>
blockquote>
	<em>Your browser doesn’t understand the APPLET tag.</em>
Here’s a screenshot of the clock applet that you
would see running here if it did:
</blockquote>
</applet>

<SCRIPT>
function changetitle(n) {
    parent.frames["title"].location=n;
}
</SCRIPT>

86
function changeNotes(n) {
  parent.frames["Notes"].location=n;
}

function loadEvents()
{
  var APP = document.applets["clock"];  
  APP.clearAllEvents();

  APP.addEvent("changetitle", "sec_title0.html", "movies/sec0.mpg", "0");
  APP.addEvent("changeNotes", "slide0.html", "movies/sec0.mpg", "34");
  APP.addEvent("changeNotes", "slide1.html", "movies/sec0.mpg", "46");

  APP.endEvent();
}
</SCRIPT>
</BODY>
</HTML>
Appendix B

Appendix B lists source code for section3.html. Section3 is a quiz section.

<HTML>
<HEAD>
<TITLE>Quiz</TITLE>
</HEAD>

<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#990000" VLINK="#333399" ALINK="#333399" OnLoad="loadEvents()">
<H2>3 Quiz</H2>
<applet name="clock" code=Display.class width=320 height=350 MAYSCRIPT>
<blockquote>
<hr>
<em>Your browser doesn't understand the APPLET tag. Here's a screenshot of the clock applet that you would see running here if it did:</em>
<p><hr></p>
</blockquote>

<PARAM name="VIDEO-FILE" value="movies/secll_l.mpg">
<PARAM name="AUDIO-FILE" value="movies/secll_l.mpa">
<PARAM name="IMAGE-FILE" value="movies/secll_l.gif">
<PARAM name="CAPTION-FILE" value="movies/secll_l.cap">

<SCRIPT>
function changetitle(n) {
    parent.frames["title"].location=n;
}

</SCRIPT>

88
function changeNotes(n) {
    parent.frames["Notes"].location=n;
}

function pauseApp() {
    var APP = document.applets["clock"];  
    APP.appletPause();
}

function loadEvents(){
    var APP = document.applets["clock"];  
    APP.clearAllEvents();
    APP.quizReset();
    APP.addEvent("changetitle", "sec_title4.html", "movies/sec11_1.mpg", "2");
    APP.addEvent("changeNotes", "q1.html", "movies/sec11_1.mpg", "11");
    APP.addEvent("changeNotes", "q2.html", "movies/sec11_1.mpg", "16");
    APP.addEvent("changeNotes", "q3.html", "movies/sec11_1.mpg", "32");
    APP.endEvent();
}

</SCRIPT>
</BODY>
</HTML>
Appendix C

Appendix C lists source code for `q1.html`.

```html
<html>
<head>
<script>

function loadEvents() {
    var APP=parent.frames["media_app"].document.applets["clock"];
    APP.addEvent("pauseApp","","movies/sec11_1.mpg","13");
}

function check_answer() {
    var correct_answer="Java Media Framework";
    var your_answer=document.quiz.answer.value;
    var APP=parent.frames["media_app"].document.applets["clock"];
    APP.one_more_quiz();
    if (your_answer == correct_answer) {
        APP.one_more_correct_answer();
        document.quiz.result.value="correct!";
    } else {
        document.quiz.result.value="wrong!";
        APP.appletContinue();
    }

</script>
</head>
```
What does JMF stand for?

What does JMF stand for?